

z/OS  
Cryptographic Services  
Integrated Cryptographic Service Facility



# Application Programmer's Guide

**Note!**

Before using this information and the product it supports, be sure to read the general information in the “Notices” on page 979.

This edition replaces SA22-7522-15

© **Copyright IBM Corporation 1997, 2012.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Figures</b>	<b>xv</b>
----------------	-----------

<b>Tables</b>	<b>xvii</b>
---------------	-------------

<b>About this information</b>	<b>xxiii</b>
-------------------------------	--------------

Who should use this information	xxiii
How to use this information	xxiii
Where to find more information	xxv
Related Publications	xxv

<b>How to send your comments to IBM</b>	<b>xxvii</b>
---	--------------

If you have a technical problem.	xxvii
----------------------------------	-------

<b>Summary of changes.</b>	<b>xxix</b>
----------------------------	-------------

Changes made in Cryptographic Support for z/OS V1R12-R13 (FMID HCR77A0).	xxix
Changes made in Cryptographic Support for z/OS V1R11-R13 (FMID HCR7790)	xxx
Changes made in Cryptographic Support for z/OS V1R10-R12 (FMID HCR7780)	xxxiii
Changes made in Cryptographic Support for z/OS V1R9-R11 (FMID HCR7770)	xxxvi

## Part 1. IBM CCA Programming. . . . 1

### Chapter 1. Introducing Programming for the IBM CCA. . . . 3

ICSF Callable Services Naming Conventions.	3
Callable Service Syntax	4
Callable Services with ALET Parameters	5
Rules for Defining Parameters and Attributes	5
Parameter Definitions	6
Invocation Requirements	9
Security Considerations.	9
Performance Considerations	10
Special Secure Mode	10
Using the Callable Services	11
When the Call Succeeds	11
When the Call Does Not Succeed	12
Linking a Program with the ICSF Callable Services	13

### Chapter 2. Introducing Symmetric Key Cryptography and Using Symmetric Key Callable Services . . . . 15

Functions of the Symmetric Cryptographic Keys	15
Key Separation	16
Master Key Variant for Fixed-length Tokens	16
Transport Key Variant for Fixed-length Tokens	16
Key Forms.	17
Key Token.	17
Key Wrapping	19
Control Vector for DES Keys.	19
Types of Keys	19

Key Strength and Wrapping of Key	25
Key Strength and Key Wrapping Access Control Points	26
DES Master Key.	27
Generating and Managing Symmetric Keys.	27
Key Generator Utility Program	27
Common Cryptographic Architecture DES Key Management Services	28
Common Cryptographic Architecture AES Key Management Services	32
Common Cryptographic Architecture HMAC Key Management Services	33
ECC Diffie-Hellman Key Agreement Models	34
Improved remote key distribution	35
Diversifying keys	49
Callable Services for Dynamic CKDS Update	49
Callable Services that Support Secure Sockets Layer (SSL)	51
System Encryption Algorithm	52
ANSI X9.17 Key Management Services	53
Enciphering and Deciphering Data	55
Encoding and Decoding Data (CSNBECO, CSNEECO, CSNBDCO, and CSNEDCO).	56
Translating Ciphertext (CSNBCTT or CSNBCTT1 and CSNECTT or CSNECTT1)	56
Translating Ciphertext (CSNBCTT2 or CSNBCTT3 and CSNECTT2 or CSNECTT3).	56
Managing Data Integrity and Message Authentication	56
Message Authentication Code Processing	57
Hashing Functions	58
Managing Personal Authentication	59
Verifying Credit Card Data	60
ANSI TR-31 key block support	62
TR-31 Export Callable Service (CSNB31X and CSNET31X)	63
TR-31 Import Callable Service (CSNB31I and CSNET31I)	63
TR-31 Parse Callable Service (CSNB31P and CSNET31P)	63
TR-31 Optional Data Read Callable Service (CSNB31R and CSNET31R)	63
TR-31 Optional Data Build Callable Service (CSNB31O and CSNET31O)	63
Secure Messaging	64
Trusted Key Entry (TKE) Support	64
Utilities.	65
Character/Nibble Conversion Callable Services (CSNBXBC and CSNBXCB)	65
Code Conversion Callable Services (CSNBXEA and CSNBXAE)	65
X9.9 Data Editing Callable Service (CSNB9ED)	65
ICSF Query Algorithm Callable Service (CSFIQA)	65
ICSF Query Facility Callable Service (CSFIQF).	65
Typical Sequences of ICSF Callable Services	66

Key Forms and Types Used in the Key Generate Callable Service . . . . .	67
Generating an Operational Key . . . . .	67
Generating an Importable Key . . . . .	67
Generating an Exportable Key . . . . .	67
Examples of Single-Length Keys in One Form Only. . . . .	67
Examples of OPIM Single-Length, Double-Length, and Triple-Length Keys in Two Forms . . . . .	68
Examples of OPEX Single-Length, Double-Length, and Triple-Length Keys in Two Forms . . . . .	68
Examples of IMEX Single-Length and Double-Length Keys in Two Forms . . . . .	69
Examples of EXEX Single-Length and Double-Length Keys in Two Forms . . . . .	69
Generating AKEs . . . . .	69
Using the Ciphertext Translate and Ciphertext	
Translate2 Callable Service . . . . .	70
Summary of Callable Services . . . . .	71

## Chapter 3. Introducing PKA Cryptography and Using PKA Callable Services . . . . . 83

PKA Key Algorithms . . . . .	83
PKA Master Keys . . . . .	83
Operational private keys . . . . .	84
Key Strength and Wrapping of Key . . . . .	85
Key Strength and Key Wrapping Access Control Points . . . . .	86
RSA Private Key Tokens . . . . .	87
PKA Callable Services . . . . .	87
Callable Services Supporting Digital Signatures	87
Callable Services for PKA Key Management . . . . .	88
Callable Services to Update the Public Key Data Set (PKDS) . . . . .	89
Callable Services for Working with Retained Private Keys . . . . .	90
Callable Services for SET Secure Electronic Transaction . . . . .	91
PKA Key Tokens . . . . .	91
PKA Key Management . . . . .	92
Security and Integrity of the Token . . . . .	93
Key Identifier for PKA Key Token . . . . .	93
Key Label . . . . .	94
Key Token. . . . .	94
The Transaction Security System and ICSF Portability . . . . .	95
Summary of the PKA Callable Services . . . . .	95

## Chapter 4. Introducing PKCS #11 and using PKCS #11 callable services . . . . 97

PKCS #11 Management Services . . . . .	97
Attribute List. . . . .	98
Handles . . . . .	99

## Part 2. CCA Callable Services . . . 101

## Chapter 5. Managing Symmetric Cryptographic Keys . . . . . 103

Clear Key Import (CSNBCKI and CSNECKI) . . . . .	104
Format . . . . .	104
Parameters . . . . .	104
Usage Notes. . . . .	105
Access Control Points . . . . .	105
Required Hardware . . . . .	105
Control Vector Generate (CSNBCVG and CSNECVG) . . . . .	106
Format . . . . .	106
Parameters . . . . .	106
Usage Notes. . . . .	110
Required Hardware . . . . .	110
Control Vector Translate (CSNBCVT and CSNECVT) . . . . .	110
Format . . . . .	111
Parameters . . . . .	111
Restrictions . . . . .	113
Usage Notes. . . . .	113
Access Control Point . . . . .	113
Required Hardware . . . . .	113
Cryptographic Variable Encipher (CSNBCVE and CSNECVE) . . . . .	114
Format . . . . .	114
Parameters . . . . .	114
Restrictions . . . . .	116
Usage Notes. . . . .	116
Access Control Point . . . . .	116
Required Hardware . . . . .	116
Data Key Export (CSNBKDX and CSNEKDX) . . . . .	116
Format . . . . .	117
Parameters . . . . .	117
Restrictions . . . . .	118
Usage Notes. . . . .	118
Access Control Points. . . . .	118
Required Hardware . . . . .	119
Data Key Import (CSNBKDM and CSNEKDM) . . . . .	119
Format . . . . .	120
Parameters . . . . .	120
Restrictions . . . . .	121
Usage Notes. . . . .	121
Access Control Points . . . . .	121
Required Hardware . . . . .	122
Diversified Key Generate (CSNBKDG and CSNEKDG) . . . . .	122
Format . . . . .	123
Parameters . . . . .	123
Restrictions . . . . .	127
Usage Notes. . . . .	127
Access Control Points . . . . .	127
Required Hardware . . . . .	128
ECC Diffie-Hellman (CSNDEDH and CSNFEDH) . . . . .	129
Format . . . . .	130
Parameters . . . . .	130
Restrictions . . . . .	134
Usage Notes. . . . .	135
Access Control Points . . . . .	135
Required Hardware . . . . .	136
Key Export (CSNBKEX and CSNEKEX) . . . . .	137
Format . . . . .	137

Parameters . . . . .	137	Access Control Point . . . . .	188
Restrictions . . . . .	139	Required Hardware . . . . .	188
Usage Notes — General . . . . .	139	Key Test Extended (CSNBKYTX and CSNEKTX)	188
Usage Notes — CCF Systems . . . . .	139	Format . . . . .	189
Usage Notes — Cryptographic Coprocessor		Parameters . . . . .	189
Systems . . . . .	140	Restrictions . . . . .	191
Access Control Points . . . . .	140	Usage Notes . . . . .	191
Required Hardware . . . . .	141	Access Control Point . . . . .	191
Key Generate (CSNBKGN and CSNEKGN)	142	Required Hardware . . . . .	191
Format . . . . .	142	Key Token Build (CSNBKTB and CSNEKTB)	192
Parameters . . . . .	143	Format . . . . .	193
Restrictions . . . . .	150	Parameters . . . . .	193
Usage Notes — General . . . . .	150	Restrictions . . . . .	199
Usage Notes — CCF systems . . . . .	150	Usage Notes . . . . .	199
Usage Notes - Cryptographic Coprocessor		Required Hardware . . . . .	202
Systems . . . . .	151	Key Token Build2 (CSNBKTB2 and CSNEKTB2)	202
Usage Notes — Key type and key form		Format . . . . .	203
combinations . . . . .	151	Parameters . . . . .	203
Access Control Points . . . . .	153	Required Hardware . . . . .	208
Required Hardware . . . . .	154	Key Translate (CSNBKTR and CSNEKTR)	209
Key Generate2 (CSNBKGN2 and CSNEKGN2)	156	Format . . . . .	209
Format . . . . .	157	Parameters . . . . .	209
Parameters . . . . .	157	Restrictions . . . . .	210
Usage Notes . . . . .	162	Usage Notes . . . . .	210
Access Control Points . . . . .	164	Access Control Points . . . . .	210
Required Hardware . . . . .	165	Required Hardware . . . . .	211
Key Import (CSNBKIM and CSNEKIM)	165	Key Translate2 (CSNBKTR2 and CSNEKTR2)	211
Format . . . . .	166	Format . . . . .	212
Parameters . . . . .	166	Parameters . . . . .	212
Restrictions . . . . .	168	Restrictions . . . . .	216
Usage Notes — General . . . . .	168	Usage Notes . . . . .	216
Usage Notes — CCF Systems . . . . .	168	Access Control Points . . . . .	216
Usage Notes — Cryptographic Coprocessor		Required Hardware . . . . .	217
Systems . . . . .	169	Multiple Clear Key Import (CSNBCKM and	
Access Control Points . . . . .	169	CSNECKM)	218
Required Hardware . . . . .	170	Format . . . . .	218
Key Part Import (CSNBKPI and CSNEKPI)	171	Parameters . . . . .	218
Format . . . . .	172	Usage Notes . . . . .	220
Parameters . . . . .	172	Access Control Points . . . . .	220
Restrictions . . . . .	174	Required Hardware . . . . .	220
Usage Notes . . . . .	174	Multiple Secure Key Import (CSNBCKM and	
Access Control Points . . . . .	174	CSNESKM)	222
Required Hardware . . . . .	175	Format . . . . .	222
Related Information . . . . .	175	Parameters . . . . .	222
Key Part Import2 (CSNBKPI2 and CSNEKPI2)	176	Usage Notes . . . . .	225
Format . . . . .	176	Access Control Points . . . . .	226
Parameters . . . . .	176	Required Hardware . . . . .	226
Usage Notes . . . . .	178	PKA Decrypt (CSNDPKD and CSNFPKD)	228
Access Control Points . . . . .	178	Format . . . . .	229
Required Hardware . . . . .	179	Parameters . . . . .	229
Key Test (CSNBKYT and CSNEKYT)	179	Restrictions . . . . .	231
Format . . . . .	180	Authorization . . . . .	231
Parameters . . . . .	180	Usage Notes — General . . . . .	232
Restrictions . . . . .	182	Usage Notes — CCF Systems . . . . .	232
Usage Notes . . . . .	182	Access Control Points . . . . .	232
Access Control Point . . . . .	183	Required Hardware . . . . .	233
Required Hardware . . . . .	183	PKA Encrypt (CSNDPKE and CSNFPKE)	234
Key Test2 (CSNBKYT2 and CSNEKYT2)	184	Format . . . . .	235
Format . . . . .	184	Parameters . . . . .	235
Parameters . . . . .	184	Restrictions . . . . .	237
Usage Notes . . . . .	187	Usage Notes . . . . .	237

Access Control Point . . . . .	237	Format . . . . .	283
Required Hardware . . . . .	237	Parameters . . . . .	283
Prohibit Export (CSNBPEX and CSNEPEX) . . . . .	239	Restrictions . . . . .	286
Format . . . . .	239	Usage Notes. . . . .	286
Parameters . . . . .	239	Access Control Points . . . . .	286
Usage Notes. . . . .	240	Required Hardware . . . . .	287
Access Control Point . . . . .	240	Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2) . . . . .	291
Required Hardware . . . . .	240	Format . . . . .	291
Prohibit Export Extended (CSNBPEXX and CSNEPEXX) . . . . .	241	Parameters . . . . .	291
Format . . . . .	241	Restrictions . . . . .	293
Parameters . . . . .	241	Usage Notes. . . . .	293
Restrictions . . . . .	242	Access Control Points . . . . .	294
Usage Notes. . . . .	242	Required Hardware . . . . .	294
Access Control Point . . . . .	242	Transform CDMF Key (CSNBTK and CSNETCK) . . . . .	295
Required Hardware . . . . .	242	Format . . . . .	296
Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL) . . . . .	243	Parameters . . . . .	296
Format . . . . .	243	Restrictions . . . . .	297
Parameters . . . . .	244	Usage Notes. . . . .	297
Usage Notes. . . . .	246	Required Hardware . . . . .	297
Required Hardware . . . . .	246	Trusted Block Create (CSNDTBC and CSNFTBC) . . . . .	298
Remote Key Export (CSNDRKX and CSNFRKX) . . . . .	247	Format . . . . .	298
Format . . . . .	248	Parameters . . . . .	298
Parameters . . . . .	248	Usage Notes. . . . .	300
Usage Notes. . . . .	254	Access Control Points . . . . .	300
Access Control Points . . . . .	254	Required Hardware . . . . .	301
Required Hardware . . . . .	254	TR-31 Export (CSNBT31X and CSNET31X) . . . . .	301
Restrict Key Attribute (CSNBRKA and CSNERKA) . . . . .	255	Format . . . . .	302
Format . . . . .	256	Parameters . . . . .	302
Parameters . . . . .	256	Restrictions . . . . .	307
Access Control Points . . . . .	259	Usage Notes. . . . .	307
Required Hardware . . . . .	259	Access Control Points . . . . .	307
Secure Key Import (CSNBSKI and CSNESKI) . . . . .	260	Required Hardware . . . . .	315
Format . . . . .	261	TR-31 Import (CSNBT31I and CSNET31I) . . . . .	315
Parameters . . . . .	261	Format . . . . .	316
Usage Notes — General . . . . .	263	Parameters . . . . .	316
Usage Notes — CCF Systems . . . . .	263	Restrictions . . . . .	321
Access Control Points . . . . .	263	Usage Notes. . . . .	321
Required Hardware . . . . .	263	Access Control Points . . . . .	323
Secure Key Import2 (CSNBSKI2 and CSNESKI2) . . . . .	264	Required Hardware . . . . .	328
Format . . . . .	265	TR-31 Optional Data Build (CSNBT31O and CSNET31O) . . . . .	329
Parameters . . . . .	265	Format . . . . .	329
Access Control Points . . . . .	268	Parameters . . . . .	329
Required Hardware . . . . .	268	Restrictions . . . . .	331
Symmetric Key Export (CSNDSYX and CSNFSYX) . . . . .	269	Usage Notes. . . . .	331
Format . . . . .	269	Required Hardware . . . . .	331
Parameters . . . . .	269	TR-31 Optional Data Read (CSNBT31R and CSNET31R) . . . . .	332
Restrictions . . . . .	272	Format . . . . .	332
Usage Notes. . . . .	272	Parameters . . . . .	332
Access Control Points . . . . .	272	Restrictions . . . . .	335
Required Hardware . . . . .	273	Usage Notes. . . . .	335
Symmetric Key Generate (CSNDSYG and CSNFSYG) . . . . .	275	Required Hardware . . . . .	335
Format . . . . .	275	TR-31 Parse (CSNBT31P and CSNET31P) . . . . .	336
Parameters . . . . .	275	Format . . . . .	336
Restrictions . . . . .	279	Parameters . . . . .	336
Usage Notes. . . . .	279	Restrictions . . . . .	338
Access Control Points . . . . .	280	Usage Notes. . . . .	338
Required Hardware . . . . .	280	Required Hardware . . . . .	339
Symmetric Key Import (CSNDSYI and CSNFSYI) . . . . .	283	Unique Key Derive (CSNBUKD and CSNEUKD) . . . . .	339

	Format . . . . .	340
	Parameters . . . . .	340
	Restrictions . . . . .	345
	Usage Notes. . . . .	345
	Access Control Points . . . . .	346
	Required Hardware . . . . .	346
	User Derived Key (CSFUDK and CSFUDK6) . . . . .	347
	Format . . . . .	347
	Parameters . . . . .	347
	Usage Notes. . . . .	349
	Required Hardware . . . . .	350
<b>Chapter 6. Protecting Data. . . . .</b>		<b>351</b>
	Modes of Operation . . . . .	351
	Electronic Code Book (ECB) Mode . . . . .	352
	Cipher Block Chaining (CBC) Mode. . . . .	352
	Cipher Feedback (CFB) Mode . . . . .	352
	Output Feedback (OFB) Mode. . . . .	352
	Galois/Counter Mode (GCM) . . . . .	353
	Triple DES Encryption . . . . .	353
	Ciphertext Translate (CSNBCTT or CSNBCTT1 and CSNECTT or CSNECTT1) . . . . .	354
	Choosing Between CSNBCTT and CSNBCTT1 . . . . .	354
	Format . . . . .	355
	Parameters . . . . .	355
	Restrictions . . . . .	357
	Usage Notes. . . . .	357
	Required Hardware . . . . .	357
	Ciphertext Translate2 (CSNBCTT2, CSNBCTT3, CSNECTT2, CSNECTT3) . . . . .	357
	Choosing Between CSNBCTT2 and CSNBCTT3 . . . . .	358
	Format . . . . .	359
	Parameters . . . . .	359
	Usage Notes. . . . .	364
	Access Control Points . . . . .	369
	Required Hardware . . . . .	370
	Decipher (CSNBDEC or CSNBDEC1 and CSNEDEC or CSNEDEC1) . . . . .	370
	Choosing Between CSNBDEC and CSNBDEC1 . . . . .	371
	Format . . . . .	372
	Parameters . . . . .	372
	Restrictions . . . . .	376
	Usage Notes. . . . .	376
	Access Control Point . . . . .	377
	Required Hardware . . . . .	377
	Decode (CSNBDCO and CSNEDCO) . . . . .	377
	Considerations . . . . .	377
	Format . . . . .	378
	Parameters . . . . .	378
	Restrictions . . . . .	379
	Required Hardware . . . . .	379
	Encipher (CSNBENC or CSNBENC1 and CSNEENC or CSNEENC1) . . . . .	379
	Choosing between CSNBENC and CSNBENC1 . . . . .	381
	Format . . . . .	381
	Parameters . . . . .	382
	Restrictions . . . . .	386
	Usage Notes. . . . .	386
	Access Control Point . . . . .	386
	Required Hardware . . . . .	386
	Related Information . . . . .	387

Encode (CSNBECO and CSNEECO) . . . . .	387
Considerations . . . . .	387
Format . . . . .	388
Parameters . . . . .	388
Restrictions . . . . .	389
Required Hardware . . . . .	389
Symmetric Algorithm Decipher (CSNBSAD or CSNBSAD1 and CSNESAD or CSNESAD1) . . . . .	389
Choosing Between CSNBSAD and CSNBSAD1 or CSNESAD and CSNESAD1. . . . .	390
Format . . . . .	390
Parameters . . . . .	391
Usage Notes. . . . .	395
Access Control Point . . . . .	395
Required Hardware . . . . .	395
Symmetric Algorithm Encipher (CSNBSE or CSNBSE1 and CSNESAE or CSNESAE1). . . . .	396
Choosing between CSNBSE and CSNBSE1 or CSNESAE and CSNESAE1 . . . . .	396
Format . . . . .	397
Parameters . . . . .	397
Usage Notes. . . . .	401
Access Control Point . . . . .	401
Required Hardware . . . . .	401
Symmetric Key Decipher (CSNBKYD or CSNBKYD1 and CSNESYD or CSNESYD1). . . . .	402
Choosing Between CSNBKYD and CSNBKYD1 . . . . .	404
Format . . . . .	404
Parameters . . . . .	405
Usage Notes. . . . .	410
Access Control Points . . . . .	410
Required Hardware . . . . .	411
Related Information . . . . .	411
Symmetric Key Encipher (CSNBKYE or CSNBKYE1 and CSNESYE or CSNESYE1) . . . . .	412
Choosing between CSNBKYE and CSNBKYE1 . . . . .	413
Format . . . . .	414
Parameters . . . . .	414
Usage Notes. . . . .	420
Access Control Points . . . . .	420
Required Hardware . . . . .	420
Related Information . . . . .	421

<b>Chapter 7. Verifying Data Integrity and Authenticating Messages . . . . .</b>		<b>423</b>
How MACs are Used. . . . .		423
How Hashing Functions Are Used . . . . .		424
How MDCs Are Used . . . . .		424
HMAC Generate (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1) . . . . .		425
Choosing Between CSNBHMG and CSNBHMG1 . . . . .		425
Format . . . . .		426
Parameters . . . . .		426
Access Control Points . . . . .		429
Required Hardware . . . . .		429
HMAC Verify (CSNBHVM or CSNBHVM1 and CSNEHVM or CSNEHVM1) . . . . .		429
Choosing Between CSNBHVM and CSNBHVM1 . . . . .		430
Format . . . . .		430
Parameters . . . . .		430
Access Control Points . . . . .		433



Required Hardware . . . . .	433
MAC Generate (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1) . . . . .	434
Choosing Between CSNBMGN and CSNBMGN1	434
Format . . . . .	435
Parameters . . . . .	435
Usage Notes — General . . . . .	438
Usage Notes — CCF Systems . . . . .	438
Access Control Point . . . . .	438
Required Hardware . . . . .	438
Related Information . . . . .	439
MAC Verify (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1) . . . . .	440
Choosing Between CSNBMVR and CSNBMVR1	440
Format . . . . .	440
Parameters . . . . .	441
Usage Notes — General . . . . .	444
Usage Notes — CCF Systems . . . . .	444
Access Control Point . . . . .	444
Required Hardware . . . . .	444
Related Information . . . . .	445
MDC Generate (CSNBMDG or CSNBMDG1 and CSNEMDG or CSNEMDG1) . . . . .	445
Choosing Between CSNBMDG and CSNBMDG1	445
Format . . . . .	446
Parameters . . . . .	446
Usage Notes . . . . .	449
Required Hardware . . . . .	449
One-Way Hash Generate (CSNBOWH or CSNBOWH1 and CSNEOWH or CSNEOWH1) . . . . .	450
Format . . . . .	450
Parameters . . . . .	450
Usage Notes . . . . .	453
Required Hardware . . . . .	453
Symmetric MAC Generate (CSNBSMG or CSNBSMG1 and CSNESMG or CSNESMG1) . . . . .	454
Choosing Between CSNBSMG and CSNBSMG1 or CSNESMG and CSNESMG1 . . . . .	454
Format . . . . .	455
Parameters . . . . .	455
Required Hardware . . . . .	458
Symmetric MAC Verify (CSNBSMV or CSNBSMV1 and CSNESMV or CSNESMV1) . . . . .	459
Choosing Between CSNBSMV and CSNBSMV1 or CSNESMV and CSNESMV1 . . . . .	459
Format . . . . .	459
Parameters . . . . .	460
Required Hardware . . . . .	463

## Chapter 8. Financial Services . . . . . 465

How Personal Identification Numbers (PINs) are Used . . . . .	465
How VISA Card Verification Values Are Used . . . . .	465
Translating Data and PINs in Networks . . . . .	466
Working with Europay–MasterCard–Visa smart cards . . . . .	466
PIN Callable Services . . . . .	467
Generating a PIN . . . . .	467
Encrypting a PIN . . . . .	467
Generating a PIN Validation Value from an Encrypted PIN Block . . . . .	467

Verifying a PIN. . . . .	467
Translating a PIN . . . . .	467
Algorithms for Generating and Verifying a PIN	467
Using PINs on Different Systems . . . . .	468
PIN-Encrypting Keys . . . . .	468
ANSI X9.8 PIN Restrictions . . . . .	469
ANSI X9.8 PIN - Enforce PIN block restrictions	469
ANSI X9.8 PIN - Allow modification of PAN	470
ANSI X9.8 PIN - Allow only ANSI PIN blocks	470
ANSI X9.8 PIN – Use stored decimalization tables only . . . . .	470
The PIN Profile . . . . .	471
PIN Block Format . . . . .	471
Enhanced PIN Security Mode . . . . .	473
Format Control . . . . .	474
Pad Digit . . . . .	475
Current Key Serial Number . . . . .	476
Decimalization Tables . . . . .	476
Clear PIN Encrypt (CSNBCPE and CSNECPE) . . . . .	476
Format . . . . .	477
Parameters . . . . .	477
Restrictions . . . . .	479
Usage Notes . . . . .	479
Access Control Point . . . . .	479
Required Hardware . . . . .	479
Clear PIN Generate (CSNBPGN and CSNEPGN) . . . . .	480
Format . . . . .	481
Parameters . . . . .	481
Restrictions . . . . .	484
Usage Notes . . . . .	484
Access Control Points . . . . .	484
Required Hardware . . . . .	484
Related Information . . . . .	485
Clear PIN Generate Alternate (CSNBCPA and CSNECPA) . . . . .	485
Format . . . . .	486
Parameters . . . . .	486
Restrictions . . . . .	489
Usage Notes . . . . .	490
Access Control Points . . . . .	490
Required Hardware . . . . .	491
CVV Key Combine (CSNBCKC and CSNECKC) . . . . .	491
Format . . . . .	492
Parameters . . . . .	492
Restrictions . . . . .	494
Usage Notes . . . . .	494
Access Control Points . . . . .	496
Required Hardware . . . . .	496
Encrypted PIN Generate (CSNBEPG and CSNEEPG) . . . . .	497
Format . . . . .	497
Parameters . . . . .	497
Restrictions . . . . .	500
Usage Notes . . . . .	500
Access Control Points . . . . .	501
Required Hardware . . . . .	501
Encrypted PIN Translate (CSNBPTR and CSNEPTR) . . . . .	502
Format . . . . .	502
Parameters . . . . .	503
Restrictions . . . . .	506



Usage Notes . . . . .	507
Access Control Points . . . . .	507
Required Hardware . . . . .	508
Encrypted PIN Verify (CSNBPVR and CSNEPVR)	509
Format . . . . .	509
Parameters . . . . .	510
Restrictions . . . . .	513
Usage Notes . . . . .	514
Access Control Points . . . . .	514
Required Hardware . . . . .	515
Related Information . . . . .	516
PIN Change/Unblock (CSNBPCU and CSNEPCU)	516
Format . . . . .	517
Parameters . . . . .	517
Usage Notes . . . . .	521
Access Control Points . . . . .	521
Required Hardware . . . . .	522
Secure Messaging for Keys (CSNBSKY and CSNESKY)	523
Format . . . . .	523
Parameters . . . . .	523
Usage Notes . . . . .	525
Access Control Point . . . . .	525
Required Hardware . . . . .	526
Secure Messaging for PINs (CSNBSPN and CSNESPN)	526
Format . . . . .	527
Parameters . . . . .	527
Usage Notes . . . . .	530
Access Control Point . . . . .	530
Required Hardware . . . . .	530
SET Block Compose (CSNDSBC and CSNFSBC)	531
Format . . . . .	531
Parameters . . . . .	531
Restrictions . . . . .	535
Usage Notes . . . . .	535
Access Control Point . . . . .	535
Required Hardware . . . . .	535
SET Block Decompose (CSNDSBD and CSNFSBD)	536
Format . . . . .	537
Parameters . . . . .	537
Restrictions . . . . .	541
Usage Notes . . . . .	541
Access Control Points . . . . .	541
Required Hardware . . . . .	541
Transaction Validation (CSNBTRV and CSNETRV)	542
Format . . . . .	543
Parameters . . . . .	543
Usage Notes . . . . .	545
Access Control Points . . . . .	545
Required Hardware . . . . .	546
VISA CVV Service Generate (CSNBCSG and CSNECSG)	546
Format . . . . .	547
Parameters . . . . .	547
Restrictions . . . . .	550
Usage Notes . . . . .	550
Access Control Point . . . . .	550
Required Hardware . . . . .	550
VISA CVV Service Verify (CSNBCSV and CSNECSV)	551

Format . . . . .	551
Parameters . . . . .	551
Restrictions . . . . .	554
Usage Notes . . . . .	554
Access Control Points . . . . .	554
Required Hardware . . . . .	555

## Chapter 9. Using Digital Signatures 557

Digital Signature Generate (CSNDDSG and CSNFDSG)	557
Format . . . . .	558
Parameters . . . . .	558
Restrictions . . . . .	561
Authorization . . . . .	561
Usage Notes . . . . .	561
Required Hardware . . . . .	562
Digital Signature Verify (CSNDDSV and CSNFDSV)	566
Format . . . . .	566
Parameters . . . . .	566
Restrictions . . . . .	569
Usage Notes . . . . .	569
Access Control Point . . . . .	569
Required Hardware . . . . .	569

## Chapter 10. Managing PKA Cryptographic Keys 571

PKA Key Generate (CSNDPKG and CSNFPKG)	571
Format . . . . .	572
Parameters . . . . .	572
Restrictions . . . . .	576
Usage Notes . . . . .	576
Access Control Points . . . . .	576
Required Hardware . . . . .	576
PKA Key Import (CSNDPKI and CSNFPKI)	578
Format . . . . .	578
Parameters . . . . .	578
Restrictions . . . . .	581
Usage Notes . . . . .	581
Access Control Points . . . . .	581
Required Hardware . . . . .	581
PKA Key Token Build (CSNDPKB and CSNFPKB)	583
Format . . . . .	584
Parameters . . . . .	584
Usage Notes . . . . .	595
Required Hardware . . . . .	595
PKA Key Token Change (CSNDKTC and CSNFKTC)	596
Format . . . . .	596
Parameters . . . . .	597
Usage Notes . . . . .	598
Access Control Points . . . . .	598
Required Hardware . . . . .	598
PKA Key Translate (CSNDPKT and CSNFPKT)	599
Format . . . . .	600
Parameters . . . . .	600
Restrictions . . . . .	602
Access Control Points . . . . .	602
Required Hardware . . . . .	603

PKA Public Key Extract (CSNDPKX and CSNFPKX)	604
Format	604
Parameters	604
Usage Notes	605
Required Hardware	605
Retained Key Delete (CSNDRKD and CSNFRKD)	606
Format	606
Parameters	606
Usage Notes	607
Access Control Point	608
Required Hardware	608
Retained Key List (CSNDRKL and CSNFRKL)	609
Format	609
Parameters	609
Usage Notes	611
Access Control Points	611
Required Hardware	611

## Chapter 11. Key Data Set Management 613

CKDS Key Record Create (CSNBKRC and CSNEKRC)	613
Format	613
Parameters	613
Restrictions	614
Usage Notes	614
Required Hardware	615
CKDS Key Record Create2 (CSNBKRC2 and CSNEKRC2)	615
Format	615
Parameters	615
Required Hardware	617
CKDS Key Record Delete (CSNBKRD and CSNEKRD)	617
Format	617
Parameters	617
Restrictions	618
Usage Notes	618
Required Hardware	619
CKDS Key Record Read (CSNBKRR and CSNEKRR)	619
Format	619
Parameters	619
Restrictions	620
Usage Notes	620
Required Hardware	620
CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2)	621
Format	621
Parameters	621
Required Hardware	622
CKDS Key Record Write (CSNBKRW and CSNEKRW)	623
Format	623
Parameters	624
Restrictions	624
Usage Notes	625
Required Hardware	625
Related Information	625
CKDS Key Record Write2 (CSNBKRW2 and CSNEKRW2)	626

Format	626
Parameters	626
Usage Notes	627
Required Hardware	627
Coordinated KDS Administration (CSFCRC and CSFCRC6)	628
Format	629
Parameters	629
Usage Notes	631
Required Hardware	631
PKDS Key Record Create (CSNDKRC and CSNFKRC)	632
Format	632
Parameters	632
Usage Notes	633
Required Hardware	633
PKDS Key Record Delete (CSNDKRD and CSNFKRD)	634
Format	634
Parameters	634
Restrictions	635
Usage Notes	635
Required Hardware	636
PKDS Key Record Read (CSNDKRR and CSNFKRR)	636
Format	636
Parameters	637
Usage Notes	638
Required Hardware	638
PKDS Key Record Write (CSNDKRW and CSNFKRW)	638
Format	639
Parameters	639
Restrictions	640
Usage Notes	640
Required Hardware	640

## Chapter 12. Utilities. 643

Character/Nibble Conversion (CSNBXBC and CSNBXCB)	643
Format	643
Parameters	643
Usage Notes	644
Required Hardware	644
Code Conversion (CSNBXEA and CSNBXAE)	645
Format	645
Parameters	646
Usage Notes	647
Required Hardware	647
ICSF Query Algorithm (CSFIQA and CSFIQA6)	647
Format	648
Parameters	648
Usage Notes	651
Required Hardware	651
ICSF Query Facility (CSFIQF and CSFIQF6)	652
Format	652
Parameters	653
Usage Notes	672
Required Hardware	672
X9.9 Data Editing (CSNB9ED)	673
Format	673

Parameters . . . . .	673
Usage Notes. . . . .	674
Required Hardware . . . . .	674

## Chapter 13. Trusted Key Entry

### Workstation Interfaces . . . . . 677

PCI Interface Callable Service (CSFPCI and CSFPCI6). . . . .	677
Format . . . . .	677
Parameters . . . . .	678
Usage Notes. . . . .	681
Required Hardware . . . . .	681
PKSC Interface Callable Service (CSFPKSC) . . . . .	682
Format . . . . .	682
Parameters . . . . .	683
Usage Notes. . . . .	684
Required Hardware . . . . .	684

## Chapter 14. Managing Keys According to the ANSI X9.17 Standard . . . . . 685

ANSI X9.17 EDC Generate (CSNAEGN and CSNGEGN) . . . . .	685
Format . . . . .	685
Parameters . . . . .	685
Usage Notes. . . . .	687
Required Hardware . . . . .	687
ANSI X9.17 Key Export (CSNAKEX and CSNGKEX) . . . . .	687
Format . . . . .	688
Parameters . . . . .	688
Usage Notes. . . . .	692
Required Hardware . . . . .	692
ANSI X9.17 Key Import (CSNAKIM and CSNGKIM) . . . . .	692
Format . . . . .	693
Parameters . . . . .	693
Usage Notes. . . . .	697
Required Hardware . . . . .	697
ANSI X9.17 Key Translate (CSNAKTR and CSNGKTR) . . . . .	697
Format . . . . .	698
Parameters . . . . .	698
Usage Notes. . . . .	702
Required Hardware . . . . .	702
ANSI X9.17 Transport Key Partial Notarize (CSNATKN and CSNGTKN) . . . . .	702
Format . . . . .	703
Parameters . . . . .	703
Usage Notes. . . . .	704
Required Hardware . . . . .	704

## Part 3. PKCS #11 Callable Services 707

### Chapter 15. Using PKCS #11 Tokens and Objects . . . . . 709

PKCS #11 Derive multiple keys (CSFPDMK and CSFPDMK6). . . . .	709
Format . . . . .	710
Parameters . . . . .	710

Authorization . . . . .	715
Usage Notes. . . . .	716
PKCS #11 Derive key (CSFPDVK and CSFPDVK6) . . . . .	717
Format . . . . .	717
Parameters . . . . .	717
Authorization . . . . .	721
Usage Notes. . . . .	722
PKCS #11 Get attribute value (CSFPGAV and CSFPGAV6) . . . . .	722
Format . . . . .	723
Parameters . . . . .	723
Authorization . . . . .	724
Usage Notes. . . . .	725
PKCS #11 Generate key pair (CSFPGKP and CSFPGKP6) . . . . .	725
Format . . . . .	726
Parameters . . . . .	726
Authorization . . . . .	727
Usage Notes. . . . .	727
PKCS #11 Generate secret key (CSFPGSK and CSFPGSK6) . . . . .	728
Format . . . . .	728
Parameters . . . . .	728
Authorization . . . . .	730
Usage Notes. . . . .	730
PKCS #11 Generate HMAC (CSFPHMG and CSFPHMG6) . . . . .	730
Format . . . . .	731
Parameters . . . . .	731
Authorization . . . . .	733
Usage Notes. . . . .	733
PKCS #11 Verify HMAC (CSFPHMV and CSFPHMV6). . . . .	734
Format . . . . .	734
Parameters . . . . .	734
Authorization . . . . .	737
Usage Notes. . . . .	737
PKCS #11 One-way hash, sign, or verify (CSFPOWH and CSFPOWH6) . . . . .	737
Format . . . . .	738
Parameters . . . . .	738
Authorization . . . . .	741
Usage Notes. . . . .	742
PKCS #11 Private key sign (CSFPPKS and CSFPPKS6) . . . . .	742
Format . . . . .	742
Parameters . . . . .	743
Authorization . . . . .	744
Usage Notes. . . . .	744
PKCS #11 Public key verify (CSFPPKV and CSFPPKV6) . . . . .	744
Format . . . . .	745
Parameters . . . . .	745
Authorization . . . . .	747
Usage Notes. . . . .	747
PKCS #11 Pseudo-random function (CSFPPRF and CSFPPRF6) . . . . .	747
Format . . . . .	747
Parameters . . . . .	747
Authorization . . . . .	749
Usage Notes. . . . .	749

PKCS #11 Set attribute value (CSFPSAV and CSFPSAV6) . . . . .	749
Format . . . . .	750
Parameters . . . . .	750
Authorization . . . . .	751
Usage Notes. . . . .	751
PKCS #11 Secret key decrypt (CSFPSKD and CSFPSKD6) . . . . .	752
Format . . . . .	752
Parameters . . . . .	752
Authorization . . . . .	756
Usage Notes. . . . .	756
PKCS #11 Secret key encrypt (CSFPSKE and CSFPSKE6) . . . . .	756
Format . . . . .	757
Parameters . . . . .	757
Authorization . . . . .	761
Usage Notes. . . . .	761
PKCS #11 Token record create (CSFPTRC and CSFPTRC6) . . . . .	762
Format . . . . .	763
Parameters . . . . .	763
Authorization . . . . .	765
Usage Notes. . . . .	765
PKCS #11 Token record delete (CSFPTRD and CSFPTRD6) . . . . .	766
Format . . . . .	766
Parameters . . . . .	766
Authorization . . . . .	767
Usage Notes. . . . .	767
PKCS #11 Token record list (CSFPTRL and CSFPTRL6) . . . . .	768
Format . . . . .	768
Parameters . . . . .	768
Authorization . . . . .	770
Usage Notes. . . . .	771
PKCS #11 Unwrap key (CSFPUWK and CSFPUWK6) . . . . .	772
Format . . . . .	772
Parameters . . . . .	772
Authorization . . . . .	774
Usage Notes. . . . .	775
PKCS #11 Wrap key (CSFPWPK and CSFPWPK6) . . . . .	775
Format . . . . .	776
Parameters . . . . .	776
Authorization . . . . .	778
Usage Notes. . . . .	778

## Part 4. Appendixes . . . . . 779

### Appendix A. ICSF and TSS Return and Reason Codes . . . . . 781

Return Codes and Reason Codes . . . . .	781
Return Codes . . . . .	781
Reason Codes for Return Code 0 (0). . . . .	782
Reason Codes for Return Code 4 (4). . . . .	783
Reason Codes for Return Code 8 (8). . . . .	786
Reason Codes for Return Code C (12). . . . .	821
Reason Codes for Return Code 10 (16). . . . .	833

### Appendix B. Key Token Formats . . . . . 835

AES Key Token Formats. . . . .	836
AES Internal Key Token . . . . .	836
Token Validation Value . . . . .	836
DES Key Token Formats. . . . .	837
DES Internal Key Token . . . . .	837
DES External Key Token. . . . .	839
External RKX DES Key Token . . . . .	839
DES Null Key Token . . . . .	841
Variable-length Symmetric Key Token Formats . . . . .	841
Variable-length Symmetric Key Token . . . . .	841
Variable-length Symmetric Null Key Token . . . . .	850
PKA Key Token Formats . . . . .	851
PKA Null Key Token . . . . .	851
RSA Key Token Formats. . . . .	851
DSS Key Token Formats. . . . .	869
ECC Key Token Format . . . . .	873
Trusted Block Key Token . . . . .	877

### Appendix C. Control Vectors and Changing Control Vectors with the CVT Callable Service . . . . . 893

Control Vector Table . . . . .	893
Specifying a Control-Vector-Base Value . . . . .	899
Changing Control Vectors with the Control Vector Translate Callable Service . . . . .	904
Providing the Control Information for Testing the Control Vectors . . . . .	904
Mask Array Preparation. . . . .	904
Selecting the Key-Half Processing Mode . . . . .	906
When the Target Key Token CV Is Null . . . . .	908
Control Vector Translate Example. . . . .	908

### Appendix D. Coding Examples . . . . . 909

C . . . . .	909
COBOL . . . . .	912
Assembler H . . . . .	914
PL/1 . . . . .	916

### Appendix E. Using ICSF with BSAFE 921

Some BSAFE Basics . . . . .	921
Computing Message Digests and Hashes . . . . .	921
Generating Random Numbers. . . . .	922
Encrypting and Decrypting with DES . . . . .	922
Generating and Verifying RSA Digital Signatures . . . . .	922
Encrypting and Decrypting with RSA . . . . .	923
Using the New Function Calls in Your BSAFE Application . . . . .	923
Using the BSAFE KI_TOKEN . . . . .	925
ICSF Triple DES via BSAFE. . . . .	925
Retrieving ICSF Error Information . . . . .	926

### Appendix F. Cryptographic Algorithms and Processes . . . . . 929

PIN Formats and Algorithms . . . . .	929
PIN Notation . . . . .	929
PIN Block Formats . . . . .	929
PIN Extraction Rules . . . . .	931

IBM PIN Algorithms . . . . .	932
VISA PIN Algorithms . . . . .	938
Cipher Processing Rules . . . . .	940
CBC and ANSI X3.106 . . . . .	940
ANSI X9.23 and IBM 4700 . . . . .	941
CUSP . . . . .	942
The Information Protection System (IPS) . . . . .	942
PKCS Padding Method . . . . .	943
Wrapping Methods for Symmetric Key Tokens . . . . .	945
ECB Wrapping of DES Keys (Original Method) . . . . .	945
CBC Wrapping of AES Keys . . . . .	945
Enhanced CBC Wrapping of DES Keys (Enhanced Method) . . . . .	945
Wrapping key derivation for enhanced wrapping of DES keys . . . . .	946
Variable length token (AESKW method) . . . . .	947
PKA92 Key Format and Encryption Process . . . . .	947
ANSI X9.17 Partial Notarization Method . . . . .	949
Partial Notarization . . . . .	949
Transform CDMF Key Algorithm . . . . .	951
Formatting Hashes and Keys in Public-Key Cryptography . . . . .	952
ANSI X9.31 Hash Format . . . . .	952
PKCS #1 Formats . . . . .	953
Visa and EMV-related smart card formats and processes . . . . .	954
Deriving the smart-card-specific authentication code . . . . .	954
Constructing the PIN-block for transporting an EMV smart-card PIN . . . . .	954

Deriving the CCA TDES-XOR session key . . . . .	955
Deriving the EMV TDESEMVn tree-based session key . . . . .	955
PIN-block self-encryption . . . . .	956
Key Test Verification Pattern Algorithms . . . . .	956
DES Algorithm (single- and double-length keys) . . . . .	956
SHAVP1 Algorithm . . . . .	956

<b>Appendix G. EBCDIC and ASCII Default Conversion Tables . . . . .</b>	<b>959</b>
---	------------

<b>  Appendix H. Access Control Points   and Callable Services . . . . .</b>	<b>961</b>
--	------------

<b>Appendix I. Accessibility . . . . .</b>	<b>977</b>
Using assistive technologies . . . . .	977
Keyboard navigation of the user interface . . . . .	977
z/OS information . . . . .	977

<b>Notices . . . . .</b>	<b>979</b>
Programming Interface Information . . . . .	980
Trademarks . . . . .	980

<b>Glossary . . . . .</b>	<b>983</b>
---------------------------	------------

<b>Index . . . . .</b>	<b>993</b>
------------------------	------------





---

## Figures

1. Overview of trusted block contents . . . . .	37		11. Control Vector Base Bit Map (Key Generating	
2. Simplified RKX key-token structure . . . . .	41		Keys) . . . . .	899
3. Trusted block creation . . . . .	42		12. Control Vector Translate Callable Service	
4. Exporting keys using a trusted block . . . . .	43		Mask_Array Processing . . . . .	906
5. Generating keys using a trusted block. . . . .	46		13. Control Vector Translate Callable Service	907
6. Typical flow of callable services for remote key			14. 3624 PIN Generation Algorithm . . . . .	933
export . . . . .	47		15. GBP PIN Generation Algorithm . . . . .	934
7. PKA Key Management. . . . .	92		16. PIN-Offset Generation Algorithm . . . . .	935
8. Control Vector Base Bit Map (Common Bits			17. PIN Verification Algorithm . . . . .	937
and Key-Encrypting Keys) . . . . .	896		18. GBP PIN Verification Algorithm . . . . .	938
9. Control Vector Base Bit Map (Data Operation			19. PVV Generation Algorithm . . . . .	939
Keys) . . . . .	897		20. The CDMF Key Transformation Algorithm	952
10. Control Vector Base Bit Map (PIN Processing				
Keys and Cryptographic Variable-Encrypting				
Keys) . . . . .	898			



## Tables

1. ICSF Callable Services Naming Conventions	3	37. Key Generate Valid Key Types and Key Forms for a Key Pair	151
2. Standard Return Code Values From ICSF Callable Services	7	38. Required access control points for Key Generate	153
3. Descriptions of Key Types.	23	39. Key generate required hardware	155
4. AES EXPORTER strength required for exporting an HMAC key under an AES EXPORTER	26	40. Keywords for Key Generate2 Control Information	158
5. Minimum RSA modulus length to adequately protect an AES key	26	41. Keywords and associated algorithms for key_type_1 parameter.	159
6. Summary of Data Encryption Standard Bits	53	42. Keywords and associated algorithms for key_type_2 parameter.	159
7. Combinations of the Callable Services.	66	43. Key Generate2 valid key type and key form for one key	163
8. Summary of ICSF Callable Services	71	44. Key Generate2 Valid key type and key forms for two keys	163
9. AES EXPORTER strength required for exporting an HMAC key under an AES EXPORTER	85	45. Key Generate2 Valid key forms for CIPHER keys	163
10. Minimum RSA modulus length to adequately protect an AES key	85	46. AES KEK strength required for generating an HMAC key under an AES KEK	164
11. Summary of PKA Key Token Sections.	92	47. Required access control points for Key Generate2.	164
12. Summary of PKA Callable Services	95	48. Key Generate2 required hardware.	165
13. Summary of PKCS #11 callable services	97	49. Required access control points for Key Import	169
14. Clear key import required hardware	105	50. Key import required hardware.	170
15. Control vector generate required hardware	110	51. Keywords for Key Part Import Control Information	173
16. Keywords for Control Vector Translate	112	52. Required access control points for Key Part Import.	174
17. Control vector translate required hardware	114	53. Key part import required hardware	175
18. Cryptographic variable encipher required hardware	116	54. Keywords for Key Part Import2 Control Information	177
19. Required access control points for Data key export	118	55. Required access control points for Key Part Import2	179
20. Data key export required hardware	119	56. Key Part Import2 required hardware	179
21. Required access control points for Data key import	121	57. Keywords for Key Test Control Information	181
22. Data key import required hardware	122	58. Key test required hardware	183
23. Rule Array Keywords for Diversified Key Generate	124	59. Keywords for Key Test2 Control Information	185
24. Required access control points for Diversified Key Generate	127	60. Key Test2 required hardware	188
25. Diversified key generate required hardware	128	61. Keywords for Key Test Extended Control Information	190
26. Keywords for ECC Diffie-Hellman	131	62. Key test extended required hardware	192
27. Valid key bit lengths and minimum curve size required for the supported output key types.	135	63. Key type keywords for key token build	194
28. ECC Diffie-Hellman required hardware	136	64. Keywords for Key Token Build Control Information	195
29. Required access control points for Key Export	140	65. Key types and field lengths for AES keys	198
30. Key export required hardware	141	66. Control Vector Generate and Key Token Build Control Vector Keyword Combinations	200
31. Key Form values for the Key Generate callable service	143	67. Key token build required hardware	202
32. Key Length values for the Key Generate callable service	145	68. Keywords for Key Token Build2 Control Information	204
33. Key lengths for DES keys - CCF systems	146	69. Key Token Build2 required hardware	208
34. Key lengths for DES keys - PCIXCC and CCA Crypto Express coprocessor systems	146	70. Key translate required hardware	211
35. Key lengths for AES keys - CCA Crypto Express coprocessor systems	147	71. Key Translate2 Access Control Points	216
36. Key Generate Valid Key Types and Key Forms for a Single Key	151	72. Key Translate2 required hardware	217
		73. Keywords for Multiple Clear Key Import Rule Array Control Information	219

74. Required access control points for Multiple Clear Key Import . . . . .	220	112. Symmetric Key Import2 Access Control Points . . . . .	294
75. Multiple clear key import required hardware	221	113. Symmetric key import2 required hardware	294
76. Keywords for Multiple Secure Key Import Rule Array Control Information . . . . .	223	114. Transform CDMF key required hardware	297
77. Required access control points for Multiple Secure Key Import . . . . .	226	115. Rule_array keywords for Trusted Block Create (CSNDTBC) . . . . .	299
78. Multiple secure key import required hardware . . . . .	227	116. Required access control points for Trusted Block Create . . . . .	300
79. Keywords for PKA Decrypt . . . . .	230	117. Trusted Block Create required hardware	301
80. PKA decrypt required hardware . . . . .	233	118. Keywords for TR-31 Export Rule Array Control Information . . . . .	303
81. Keywords for PKA Encrypt . . . . .	236	119. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs) . . . . .	308
82. PKA encrypt required hardware . . . . .	238	120. TR-31 export required hardware . . . . .	315
83. Prohibit export required hardware . . . . .	240	121. Keywords for TR-31 Import Rule Array Control Information . . . . .	317
84. Prohibit export extended required hardware	243	122. Export attributes of an imported CCA token	322
85. Keywords for the Form Parameter . . . . .	244	123. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) . . . . .	323
86. Keywords for Random Number Generate Control Information . . . . .	245	124. TR-31 export required hardware . . . . .	328
87. Random number generate required hardware	246	125. TR-31 Optional Data Build required hardware	331
88. Structure of values used by RKX . . . . .	250	126. Keywords for TR-31 Optional Data Read Rule Array Control Information . . . . .	333
89. Values defined for hash algorithm identifier at offset 24 in the structure for remote key export . . . . .	251	127. TR-31 Optional Data Read required hardware	335
90. Transport_key_idenfiter used by RKX	251	128. TR-31 Parse required hardware . . . . .	339
91. Examination of key token for source_key_identifier . . . . .	252	129. Keywords for Unique Key Derive. . . . .	341
92. Remote key export required hardware	255	130. Valid Control Vectors for Derived Keys	345
93. Keywords for Restrict Key Attribute Control Information . . . . .	257	131. Derivation Variants . . . . .	346
94. Restrict Key Attribute required hardware	259	132. Unique Key Derive required hardware	346
95. Required access control points for Secure Key Import . . . . .	263	133. Keywords for User Derived Key Control Information . . . . .	349
96. Secure key import required hardware	264	134. User derived key required hardware	350
97. Keywords for Secure Key Import2 Control Information . . . . .	266	135. Ciphertext translate required hardware	357
98. Required access control points for Secure Key Import2 . . . . .	268	136. Keywords for Ciphertext Translate2 . . . . .	360
99. Secure Key Import2 required hardware	268	137. Restrictions for ciphertext_in_length and ciphertext_out_length . . . . .	365
100. Keywords for Symmetric Key Export Control Information . . . . .	270	138. Ciphertext translate2 key usage . . . . .	369
101. Minimum RSA modulus strength required to contain a PKOAEP2 block when exporting an AES key . . . . .	272	139. Ciphertext translate2 Access Control Points	369
102. Required access control points for Symmetric Key Export . . . . .	272	140. Ciphertext translate2 required hardware	370
103. Symmetric key export required hardware	273	141. Keywords for the Decipher Rule Array Control Information . . . . .	374
104. Keywords for Symmetric Key Generate Control Information . . . . .	276	142. Decipher required hardware . . . . .	377
105. Required access control points for Symmetric Key Generate . . . . .	280	143. Decode required hardware . . . . .	379
106. Symmetric key generate required hardware	281	144. Keywords for the Encipher Rule Array Control Information . . . . .	384
107. Keywords for Symmetric Key Import Control Information . . . . .	284	145. Encipher required hardware . . . . .	386
108. Required access control points for Symmetric Key Import . . . . .	286	146. Encode required hardware . . . . .	389
109. Symmetric key import required hardware	288	147. Symmetric Algorithm Decipher Rule Array Keywords. . . . .	392
110. Keywords for Symmetric Key Import2 Control Information . . . . .	292	148. Symmetric Algorithm Decipher required hardware . . . . .	395
111. PKCS#1 OAEP encoded message layout (PKOAEP2) . . . . .	294	149. Symmetric Algorithm Encipher Rule Array Keywords. . . . .	398
		150. Symmetric Algorithm Encipher required hardware . . . . .	402
		151. Symmetric Key Decipher Rule Array Keywords. . . . .	406
		152. Required access control points for Symmetric Key Decipher . . . . .	411
		153. Symmetric Key Decipher required hardware	411

154. Symmetric Key Encipher Rule Array Keywords. . . . .	415	194. PIN Block Variant Constants (PBVCs)	490
155. Required access control points for Symmetric Key Encipher . . . . .	420	195. Required access control points for Clear PIN Generate Alternate. . . . .	490
156. Symmetric Key Encipher required hardware	420	196. Clear PIN generate alternate required hardware . . . . .	491
157. Keywords for HMAC Generate Control Information . . . . .	427	197. Keywords for CVV Key Combine Rule Array Control Information . . . . .	493
158. HMAC Generate Access Control Points	429	198. Key type combinations for the CVV key combine callable service . . . . .	495
159. HMAC generate required hardware . . . .	429	199. Wrapping combinations for the CVV Combine Callable Service . . . . .	495
160. Keywords for HMAC Verify Control Information . . . . .	431	200. TR-31 export required hardware . . . . .	496
161. HMAC Verify Access Control Points . . . .	433	201. Process Rules for the Encrypted PIN Generate Callable Service. . . . .	499
162. HMAC generate required hardware . . . .	433	202. Array Elements for the Encrypted PIN Generate Callable Service . . . . .	499
163. Keywords for MAC generate Control Information . . . . .	436	203. Array Elements Required by the Process Rule	500
164. MAC generate required hardware. . . . .	439	204. Required access control points for Encrypted PIN Generate . . . . .	501
165. Keywords for MAC verify Control Information . . . . .	442	205. Encrypted PIN generate required hardware	501
166. MAC verify required hardware . . . . .	444	206. Keywords for Encrypted PIN Translate	504
167. Keywords for MDC Generate Control Information . . . . .	448	207. Additional Names for PIN Formats . . . .	507
168. MDC generate required hardware. . . . .	449	208. PIN Block Variant Constants (PBVCs)	507
169. Keywords for One-Way Hash Generate Rule Array Control Information . . . . .	451	209. Required access control points for Encrypted PIN Translate . . . . .	507
170. One-way hash generate required hardware	454	210. Encrypted PIN translate required hardware	508
171. Keywords for symmetric MAC generate control information . . . . .	457	211. Keywords for Encrypted PIN Verify . . . .	511
172. Symmetric MAC generate required hardware	458	212. Array Elements for the Encrypted PIN Verify Callable Service. . . . .	513
173. Keywords for symmetric MAC verify control information . . . . .	461	213. Array Elements Required by the Process Rule	513
174. Symmetric MAC verify required hardware	463	214. PIN Block Variant Constants (PBVCs)	514
175. ANSI X9.8 PIN - Allow only ANSI PIN blocks	470	215. Required access control points for Encrypted PIN Verify . . . . .	514
176. Format of a PIN Profile . . . . .	471	216. Encrypted PIN verify required hardware	515
177. Format Values of PIN Blocks . . . . .	471	217. Rule Array Keywords for PIN Change/Unblock . . . . .	518
178. PIN Block Format and PIN Extraction Method Keywords . . . . .	472	218. Required access control points for PIN Change/Unblock . . . . .	522
179. Callable Services Affected by Enhanced PIN Security Mode . . . . .	474	219. PIN Change/Unblock hardware . . . . .	522
180. Format of a Pad Digit. . . . .	475	220. Rule Array Keywords for Secure Messaging for Keys . . . . .	524
181. Pad Digits for PIN Block Formats. . . . .	475	221. Secure messaging for keys required hardware	526
182. Format of the Current Key Serial Number Field . . . . .	476	222. Rule Array Keywords for Secure Messaging for PINs . . . . .	528
183. Process Rules for the Clear PIN Encryption Callable Service. . . . .	478	223. Secure messaging for PINs required hardware	530
184. Clear PIN encrypt required hardware	480	224. Keywords for SET Block Compose Control Information . . . . .	532
185. Process Rules for the Clear PIN Generate Callable Service. . . . .	482	225. SET block compose required hardware	536
186. Array Elements for the Clear PIN Generate Callable Service. . . . .	483	226. Keywords for SET Block Compose Control Information . . . . .	538
187. Array Elements Required by the Process Rule	483	227. Required access control points for PIN-block encrypting key . . . . .	541
188. Required access control points for Clear PIN Generate . . . . .	484	228. SET block decompose required hardware	542
189. Clear PIN generate required hardware	484	229. Rule Array Keywords for Transaction Validation. . . . .	544
190. Rule Array Elements for the Clear PIN Generate Alternate Service . . . . .	488	230. Output description for validation values	545
191. Rule Array Keywords (First Element) for the Clear PIN Generate Alternate Service . . . .	488	231. Required access control points for Transaction Validation. . . . .	545
192. Data Array Elements for the Clear PIN Generate Alternate Service (IBM-PINO) . . . .	489	232. Transaction validation required hardware	546
193. Data Array Elements for the Clear PIN Generate Alternate Service (VISA-PVV) . . . .	489	233. CVV Generate Rule Array Keywords	548

234. VISA CVV service generate required hardware . . . . .	550	282. Output for option NUM-DECT . . . . .	663
235. CVV Verify Rule Array Keywords . . . . .	552	283. Output for option STATAES. . . . .	663
236. VISA CVV service verify required hardware	555	284. Output for option STATCCA . . . . .	664
237. Keywords for Digital Signature Generate Control Information . . . . .	559	285. Output for option STATCCAE . . . . .	664
238. Digital signature generate required hardware	563	286. Output for option STATCARD . . . . .	665
239. Keywords for Digital Signature Verify Control Information . . . . .	567	287. Output for option STATDECT . . . . .	666
240. Digital signature verify required hardware	569	288. Output for option STATDIAG . . . . .	667
241. Keywords for PKA Key Generate Rule Array	573	289. Output for option STATEID. . . . .	668
242. Required access control points for PKA Key Generate rule array keys . . . . .	576	290. Output for option STATEXPT . . . . .	668
243. PKA key generate required hardware	577	291. Output for option STATAPKA . . . . .	670
244. Keywords for PKA Key Import . . . . .	579	292. Output for option WRAPMTHD . . . . .	670
245. PKA key import required hardware . . . . .	582	293. Output for option STATP11 . . . . .	671
246. Keywords for PKA Key Token Build Control Information . . . . .	585	294. ICSF Query Service required hardware	672
247. Key Value Structure Length Maximum Values for Key Types . . . . .	586	295. X9.9 data editing required hardware . . . . .	674
248. Key Value Structure Elements for PKA Key Token Build . . . . .	587	296. Keywords for PCI Interface Callable Service	678
249. PKA key token build required hardware	596	297. PCI Interface required hardware . . . . .	682
250. Rule Array Keywords for PKA Key Token Change . . . . .	597	298. PKSC Interface required hardware . . . . .	684
251. PKA key token change required hardware	599	299. ANSI X9.17 EDC generate required hardware	687
252. Keywords for PKA Key Generate Rule Array	601	300. Keywords for ANSI X9.17 Key Export Rule Array . . . . .	689
253. Required access control points for PKA Key Translate . . . . .	602	301. ANSI X9.17 key export required hardware	692
254. Required access control points for source/target transport key combinations . . . . .	603	302. Keywords for ANSI X9.17 Key Import Rule Array . . . . .	694
255. PKA key translate required hardware	603	303. ANSI X9.17 key import required hardware	697
256. PKA public key extract build required hardware . . . . .	606	304. Keywords for ANSI X9.17 Key Translate Rule Array . . . . .	699
257. Retained key delete required hardware	608	305. ANSI X9.17 key translate required hardware	702
258. Retained key list required hardware . . . . .	611	306. ANSI X9.17 transport key partial notarize required hardware . . . . .	704
259. CKDS record create required hardware	615	307. Keywords for derive multiple keys . . . . .	711
260. CKDS Key Record Create2 required hardware	617	308. parms_list parameter format for SSL-KM and TLS-KM mechanisms . . . . .	713
261. CKDS record delete required hardware	619	309. parms_list parameter format for IKE1PHA1 mechanism . . . . .	714
262. CKDS record read required hardware	621	310. parms_list parameter format for IKE2PHA1 mechanism . . . . .	714
263. CKDS key record read2 required hardware	623	311. parms_list parameter format for IKE1PHA2 and IKE2PHA2 mechanisms . . . . .	714
264. CKDS record write required hardware	625	312. Keywords for derive key. . . . .	718
265. CKDS key record write2 required hardware	627	313. parms_list parameter format for PKCS-DH mechanism . . . . .	720
266. Coordinated KDS administration required hardware . . . . .	631	314. parms_list parameter format for SSL-MS, SSL-MSDH, TLS-MS, and TLS-MSDH mechanisms . . . . .	720
267. PKDS key record create required hardware	634	315. parms_list parameter format for EC-DH mechanism . . . . .	720
268. Keywords for PKDS Key Record Delete	635	316. parms_list parameter format for IKESSEED, IKESHARE, and IKEREKEY mechanisms . . . . .	721
269. PKDS key record delete required hardware	636	317. Get attribute value processing for objects possessing sensitive attributes . . . . .	725
270. PKDS key record read required hardware	638	318. Keywords for generate secret key. . . . .	729
271. Keywords for PKDS Key Record Write	640	319. parms_list parameter format for SSL and TLS mechanism . . . . .	730
272. PKDS key record write required hardware	641	320. parms_list parameter format for PBEKEY mechanism . . . . .	730
273. Character/Nibble conversion required hardware . . . . .	645	321. Keywords for generate HMAC. . . . .	732
274. Code conversion required hardware . . . . .	647	322. chain_data parameter format . . . . .	733
275. Keywords for ICSF Query Algorithm	649	323. Keywords for verify HMAC . . . . .	735
276. Output for ICSF Query Algorithm . . . . .	650	324. chain_data parameter format . . . . .	736
277. ICSF Query Algorithm required hardware	651	325. Keywords for one-way hash generate	739
278. Keywords for ICSF Query Service. . . . .	653	326. chain_data parameter format . . . . .	740
279. Output for option ICSFSTAT . . . . .	655		
280. Output for option ICSFSP11. . . . .	657		
281. Output for option ICSFST2 . . . . .	657		



327.	Keywords for private key sign . . . . .	743	367.	RSA Private Internal Key Token Basic Record Format. . . . .	860
328.	Keywords for public key verify . . . . .	746	368.	RSA Private Internal Key Token, 1024-bit ME Form for Cryptographic Coprocessor Feature .	861
329.	Keywords for PKCS #11 Pseudo-random function . . . . .	748	369.	RSA Private Internal Key Token, 1024-bit ME Form for PCICC, PCIXCC, or CCA Crypto Express coprocessor . . . . .	862
330.	parms_list parameter format for TLS-PRF mechanism . . . . .	749	370.	RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form . . . . .	863
331.	Authorization requirements for the set attribute value callable service . . . . .	751	371.	RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form . . . . .	865
332.	Keywords for secret key decrypt . . . . .	753	372.	RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format .	867
333.	initialization_vector parameter format for GCM mechanism . . . . .	754	373.	DSS Public Key Token . . . . .	869
334.	chain_data parameter format . . . . .	755	374.	DSS Private External Key Token . . . . .	870
335.	Keywords for secret key encrypt . . . . .	758	375.	DSS Private Internal Key Token . . . . .	871
336.	initialization_vector parameter format for GCM mechanism . . . . .	759	376.	ECC Key Token Format . . . . .	873
337.	initialization_vector parameter format for GCMIVGEN mechanism . . . . .	760	377.	Associated Data Format for ECC Private Key Token . . . . .	876
338.	chain_data parameter format . . . . .	760	378.	AESKW Wrapped Payload Format for ECC Private Key Token . . . . .	877
339.	Authorization requirements for the token record create callable service . . . . .	765	379.	Trusted block header . . . . .	879
340.	Authorization requirements for the token record delete callable service . . . . .	767	380.	Trusted block trusted RSA public-key section (X'11') . . . . .	880
341.	Keywords for unwrap key . . . . .	773	381.	Trusted block rule section (X'12') . . . . .	881
342.	Keywords for wrap key . . . . .	777	382.	Summary of trusted block rule subsection .	882
343.	Return Codes . . . . .	781	383.	Transport key variant subsection (X'0001' of trusted block rule section (X'12') . . . . .	883
344.	Reason Codes for Return Code 0 (0) . . . .	782	384.	Transport key rule reference subsection (X'0002') of trusted block rule section (X'12') .	884
345.	Reason Codes for Return Code 4 (4) . . . .	784	385.	Common export key parameters subsection (X'0003') of trusted block rule section (X'12') .	884
346.	Reason Codes for Return Code 8 (8) . . . .	786	386.	Source key rule reference subsection (X'0004' of trusted block rule section (X'12') . . . . .	886
347.	Reason Codes for Return Code C (12) . . . .	822	387.	Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12') .	886
348.	Reason Codes for Return Code 10 (16) . . . .	833	388.	Trusted block key label (name) section X'13' .	888
349.	Internal Key Token Format . . . . .	836	389.	Trusted block information section X'14' .	888
350.	Internal Key Token Format . . . . .	837	390.	Summary of trusted block information subsections . . . . .	889
351.	Format of External Key Tokens . . . . .	839	391.	Protection information subsection (X'0001') of trusted block information section (X'14') . .	889
352.	External RKX DES key-token format, version X'10' . . . . .	840	392.	Activation and expiration dates subsection (X'0002') of trusted block information section (X'14') . . . . .	890
353.	Format of Null Key Tokens . . . . .	841	393.	Trusted block application-defined data section X'15' . . . . .	890
354.	Variable-length Symmetric Key Token . . . .	841	394.	Default Control Vector Values . . . . .	893
355.	HMAC Algorithm Key-usage fields . . . . .	846	395.	PKA96 Clear DES Key Record . . . . .	948
356.	AES Algorithm KEK Key-usage fields . . . .	847	396.	EBCDIC to ASCII Default Conversion Table .	959
357.	AES Algorithm Cipher Key Associated Data .	850	397.	ASCII to EBCDIC Default Conversion Table .	960
358.	Variable-length Symmetric Null Token . . . .	850	398.	Access control points affecting multiple services or requiring special consideration .	961
359.	Format of PKA Null Key Tokens . . . . .	851	399.	Access control points – Callable Services .	965
360.	RSA Public Key Token . . . . .	851			
361.	RSA Private External Key Token Basic Record Format. . . . .	852			
362.	RSA Private Key Token, 1024-bit Modulus-Exponent External Format . . . . .	853			
363.	RSA Private Key Token, 4096-bit Modulus-Exponent External Format . . . . .	854			
364.	RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external form . . . . .	855			
365.	RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form. . . . .	856			
366.	RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Format . . . . .	858			



---

## About this information

This information supports z/OS (5694-A01). It describes how to use the callable services provided by the Integrated Cryptographic Service Facility (ICSF). The z/OS Cryptographic Services includes these components:

- z/OS Integrated Cryptographic Service Facility (ICSF)
- z/OS Open Cryptographic Services Facility (OCSF)
- z/OS System Secure Socket Level Programming (SSL)
- z/OS Public Key Infrastructure Services (PKI)

ICSF is a software element of z/OS that works with the hardware cryptographic feature and the Security Server RACF to provide secure, high-speed cryptographic services. ICSF provides the application programming interfaces by which applications request the cryptographic services.

**Note:** References to the IBM @server zSeries 800 (z800) do not appear in this information. Be aware that the documented notes and restrictions for the IBM @server zSeries 900 (z900) also apply to the z800.

---

## Who should use this information

This information is intended for application programmers who:

- Are responsible for writing application programs that use the security application programming interface (API) to access cryptographic functions.
- Want to use ICSF callable services in high-level languages such as C, COBOL, FORTRAN, and PL/I, as well as in assembler.

---

## How to use this information

ICSF includes Advanced Encryption Standard (AES), Data Encryption Standard (DES) and public key cryptography. These are very different cryptographic systems.

These topics focus on IBM CCA programming and include:

- Chapter 1, “Introducing Programming for the IBM CCA” describes the programming considerations for using the ICSF callable services. It also explains the syntax and parameter definitions used in callable services.
- Chapter 2, “Introducing Symmetric Key Cryptography and Using Symmetric Key Callable Services” gives an overview of AES and DES cryptography and provides general guidance information on how the callable services use different key types and key forms. It also discusses how to write your own callable services called installation-defined callable services and provides suggestions on what to do if there is a problem.
- Chapter 3, “Introducing PKA Cryptography and Using PKA Callable Services” introduces Public Key Algorithm (PKA) support and describes programming considerations for using the ICSF PKA callable services, such as the PKA key token structure and key management.
- Chapter 4, “Introducing PKCS #11 and using PKCS #11 callable services” gives an overview of PKCS #11 support and management services.

These topics focus on CCA callable services and include:

- Chapter 5, “Managing Symmetric Cryptographic Keys” describes the callable services for generating and maintaining cryptographic keys and the random number generate callable service. It also presents utilities to build AES and DES tokens and generate and translate control vectors and describes the PKA callable services that support AES and DES key distribution.
- Chapter 6, “Protecting Data” describes the callable services for deciphering ciphertext from one key and enciphering it under another key. It also describes enciphering and deciphering data with encrypted keys and encoding and decoding data with clear keys.
- Chapter 7, “Verifying Data Integrity and Authenticating Messages” describes the callable services for generating and verifying message authentication codes (MACs), generating modification detection codes (MDCs) and generating hashes (SHA-1, SHA-2, MD5, RIPEMD-160).
- Chapter 8, “Financial Services” describes the callable services for generating, verifying, and translating personal identification numbers (PINs). It also describes the callable services that support the Secure Electronic Transaction (SET) protocol and those that generate and verify VISA card verification values and American Express card security codes.
- Chapter 9, “Using Digital Signatures” describes the PKA callable services that support using digital signatures to authenticate messages.
- Chapter 10, “Managing PKA Cryptographic Keys” describes the PKA callable services that generate and manage PKA keys.
- Chapter 11, “Key Data Set Management,” on page 613 describes the callable services that manage key tokens in the Cryptographic Key Data Set (CKDS) and the PKA Key Data Set (PKDS).
- Chapter 12, “Utilities” describes callable services that convert data between EBCDIC and ASCII format, convert between binary strings and character strings, and query ICSF services and algorithms.
- Chapter 13, “Trusted Key Entry Workstation Interfaces” describes the PCI interface (PCI) and the Public Key Secure Cable (PKSC) interface that supports Trusted Key Entry (TKE), an optional feature available with ICSF.
- Chapter 14, “Managing Keys According to the ANSI X9.17 Standard” describes the callable services that support the ANSI X9.17 key management standard <sup>1</sup>, which defines a process for protecting and exchanging DES keys.
- Chapter 15, “Using PKCS #11 Tokens and Objects” describes the callable services for managing the PKCS #11 tokens and objects in the TKDS.

The appendixes include this information:

- Appendix A, “ICSF and TSS Return and Reason Codes” explains the return and reason codes returned by the callable services.
- Appendix B, “Key Token Formats” describes the formats for AES key tokens, DES internal, external, and null key tokens and for PKA public, private external, and private internal key tokens containing either Rivest-Shamir-Adleman (RSA) or Digital Signature Standard (DSS) information. This appendix also describes the PKA null key token.
- Appendix C, “Control Vectors and Changing Control Vectors with the CVT Callable Service,” on page 893 contains a table of the default control vector values that are associated with each key type and describes the control

---

1. ANSI X9.17-1985: Financial Institution Key Management (Wholesale)

information for testing control vectors, mask array preparation, selecting the key-half processing mode, and an example of Control Vector Translate.

- Appendix D, “Coding Examples” provides examples for COBOL, assembler, and PL/1.
- Appendix E, “Using ICSF with BSAFE” explains how to access ICSF services from applications written using RSA’s BSAFE cryptographic toolkit.
- Appendix F, “Cryptographic Algorithms and Processes,” on page 929 describes the PIN formats and algorithms, cipher processing and segmenting rules, multiple encipherment and decipherment and their equations, the PKA92 encryption process, partial notarization of an ANSI key-encrypting key (AKEK), and the algorithm for transforming a Commercial Data Masking Facility (CDMF) key.
- Appendix G, “EBCDIC and ASCII Default Conversion Tables” presents EBCDIC to ASCII and ASCII to EBCDIC conversion tables.
- Appendix H, “Access Control Points and Callable Services” lists which access control points correspond to which callable services.
- Appendix I, “Accessibility,” on page 977 contains information on accessibility features in z/OS.
- Notices contains notices, programming interface information, and trademarks.

---

## Where to find more information

The publications in the z/OS ICSF library include:

- *z/OS Cryptographic Services ICSF Overview*
- *z/OS Cryptographic Services ICSF Administrator’s Guide*
- *z/OS Cryptographic Services ICSF System Programmer’s Guide*
- *z/OS Cryptographic Services ICSF Application Programmer’s Guide*
- *z/OS Cryptographic Services ICSF Messages*
- *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*

## Related Publications

- *z/OS Cryptographic Services ICSF TKE Workstation User’s Guide*, SA23-2211
- *z/OS MVS Programming: Callable Services for High-Level Languages*, SA22-7613
- *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU*, SA22-7611
- *BSAFE User’s Guide*
- *z/OS Security Server RACF Command Language Reference*
- *z/OS Security Server RACF Security Administrator’s Guide*
- *IBM Common Cryptographic Architecture (CCA) Basic Services API, Release 2.53*

This publication can be obtained in PDF format from the Library page at <http://www.ibm.com/security/cryptocards>.





---

## How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or give us any other feedback that you might have.

Use one of the following methods to send us your comments:

1. Send an email to [mhvrcfs@us.ibm.com](mailto:mhvrcfs@us.ibm.com)
2. Visit the Contact z/OS web page at <http://www.ibm.com/systems/z/os/zos/webqs.html>
3. Mail the comments to the following address:  
IBM Corporation  
Attention: MHVRCFS Reader Comments  
Department H6MA, Building 707  
2455 South Road  
Poughkeepsie, NY 12601-5400  
U.S.A.
4. Fax the comments to us as follows:  
From the United States and Canada: 1+845+432-9405  
From all other countries: Your international access code +1+845+432-9405

Include the following information:

- Your name and address
- Your email address
- Your telephone or fax number
- The publication title and order number:  
z/OS V1R13.0 ICSF Application Programmer's Guide  
SA22-7522-16
- The topic and page number related to your comment
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you submit.

---

## If you have a technical problem

Do not use the feedback methods listed above. Instead, do one of the following:

- Contact your IBM service representative
- Call IBM technical support
- Visit the IBM support portal at <http://www.ibm.com/systems/z/support/>



---

## Summary of changes

ICSF is an element of z/OS, but provides independent ICSF releases as web deliverables. These web deliverables are identified by their FMID. Each release of z/OS includes a particular ICSF FMID level as part of its base. Refer to *z/OS Cryptographic Services ICSF Overview* for more information on z/OS ICSF FMIDs and their relationships to z/OS releases.

---

### Changes made in Cryptographic Support for z/OS V1R12-R13 (FMID HCR77A0)

This document contains information previously presented in *z/OS ICSF Application Programmer's Guide*, SA22-7522-15.

This document is for ICSF FMID HCR77A0. This release of ICSF runs on z/OS V1R12 and z/OS V1R13 and only on zSeries hardware.

#### New information:

- Added information for the Crypto Express4 feature, which can be configured as a CCA coprocessor (CEX4C), a Enterprise PKCS #11 coprocessor (CEX4P), or an accelerator (CEX4A).
- A new callable service, Unique Key Derive (CSFBUKD and CSFEUKD) has been added to perform the key derivation process using a base derivation key and derivation data as inputs for increased support for the Derived Unique Key Per Transaction (DUKPT) key-management scheme as described in ANSI X9.24 Part 1.
- A new callable service, Cipher Text Translate2 (CSNBCTT2, CSNBCTT3, CSNECTT2, CSNECTT3), has been added to translate cipher text. This new service supports both AES and DES algorithms and different encryption modes.
- Added a set of access control points in the ICSF role to control the wrapping of keys. ICSF administrators can use these access control points to ensure that a key is not wrapped with a key weaker than itself.

#### Changed information:

- Clear Key Import (CSNBCKI and CSNECKI)
- Control Vector Generate (CSNBCVG and CSNECVG)
- Coordinated KDS Administration Callable Service (CSFCRC and CSFCRC6)
- Data Key Export (CSNBDKX and CSNEDKX)
- Data Key Import (CSNBDKM and CSNEDKM)
- Diversified Key Generate (CSNBDKG and CSNEDKG)
- ECC Diffie-Hellman (CSNDEDH and CSNFEDH)
- Key Export (CSNBKEX and CSNEKEX)
- Key Generate (CSNBKGN and CSNEKGN)
- Key Generate2 (CSNBKGN2 and CSNEKGN2)
- Key Import (CSNBKIM and CSNEKIM)
- Key Token Build (CSNBKTB and CSNEKTB)
- Key Token Build2 (CSNBKTB2 and CSNEKTB2)
- Key Translate (CSNBKTR and CSNEKTR)

- Key Translate2 (CSNBKTR2 and CSNEKTR2)
- Multiple Clear Key Import (CSNBCKM and CSNECKM)
- Multiple Secure Key Import (CSNBSKM and CSNESKM)
- PIN Change/Unblock (CSNBPCU and CSNEPCU)
- PKA Decrypt (CSNDPKD and CSNFPKD)
- Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)
- Remote Key Export (CSNDRKX and CSNFRKX)
- Restrict Key Attribute (CSNBRKA and CSNERKA)
- Secure Key Import (CSNBSKI and CSNESKI)
- Secure Key Import2 (CSNBSKI2 and CSNESKI2)
- Symmetric Key Export (CSNDSYX and CSNFSYX)
- Symmetric Key Generate (CSNDSYG and CSNFSYG)
- Symmetric Key Import (CSNDSYI and CSNFSYI)
- Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2)
- Transaction Validation (CSNBTRV and CSNETRV)
- TR-31 Export (CSNBT31X and CSNET31X)
- TR-31 Import (CSNBT31I and CSNET31I)
- Digital Signature Generate (CSNDDSG and CSNFDSG)
- Digital Signature Verify (CSNDDSV and CSNFDSV)
- PKA Key Generate (CSNDPKG and CSNFPKG)
- PKA Key Import (CSNDPKI and CSNFPKI)
- PKA Key Token Build (CSNDPKB and CSNFPKB)
- PKA Key Token Change (CSNDKTC and CSNFKTC)
- PKA Key Translate (CSNDPKT and CSNFPKT)
- Coordinated KDS Administration (CSFCRC and CSFCRC6)
- ICSF Query Facility (CSFIQF and CSFIQF6)
- PCI Interface Callable Service (CSFPCI and CSFPCI6)
- PKCS #11 Generate secret key (CSFPGSK and CSFPGSK6)
- PKCS #11 Verify HMAC (CSFPHMV and CSFPHMV6)
- PKCS #11 Private key sign (CSFPPKS and CSFPPKS6)
- PKCS #11 Public key verify (CSFPPKV and CSFPPKV6)
- PKCS #11 Pseudo-random function (CSFPPRF and CSFPPRF6)
- PKCS #11 Secret key encrypt (CSFPSKE and CSFPSKE6)
- PKCS#11 Secret key decrypt (CSFPSKD and CSFPSKD6)
- PKCS #11 Token record create (CSFPTRC and CSFPTRC6)
- PKCS #11 Unwrap key (CSFPUWK and CSFPUWK6)
- PKCS #11 Wrap key (CSFPWPK and CSFPWPK6)

---

## Changes made in Cryptographic Support for z/OS V1R11-R13 (FMID HCR7790)

This document contains information previously presented in *z/OS ICSF Application Programmer's Guide*, SA22-7522-14.

This document is for ICSF FMID HCR7790. This release of ICSF runs on z/OS V1R11, z/OS V1R12, and z/OS V1R13 and only on zSeries hardware.

**New information:**

- Added support for the TR-31 key block format defined by the American National Standards Institute (ANSI). ICSF enables applications to convert a CCA token to a TR-31 key block for export to another party, and to convert an imported TR-31 key block to a CCA token. This enables you to securely exchange keys and their attributes with non-CCA systems. The following callable services have been added to provide this support:
  - TR-31 Export (CSNBT31X and CSNET31X)
  - TR-31 Import (CSNBT31I and CSNET31I) TR-31
  - TR-31 Parse (CSNBT31P and CSNET31P) TR-31
  - TR-31 Optional Data Read (CSNBT31R and CSNET31R)
  - TR-31 Optional Data Build (CSNBT31O and CSNET31O)
- Added new callable service, CVV key combine (CSNBCKC and CSNECKC). This callable service combines 2 single-length CCA internal key tokens into 1 double-length CCA key token containing a CVVKEY-A key type for use with the VISA CVV Service Generate or VISA CVV Service Verify callable services. This combined double-length key satisfies current VISA requirements and eases translation between TR-31 and CCA formats for CVV keys. See “CVV Key Combine (CSNBCKC and CSNECKC)” on page 491 for more information.
- Added support for coordinated and dynamic update of a CKDS. The new callable service Coordinated KDS Administration (CSFCRC and CSFCRC6) which performs a CKDS refresh or reencipher operation while allowing applications to update the CKDS. In a sysplex environment, this callable service enables an application to perform a coordinated sysplex-wide refresh or reencipher operation from a single ICSF instance.
- Added new callable service ECC Diffie-Hellman (CSNDEDH and CSNFEDH), which applications can use to create symmetric key material from a pair of ECC keys using the Elliptic Curve Diffie-Hellman protocol and the static unified model key agreement scheme.
- A new health check, ICSFMIG\_DEPRECATED\_SERV\_WARNINGS, has been added to the Health Checker to detect the use of services that will not be supported in subsequent releases: The deprecated services checked in this release are listed below. These are not supported on post zSeries 900 hardware, and will not be supported in subsequent releases of ICSF.
  - ANSI X9.17 EDC Generate
  - ANSI X9.17 Key Export
  - ANSI X9.17 Key Import
  - ANSI X9.17 Key Translate
  - ANSI X9.17 Transport Key Partial Notarize
  - Ciphertext Translate
  - Ciphertext Translate with ALET
  - Transform CDMF Key
  - Unique Key Derive
  - PKSC Interface Callable Service

You should use the ICSFMIG\_DEPRECATED\_SERV\_WARNINGS check to determine if these services are being used. For more information on this health check, refer to *z/OS Cryptographic Services ICSF Administrator's Guide*.

**Changed information:**

- CKDS Key Record Write2 (CSNBKRW2 and CSNEKRW2)
- Clear PIN Generate (CSNBPGN and CSNEPGN)
- Clear PIN Generate Alternate (CSNBCPA and CSNECPA)
- Control Vector Generate (CSNBCVG and CSNECVG)
- Digital Signature Verify (CSNDDSV and CSNFDSV)
- Encrypted PIN Generate (CSNBEPG and CSNEEPG)
- Encrypted PIN Verify (CSNBPVR and CSNEPVR)
- ICSF Query Facility (CSFIQF and CSFIQF6)
- Key Generate2 (CSNBKGN2 and CSNEKGN2)
- Key Part Import2 (CSNBKPI2 and CSNEKPI2)
- Key Test2 (CSNBKYT2 and CSNEKYT2)
- Key Token Build (CSNBKTB and CSNEKTB)
- Key Token Build2 (CSNBKTB2 and CSNEKTB2)
- Key Translate2 (CSNBKTR2 and CSNEKTR2)
- PKDS Key Record Create (CSNDKRC and CSNFKRC)
- PKDS Key Record Delete (CSNDKRD and CSNFKRD)
- PKDS Key Record Read (CSNDKRR and CSNFKRR)
- PKDS Key Record Write (CSNDKRW and CSNFKRW)
- PKA Decrypt (CSNDPKD and CSNFPKD)
- PKA Encrypt (CSNDPKE and CSNFPKE)
- PKA Key Generate (CSNDPKG and CSNFPKG)
- PKA Key Import (CSNDPKI and CSNFPKI)
- PKA Key Token Change (CSNDKTC and CSNFKTC)
- Restrict Key Attribute (CSNBRKA and CSNERKA)
- Secure Key Import2 (CSNBSKI2 and CSNESKI2)
- Symmetric Algorithm Decipher (CSNBSAD or CSNBSAD1 and CSNESAD or CSNESAD1)
- Symmetric Algorithm Encipher (CSNBSAE or CSNBSAE1 and CSNESAE or CSNESAE1)
- Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2)
- Symmetric Key Generate (CSNDSYG and CSNFSYG)
- Symmetric Key Import (CSNDSYI and CSNFSYI)
- Symmetric Key Export (CSNDSYX and CSNFSYX)
- VISA CVV Service Generate (CSNBCSG and CSNECSG)
- VISA CVV Service Verify (CSNBCSV and CSNECSV)

For clarity:

- CSNBKRC and CSNEKRC, which had been referred to as the "Key Record Create" service, are now referred to as the "CKDS Key Record Create" service
- CSNBKRC2 and CSNEKRC2, which had been referred to as the "Key Record Create2" service, are now referred to as the "CKDS Key Record Create2" service
- CSNBKRD and CSNEKRD, which had been referred to as the "Key Record Delete" service, are now referred to as the "CKDS Key Record Delete" service
- CSNBKRR and CSNEKRR, which had been referred to as the "Key Record Read" service, are now referred to as the "CKDS Key Record Read" service



- CSNBKRR2 and CSNEKRR2, which had been referred to as the "Key Record Read2" service, are now referred to as the "CKDS Key Record Read2" service
- CSNBKRW and CSNEKRW, which had been referred to as the "Key Record Write" service, are now referred to as the "CKDS Key Record Write" service
- CSNBKRW2 and CSNEKRW2, which had been referred to as the "Key Record Write2" service, are now referred to as the "CKDS Key Record Write2" service
- CSNDKRC and CSNFKRC, which had been referred to as the "PKDS Record Create" service, are now referred to as the "PKDS Key Record Create" service
- CSNDKRD and CSNFKRD, which had been referred to as the "PKDS Record Delete" service, are now referred to as the "PKDS Key Record Delete" service
- CSNDKRR and CSNFKRR, which had been referred to as the "PKDS Record Read" service, are now referred to as the "PKDS Key Record Read" service
- CSNDKRW and CSNFKRW, which had been referred to as the "PKDS Record Write" service, are now referred to as the "PKDS Key Record Write" service

References to the IBM @server zSeries 800 (z800) do not appear in this information. Be aware that the documented notes and restrictions for the IBM @server zSeries 900 (z900) also apply to the z800.

---

## Changes made in Cryptographic Support for z/OS V1R10-R12 (FMID HCR7780)

This document contains information previously presented in *z/OS ICSF Application Programmer's Guide*, SA22-7522-13.

This document is for ICSF FMID HCR7780. This release of ICSF runs on z/OS V1R10, z/OS V1R11, and z/OS V1R12 and only on zSeries hardware. Although originally available as a Web Deliverable, this FMID was subsequently included in the base of z/OS Version 1 Release 13.

### New information:

- All ICSF callable services now support invocation in AMODE(64). Previously, only certain callable services had this support. Applications that are written for AMODE(64) operation must be linked with the ICSF 64-bit service stubs.
- Added information for HMAC key support. This support is to be enabled with the PTF for APAR OA33260, planned for February 2011 availability.
  - Added the following services to support CCA key management of HMAC keys:
    - Key Generate2 (CSNBKGN2 and CSNEKGN2)
    - Key Part Import2 (CSNBKPI2 and CSNEKPI2)
    - Key Test2 (CSNBKYT2 and CSNEKYT2)
    - Key Token Build2 (CSNBKTB2 and CSNEKTB2)
    - Restrict Key Attribute (CSNBRKA and CSNERKA)
    - Secure Key Import2 (CSNBSKI2 and CSNESKI2)
    - Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2)
  - The following services for Dynamic CKDS update were added for use with HMAC tokens. These services must be used with HMAC tokens, but also support existing DES and AES tokens.
    - Key Record Create2 (CSNBKRC2 and CSNEKRC2)
    - Key Record Read2 (CSNBKRR2 and CSNEKRR2)

- Key Record Write2 (CSNBKRW2 and CSNEKRW2)
- Added the following services to support the generation and verification of MACs using keyed-HMAC algorithm:
  - HMAC Generate (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1)
  - HMAC Verify (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1)
- Added the following modes of operation for protecting data to the Symmetric Key Encipher and Symmetric Key Decipher callable services:
  - Cipher Block Chaining with Ciphertext Stealing (CBC-CS) Mode
  - Cipher Feedback (CFB) Mode
  - Output Feedback (OFB) Mode
  - Galois/Counter Mode (GCM)
- Added an enhanced method of key wrapping that is ANSI X9.24 compliant. See “Key Wrapping” on page 19 for more information.
- Added Elliptic Curve Cryptography (ECC) support to the following callable services:
  - Digital Signature Generate (CSNDDSG and CSNFD SG)
  - Digital Signature Verify (CSNDDSV and CSNFD SV)
  - PKA Key Generate (CSNDPKG and CSNFPKG)
  - PKA Key Import (CSNDPKI and CSNFPKI)
  - PKA Key Token Build (CSNDPKB and CSNFPKB)
  - PKA Key Token Change (CSNDKTC and CSNFKTC)
  - PKA Public Key Extract (CSNDPKX and CSNFPKX)
  - PKDS Record Create (CSNDKRC and CSNFKRC)
  - PKDS Record Delete (CSNDKRD and CSNFKRD)
  - PKDS Record Read (CSNDKRR and CSNFKRR)
  - PKDS Record Write (CSNDKRW and CSNFKRW)

#### **Changed information**

- ANSI X9.17 EDC Generate - CSNGEGN
- ANSI X9.17 Key Export - CSNGKEX
- ANSI X9.17 Key Import - CSNGKIM
- ANSI X9.17 Key Translate - CSNGKTR
- ANSI X9.17 Transport Key Partial Notarize - CSNGTKN
- Ciphertext Translate - CSNECTT and CSNECTT1
- Clear PIN Encrypt - CSNECPE
- Clear PIN Generate - CSNEPGN
- Clear PIN Generate Alternate - CSNECPA
- Control Vector Generate - CSNECVG
- Control Vector Translate - CSNECVT
- Cryptographic Variable Encipher - CSNECVE
- Data Key Export - CSNEDKX
- Data Key Import - CSNEDKM
- Decipher - CSNEDEC and CSNEDEC1
- Decode - CSNEDCO
- Diversified Key Generate - CSNEDKG

- Encipher - CSNEENC and CSNEENC1
- Encode - CSNEECO
- Encrypted PIN Generate - CSNEEPG
- Encrypted PIN Translate - CSNEPTR
- Encrypted PIN Verify - CSNEPVR
- HMAC Generate - CSNEHMG and CSFEHMG1
- HMAC Verify - CSNEHMGV and CSFEHMGV1
- Key Export - CSNEKEX
- Key Generate2 - CSNEKGN2
- Key Import - CSNEKIM
- Key Part Import - CSNEKPI
- Key Part Import2 - CSNEKPI2
- Key Record Create - CSNEKRC
- Key Record Create2 - CSNEKRC2
- Key Record Delete - CSNEKRD
- Key Record Read - CSNEKRR
- Key Record Read2 - CSNEKRR2
- Key Record Write - CSNEKRW
- Key Record Write2 - CSNEKRW2
- Key Test - CSNEKYT
- Key Test2 - CSNEKYT2
- Key Test Extended - CSNEKYTX
- Key Token Build - CSNEKTB
- Key Token Build2 - CSNEKTB2
- Key Translate - CSNEKTR
- MAC Generate - CSNEMGN and CSNEMGN1
- MAC Verify - CSNEMVR and CSNEMVR1
- MDC Generate - CSNEMDG and CSNEMDG1
- Multiple Secure Key Import - CSNESKM
- PCI Interface - CSFPCI and CSFPCI6
- PIN Change/Unblock - CSNEPCU
- PKA Key Token Change - CSNDKTC
- PKDS Record Read - CSNFKRR
- PKDS Record Write - CSNFKRW
- Prohibit Export - CSNEPEX
- Prohibit Export Extended - CSNEPEXX
- Remote Key Export - CSNFRKX
- Restrict Key Attribute - CSNBRKA and CSNERKA
- Secure Key Import - CSNESKI
- Secure Key Import2 - CSNESKI2
- Secure Messaging for Keys - CSNESKY
- Secure Messaging for PINS - CSNESPN
- SET Block Compose - CSNFSBC
- SET Block Decompose - CSNFSBD
- Symmetric Key Generate - CSNFSYG

- Transaction Validation - CSNETRV
- Transform CDMF Key - CSNETCK
- Trusted Block Create - CSNFTBC
- Unique Key Derive - CSFUDK6
- VISA CVV Service Generate - CSNECSG
- VISA CVV Service Verify - CSNECSV
- PKCS #11 Secret Key Encrypt (CSFPSKE)
- PKCS #11 One-way hash, sign, or verify (CSFPOWH). This was previously referred to as PKCS #11 One-way hash generate (CSFPOWH).

---

## Changes made in Cryptographic Support for z/OS V1R9-R11 (FMID HCR7770)

This document contains information previously presented in *z/OS ICSF Application Programmer's Guide*, SA22-7522-12, which supports z/OS Version 1 Release 10.

This document is for ICSF FMID HCR7770. This release of ICSF runs on z/OS V1R9 and z/OS V1R10 and only on zSeries hardware. Although originally available as a Web Deliverable, this FMID was subsequently included in the base of z/OS Version 1 Release 12.

### New information:

- Added new callable service PKA Key Translate (CSNDPKT and CSNFPKT). Using this callable service, applications can translate a source CCA RSA key token into a target external smart card key token.
- Added new callable services for managing PKCS #11 tokens and objects. These additional services are:
  - PKCS #11 Derive key (CSFPDVK)
  - PKCS #11 Derive multiple keys (CSFPDMK)
  - PKCS #11 Generate HMAC (CSFPHMG)
  - PKCS #11 Generate key pair (CSFPGKP)
  - PKCS #11 Generate secret key (CSFPGSK)
  - PKCS #11 One-way hash generate (CSFPOWH)
  - PKCS #11 Private key sign (CSFPPKS)
  - PKCS #11 Pseudo-random function (CSFPPRF)
  - PKCS #11 Public key verify (CSFPPKV)
  - PKCS #11 Secret key decrypt (CSFPSKD)
  - PKCS #11 Secret key encrypt (CSFPSKE)
  - PKCS #11 Unwrap key (CSFPUWK)
  - PKCS #11 Verify HMAC (CSFPHMV)
  - PKCS #11 Wrap key (CSFPWPK)
- Added information for the Crypto Express3 Coprocessor.

### Changed information:

- The Symmetric Key Export and Symmetric Key Import callable services now support invocation in AMODE(64).

The Symmetric Key Encipher and Symmetric Key Decipher callable services now support an encrypted key in the CKDS. This enables applications to leverage the performance capabilities of CPACF when enciphering/deciphering data using encrypted symmetric keys.

- The ICSF Query Algorithm (CSFIQA and CSFIQA6) utility and ICSF Query Facility (CSFIQF and CSFIQF6) updated to return additional data.

This document contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.



---

## Part 1. IBM CCA Programming

IBM CCA Programming introduces programming for the IBM CCA, AES cryptography, DES cryptography and PKA cryptography. It explains how to use DES, AES and PKA callable services.

**Note:** References to the IBM @server zSeries 800 (z800) do not appear in this information. Be aware that the documented notes and restrictions for the IBM @server zSeries 900 (z900) also apply to the z800.





---

## Chapter 1. Introducing Programming for the IBM CCA

ICSF provides access to cryptographic functions through callable services, which are also known as verbs. A callable service is a routine that receives control using a CALL statement in an application language.

Prior to invoking callable services in an application program, you must link them into the application program. See “Linking a Program with the ICSF Callable Services” on page 13.

To invoke the callable service, the application program must include a procedure call statement that has the entry point name and parameters for the callable service. The parameters that are associated with a callable service provide the only communication between the application program and ICSF.

---

### ICSF Callable Services Naming Conventions

The ICSF callable services generally follow the naming conventions outlined in the following table.

There are five exceptions where the CSFzzz names would collide and in those cases, the CSFzzz alias is CSFPzzz instead: PKDS Key Record Create (CSFPKRC), PKDS Key Record Delete (CSFPKRD), PKDS Key Record Read (CSFPKRR), PKDS Key Record Write (CSFPKRW), PKA Key Token Change (CSFPKTC),

In the following table, zzz is a 3- or 4-letter service name, such as ENC for the Encipher service or PKG for the PKA Key Generate service. Not all CSNBzzz/CSNEzzz services have ALET-qualified entry points (where certain parameters can be in a dataspace or an address space other than the caller's). See each specific service for details.

*Table 1. ICSF Callable Services Naming Conventions*

This callable service prefix:	Identifies:	
CSNBzzz / CSFzzz	31-bit	Symmetric Key Services and Hashing Services
CSNBzzz1 / CSFzzz1	31-bit ALET-qualified	
CSNEzzz / CSFzzz6	64-bit	
CSNEzzz1 / CSFzzz16	64-bit ALET-qualified	
CSNDzzz / CSFzzz	31-bit	Asymmetric Key Services
CSNFzzz / CSFzzz6	64-bit	
CSNAzzz / CSFAzzz	31-bit	ANSI X9.17 Services
CSNGzzz / CSFAzzz6	64-bit	
CSFPzzz	31-bit	PKCS #11 Services
CSFPzzz6	64-bit	
CSFzzz	31-bit	Utility Services and TKE Workstation Interfaces
CSFzzz6	64-bit	

---

## Callable Service Syntax

This publication uses a general call format to show the name of the ICSF callable service and its parameters. An example of that format is shown here:

```
CALL CSNBxxx (return_code,  
              reason_code,  
              exit_data_length,  
              exit_data,  
              parameter_5,  
              parameter_6,  
              .  
              .  
              .  
              parameter_N)
```

where CSNBxxx is the name of the callable service. The return code, reason code, exit data length, exit data, parameter 5 through parameter *N* represent the parameter list. The call generates a fixed length parameter list. You must supply the parameters in the order shown in the syntax diagrams. “Parameter Definitions” on page 6 describes the parameters in more detail.

ICSF callable services can be called from application programs written in a number of high-level languages as well as assembler. The high-level languages are:

- C
- COBOL
- FORTRAN
- PL/I

The ICSF callable services comply with the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface. The services can be invoked using the generic format, **CSNBxxx**. Use the generic format if you want your application to work with more than one cryptographic product. The format **CSFxxxx** can be used in place of **CSNBxxx**. Otherwise, use the **CSFxxxx** format.

Specific formats for the languages that can invoke ICSF callable services are as follows:

### C

```
CSNBxxxx (return_code,reason_code,exit_data_length,exit_data,  
          parameter_5,...parameter_N)
```

### COBOL

```
CALL 'CSNBxxxx' USING return_code,reason_code,exit_data_length,  
exit_data,parameter_5,...parameter_N
```

### FORTRAN

```
CALL CSNBxxxx (return_code,reason_code,exit_data_length,exit_data,  
parameter_5,...parameter_N)
```

### PL/I

```
DCL CSNBxxxx ENTRY OPTIONS(ASM);  
CALL CSNBxxxx return_code,reason_code,exit_data_length,exit_data,  
parameter_5,...parameter_N;
```

**Assembler** language programs must use standard linkage conventions when invoking ICSF callable services. An example of how an assembler language program can invoke a callable service is shown as follows:

```
CALL CSNBxxxx,(return_code,reason_code,exit_data_length,exit_data,  
parameter_5,...parameter_N)
```

Coding examples using the high-level languages are shown in Appendix D, “Coding Examples.”

## Callable Services with ALET Parameters

Some callable services have an alternate entry point (with ALET parameters—for data that resides in data spaces). They are in the format of *CSNBxxx1* as shown in the following table. For the associated 64-bit versions of the callable services (*CSNExxx*), the ALET-qualified versions are in the format *CSNExxx1*.

Verb	Callable Service without ALET	Callable Service with ALET
Ciphertext translate	CSNBCTT	CSNBCTT1
Ciphertext translate2	CSNBCTT2	CSNBCTT3
Decipher	CSNBDEC	CSNBDEC1
Encipher	CSNBENC	CSNBENC1
HMAC Generate	CSNBHMG	CSNBHMG1
HMAC Verify	CSNBHMV	CSNBHMV1
MAC generate	CSNBMGN	CSNBMGN1
MAC verify	CSNBMVR	CSNBMVR1
MDC generate	CSNBMDG	CSNBMDG1
One way hash generate	CSNBOWH	CSNBOWH1
Symmetric algorithm decipher	CSNBSAD	CSNBSAD1
Symmetric algorithm encipher	CSNBSAE	CSNBSAE1
Symmetric key decipher	CSNBSYD	CSNBSYD1
Symmetric key encipher	CSNBSYE	CSNBSYE1
Symmetric MAC generate	CSNBSMG	CSNBSMG1
Symmetric MAC verify	CSNBSMV	CSNBSMV1

When choosing which service to use, consider the fact that:

- Callable services that do not have an ALET parameter require data to reside in the caller's primary address space. A program using these services adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- Callable services that have an ALET parameter allow data to reside either in the caller's primary address space or in a data space. This can allow you to encipher more data with one call. However, a program using these services does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to running with other cryptographic products that follow this programming interface.

## Rules for Defining Parameters and Attributes

These rules apply to the callable services:

- Parameters are required and positional.
- Each parameter list has a fixed number of parameters.
- Each parameter is defined as an integer or a character string. Null pointers are not acceptable for any parameter.
- Keywords passed to the callable services, such as TOKEN, CUSP, and FIRST can be in lower, upper, or mixed case. The callable services fold them to uppercase prior to using them.

Each callable service defines its own list of parameters. The entire list must be supplied on every call. If you do not use a specific parameter, you must supply that parameter with hexadecimal zeros or binary zeros.

Parameters are passed to the callable service. All information that is exchanged between the application program and the callable service is through parameters passed on the call.

Each parameter definition begins with the direction that the data flows and the attributes that the parameter must possess (called “type”). This describes the direction.

Direction	Meaning
<b>Input</b>	The application sends ( <i>supplies</i> ) the parameter to the callable service. The callable service does not change the value of the parameter.
<b>Output</b>	The callable service <i>returns</i> the parameter to the application program. The callable service may have changed the value of the parameter on return.
<b>Input/Output</b>	The application sends ( <i>supplies</i> ) the parameter to the callable service. The callable service may have changed the value of the parameter on return.

This describes the attributes or type.

Type	Meaning
<b>Integer (I)</b>	A 4-byte (32-bit), twos complement, binary number that has sign significance.
<b>String</b>	A series of bytes where the sequence of the bytes must be maintained. Each byte can take on any bit configuration. The string consists only of data bytes. No string terminators, field-length values, or type-casting parameters are included. The maximum size of a string is X'7FFFFFFF' or 2 gigabytes. In some of the callable services, the length of some string data has an upper bound defined by the installation. The upper bound of a string can also be defined by the service.

#### Alphanumeric character string

A string of bytes in which each byte represents characters from this set:

Character	EBCDIC Value	Character	EBCDIC Value	Character	EBCDIC Value
A-Z		(	X'4D'	/	X'61'
a-z		)	X'5D'	,	X'6B'
0-9		+	X'4E'	%	X'6C'
Blank	X'40'	&	X'50'	?	X'6F'
*	X'5C'	.	X'4B'	:	X'7A'
<	X'4C'	;	X'5E'	=	X'7E'
>	X'6E'	-	X'60'	'	X'7D'

## Parameter Definitions

This topic describes these parameters, which are used by most of the callable services:

- *Return\_code*
- *Reason\_code*

- *Exit\_data\_length*
- *Exit\_data*
- *Key\_identifier*

**Note:** The *return\_code* parameter, the *reason\_code* parameter, the *exit\_data\_length* parameter, and the *exit\_data* parameter are required with every callable service.

## Return and Reason Codes

*Return\_code* and *reason\_code* parameters return integer values upon completion of the call.

### Return\_code

The return code parameter contains the general results of processing as an integer.

Table 2 shows the standard return code values that the callable services return. A complete list of return codes is shown in Appendix A, “ICSF and TSS Return and Reason Codes.”

*Table 2. Standard Return Code Values From ICSF Callable Services*

Value Hex (Decimal)	Meaning
00 (00)	Successful. Normal return.
04 (04)	A warning. Execution was completed with a minor, unusual event encountered.
08 (08)	An application error occurred. The callable service was stopped due to an error in the parameters. Or, another condition was encountered that needs to be investigated.
0C (12)	Error. ICSF is not active or an environment error was detected.
10 (16)	System error. The callable service was stopped due to a processing error within the software or hardware.

Generally, PCF macros will receive identical error return codes if they execute on PCF or on ICSF. A single exception has been noted: if a key is installed on the ICSF CKDS with the correct label but with the wrong key type, PCF issues a return code of 8, indicating that the key type was incorrect. ICSF issues a return code of 12, indicating that the key could not be found.

### Reason\_code

The reason code parameter contains the results of processing as an integer. You can specify which set of reason codes (ICSF or TSS) are returned from callable services. The default value is ICSF. For more information about the REASONCODES installation option, see *z/OS Cryptographic Services ICSF System Programmer's Guide*. Different results are assigned to unique reason code values under a return code.

A list of reason codes is shown in Appendix A, “ICSF and TSS Return and Reason Codes.”

## Exit Data Length and Exit Data

The *exit\_data\_length* and *exit\_data* parameters are described here. The parameters are input to all callable services. Although all services require these parameters, several services ignore them.

**Exit\_data\_length**

The integer that has the string length of the data passed to the exit. The data is identified in the *exit\_data* parameter.

**Exit\_data**

The installation exit data string that is passed to the callable service's preprocessing exit. The installation exit can use the data for its own processing.

ICSF provides two installation exits for each callable service. The preprocessing exit is invoked when an application program calls a callable service, but prior to when the callable service starts processing. For example, this exit is used to check or change parameters passed on the call or to stop the call. It can also be used to perform additional security checks.

The post-processing exit is invoked when the callable service has completed processing, but prior to when the callable service returns control to the application program. For example, this exit can be used to check and change return codes from the callable service or perform clean-up processing.

For more information about the exits, see *z/OS Cryptographic Services ICSF System Programmer's Guide*.

**Key Identifier for Key Token**

A *key identifier* for a key token is an area that contains one of these:

- **Key label** identifies keys that are in the CKDS or PKDS. Ask your ICSF administrator for the key labels that you can use.
- **Key token** can be either an internal key token, an external key token, or a null key token. Key tokens are generated by an application (for example, using the key generate callable service), or received from another system that can produce external key tokens.

An **internal key token** can be used only on ICSF because the master key encrypts the key value. Internal key tokens contain keys in operational form only.

An **external key token** can be exchanged with other systems because a transport key that is shared with the other system encrypts the key value. External key tokens contain keys in either exportable or importable form.

A **null key token** can be used to import a key from a system that cannot produce external key tokens. A null key token contains a key encrypted under an importer key-encrypting key but does not contain the other information present in an external key token.

The term *key identifier* is used to indicate that different inputs are possible for a parameter. One or more of the previously described items may be accepted by the callable service.

**Key Label:** If the first byte of the key identifier is greater than X'40', the field is considered to be holding a **key label**. The contents of a key label are interpreted as a pointer to a CKDS or PKDS key entry. The key label is an indirect reference to an internal key token.

A key label is specified on callable services with the *key\_identifier* parameter as a 64-byte character string, left-justified, and padded on the right with blanks. In most cases, the callable service does not check the syntax of the key label beyond the



first byte. One exception is the CKDS key record create callable service which enforces the KGUP rules for key labels unless syntax checking is bypassed by a preprocessing exit.

A key label has this form:

Offset	Length	Data
00-63	64	Key label name

There are some general rules for creating labels for CKDS key records.

- Each label can consist of up to 64 characters. The first character must be alphabetic or a national character (#, \$, @). The remaining characters can be alphanumeric, a national character (#, \$, @), or a period (.).
- All alphabetic characters must be upper case (A-Z). All labels in the key data sets are created with upper case characters.
- Labels must be unique for DATA, DATAXLAT, MAC, MACVER, DATAM, and DATAMV keys.
- Transport and PIN keys can have duplicate labels for different key types. Keys that use the dynamic CKDS update services to create or update, however, must have unique key labels.
- Labels must be unique for any key record, including transport and PIN keys, created or updated using the dynamic CKDS update services.

## Invocation Requirements

Applications that use ICSF callable services must meet these invocation requirements:

- All output parameters must be in storage that the caller is allowed to modify in their execution key.
- All input parameters must be in storage that the caller is allowed to read in their execution key.
- Data can be located higher or lower than 16Mb but must be 31-bit addressable. Data can be located above 2Gb if the service is invoked in AMODE(64)
- Problem or supervisor state
- Any PSW key
- Task mode or Service Request Block (SRB) mode
- No mode restrictions
- Enabled for interrupts
- No locks held

The exceptions to this list are documented with the individual callable services.

All ICSF callable services support invocation in AMODE(64). Applications which are written for AMODE(64) operation must be linked with the ICSF 64-bit service stubs, and must invoke the service with the appropriate service name. (Refer to the description of the individual callable service to determine the service name to be used.)

## Security Considerations

Your installation can use the Security Server RACF or an equivalent product to control who can use ICSF callable services or key labels. Prior to using an ICSF callable service or a key label, ask your security administrator to ensure that you have the necessary authorization. For more information, see *z/OS Security Server RACF Security Administrator's Guide*.

HCR7751 and later supports a key store policy using the RACF XFACILIT class. See *z/OS Security Server RACF Security Administrator's Guide*.

RACF does not control all services. The usage notes topic in the callable service description will highlight those services which are not controlled.

---

## Performance Considerations

In most cases, the z/OS operating system dispatcher provides optimum performance. However, if your application makes extensive use of ICSF functions, you should consider using one or both of these:

- **CCF Systems Only:** If your application runs in SRB mode, use the SCHEDULE macro or IEAAFFN callable service. You should consider scheduling an SRB to run on a processor with the cryptographic feature installed (using the FEATURE=CRYPTO keyword on the SCHEDULE macro). For more information on the SCHEDULE macro, refer to *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU*.

**Restriction:** The FEATURE=CRYPTO keyword should not be specified when running on an IBM @server zSeries 990.

- Use the IEAAFFN callable service (processor affinity) to avoid system overhead in selecting which processor your program (specifically, a particular TCB in the application) runs in. Note that you do **not** have to use the IEAAFFN service to ensure that the system runs a program on a processor with a cryptographic feature; the system ensures that automatically. However, you can avoid some of the system overhead involved in the selection process by using the IEAAFFN service, thus improving the program's performance. For more information on using the IEAAFFN callable service, refer to *z/OS MVS Programming: Callable Services for High-Level Languages*.

IBM recommends that you run applications first without using these options. Consider these options when you are tuning your application for performance. Use these options only if they improve the performance of your application.

---

## Special Secure Mode

Special secure mode is a special processing mode in which:

- The Secure Key Import, Secure Key Import2, and Multiple Secure Key Import callable services, which work with clear keys, can be used.
- The Clear PIN Generate callable service, which works with clear PINs, can be used.
- The Symmetric Key Generate callable service with the "IM" keyword (the DES enciphered key is enciphered by an IMPORTER key) can be used (**CCF Systems Only**).
- The key generator utility program (KGUP) can be used to enter clear keys into the CKDS.

To use special secure mode, several conditions must be met.

- The installation options data set must specify YES for the SSM installation option.

For information about specifying installation options, see *z/OS Cryptographic Services ICSF System Programmer's Guide*.

This is required for all systems.

- The environmental control mask (ECM) must be configured to permit special secure mode.

The ECM is a 32-bit mask defined for each cryptographic domain during hardware installation. The second bit in this mask must have been turned on to enable special secure mode. The default is to have this bit turned on in the ECM. The bit can only be turned off/on through the optional TKE Workstation.

This is required for systems with the Cryptographic Coprocessor Feature.

- If you are running in LPAR mode, special secure mode must be enabled.

On the IBM @server zSeries 900, you enable special secure mode during activation using the Crypto page of the Customize Activation Profiles task. When activated, you can enable or disable special secure mode on the Change LPAR Crypto task. Both of these tasks can be accessed from the Hardware Management Console.

This is required for systems with the Cryptographic Coprocessor Feature.

For the IBM @server zSeries 900 with TKE, TKE can disable/enable special secure mode.

---

## Using the Callable Services

This topic discusses how ICSF callable services use the different key types and key forms. It also provides suggestions on what to do if there is a problem.

ICSF provides callable services that perform cryptographic functions. You call and pass parameters to a callable service from an application program. Besides the callable services ICSF provides, you can write your own callable services called *installation-defined callable services*. **Note that only an experienced system programmer should attempt to write an installation-defined callable service.**

To write an installation-defined callable service, you must first write the callable service and link-edit it into a load module. Then define the service in the installation options data set.

You must also write a service stub. To execute an installation-defined callable service, you call a service stub from your application program. In the service stub, you specify the service number that identifies the callable service.

For more information about installation-defined callable services, see *z/OS Cryptographic Services ICSF System Programmer's Guide*.

## When the Call Succeeds

If the return code is 0, ICSF has successfully completed the call. If a reason code other than 0 is included, refer to Appendix A, "ICSF and TSS Return and Reason Codes," on page 781, for additional information. For instance, reason code 10000 indicates the key in the key identifier (or more than one key identifier, for services that use two internal key identifiers) has been reenciphered from encipherment under the old master key to encipherment under the current master key. Keys in external tokens are not affected by this processing because they contain keys enciphered under keys other than the host master key. If you manage your key identifiers on disk, then reason code 10000 should be a "trigger" to store these updated key identifiers back on disk.

Your program can now continue providing its function, but you may want to communicate the key that you used to another enterprise. This process is exporting a key.

If you want to communicate the key that you are using to a cryptographic partner, there are several methods to use:

- For DATA keys only, call the data key export callable service. You now have a DATA key type in exportable form.
- Call the key export callable service. You now have the key type in exportable form.
- When you use the key generate callable service to create your operational or importable key form, you can create an exportable form, **at the same time**, and you now have the key type, in exportable form, at the same time as you get the operational or importable form.

## When the Call Does Not Succeed

Now you have planned your use of the ICSF callable services, made the call, but the service has completed with a return and reason codes other than zero.

If the return code is **4**, there was a minor problem. For example, reason code 8004 indicates the trial MAC that was supplied does not match the message text provided.

If the return code is **8**, there was a problem with one of your parameters. Check the meaning of the reason code value, correct the parameter, and call the service again. You may go through this process several times prior to succeeding.

If the return code is **12**, ICSF is not active, or has no access to cryptographic units, or has an environmental problem. Check with your ICSF administrator.

If the return code is **16**, the service has a serious problem that needs the help of your system programmer.

There are several reason codes that can occur when you have already fully debugged and tested your program. For example:

- Reason code 10004 indicates that you provided a key identifier that holds a key enciphered under a host master key. The host master key is not installed in the cryptographic unit. If this happens, you have to go back and import your importable key form again and call the service again. You need to build this flow into your program logic.
- Reason code 10012 indicates a key corresponding to the label that you specified is not in the CKDS or PKDS. Check with your ICSF administrator to see if the label is correct.
- Reason code 3063 indicates RACF failed your request to use a token.
- Reason code 16000 indicates RACF failed your request to use a service.
- Reason code 16004 indicates RACF failed your request to use the key label. Examine your CSFKEYS profiles and key store policies for possible errors.

Return and reason codes are described in Appendix A, "ICSF and TSS Return and Reason Codes," on page 781.

---

## Linking a Program with the ICSF Callable Services

To link the ICSF callable services into an application program, you can use these sample JCL statements. In the SYSLIB concatenation, include the CSF.SCSFMOD0 module in the link edit step. This provides the application program access to all ICSF callable services (those that can be invoked in AMODE(24)/AMODE(31) as well as those that can be invoked in AMODE(64)).

```
//LKEDENC JOB
//*-----*
//*
/* The JCL links the ICSF encipher callable service, CSNBENC,
/* into an application called ENCIPHER.
/*
/*-----*
//LINK EXEC PGM=IEWL,
// PARM='XREF,LIST,LET'
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(10,10))
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DSN=CSF.SCSFMOD0,DISP=SHR * SERVICES ARE IN HERE
//SYSLMOD DD DSN=MYAPPL.LOAD,DISP=SHR * MY APPLICATION LIBRARY
//SYSLIN DD DSN=MYAPPL.ENCIPHER.OBJ,DISP=SHR * MY ENCIPHER PROGRAM
// DD *
      ENTRY ENCIPHER
      NAME ENCIPHER(R)
/*
```



---

## Chapter 2. Introducing Symmetric Key Cryptography and Using Symmetric Key Callable Services

The Integrated Cryptographic Service Facility protects data from unauthorized disclosure or modification. ICSF protects data stored within a system, stored in a file off a system on magnetic tape, and sent between systems. ICSF also authenticates the identity of customers in the financial industry and authenticates messages from originator to receiver. It uses cryptography to accomplish these functions.

ICSF provides access to cryptographic functions through callable services. A callable service is a routine that receives control using a CALL statement in an application language. Each callable service performs one or more cryptographic functions, including:

- Generating and managing cryptographic keys
- Enciphering and deciphering data with encrypted keys using the U.S. National Institute of Standards and Technology (NIST) Data Encryption Standard (DES), Advanced Encryption Standard (AES) or the Commercial Data Masking Facility (CDMF)
- Enciphering and deciphering data with clear keys using either the NIST Data Encryption Standard (DES), or Advanced Encryption Standard (AES)
- Transforming a CDMF DATA key to a transformed shortened DES key
- Reenciphering text from encryption under one key to encryption under another key
- Encoding and decoding data with clear keys
- Generating random numbers
- Ensuring data integrity and verifying message authentication
- Generating, verifying, and translating personal identification numbers (PINs) that identify a customer on a financial system

This topic provides an overview of the symmetric key cryptographic functions provided in ICSF, explains the functions of the cryptographic keys, and introduces the topic of building key tokens. Many services have hardware requirements. See each service for details.

---

### Functions of the Symmetric Cryptographic Keys

ICSF provides functions to create, import, and export AES, DES, and HMAC keys. This topic gives an overview of these cryptographic keys. Detailed information about how ICSF organizes and protects keys is in *z/OS Cryptographic Services ICSF Administrator's Guide*.

ICSF supports two types of symmetric key tokens: fixed-length and variable-length. In fixed-length key tokens, key type and usage are defined by the control vector. In variable-length key tokens, the key type and usage are defined in the associated data section. The control vector and associated data section are cryptographically bound to the encrypted key value in the token.



## Key Separation

The cryptographic feature controls the use of keys by separating them into unique types, allowing you to use a specific type of key only for its intended purpose. For example, a key used to protect data cannot be used to protect a key.

An ICSF system has only one DES master key and one AES master key. However, to provide for key separation, the cryptographic feature automatically encrypts each type of key in a fixed-length token under a unique variation of the master key. Each variation of the master key encrypts a different type of key. Although you enter only one master key, you have a unique master key to encrypt all other keys of a certain type.

**Note:** In PCF, key separation applies only to keys enciphered under the master key (keys in operational form). In ICSF, key separation also applies to keys enciphered under transport keys (keys in importable or exportable form). This allows the creator of a key to transmit the key to another system and to enforce its use at the other system.

## Master Key Variant for Fixed-length Tokens

Whenever the master key is used to encipher a key, the cryptographic coprocessor produces a variation of the master key according to the type of key the master key will encipher. These variations are called *master key variants*. The cryptographic coprocessor creates a master key variant by exclusive ORing a fixed pattern, called a *control vector*, onto the master key. A unique control vector is associated with each type of key. For example, all the different types of data-encrypting, PIN, MAC, and transport keys each use a unique control vector which is exclusive ORed with the master key in order to produce the variant. The different key types are described in "Types of Keys" on page 19.

Each master key variant protects a different type of key. It is similar to having a unique master key protect all the keys of a certain type.

The master key, in the form of master key variants, protects keys operating on the system. A key can be used in a cryptographic function only when it is enciphered under a master key. When systems want to share keys, transport keys are used to protect keys sent outside of systems. When a key is enciphered under a transport key, the key cannot be used in a cryptographic function. It must first be brought on to a system and enciphered under the system's master key, or exported to another system where it will then be enciphered under that system's master key.

## Transport Key Variant for Fixed-length Tokens

Like the master key, ICSF creates variations of a transport key to encrypt a key according to its type. This allows for key separation when a key is transported off the system. A *transport key variant*, also called *key-encrypting key variant*, is created the same way a master key variant is created. The transport key's clear value is exclusive ORed with a control vector associated with the key type of the key it protects.

**Note:** To exchange keys with systems that do not recognize transport key variants, ICSF allows you to encrypt selected keys under a transport key itself, not under the transport key variant. For more information, see the 'Transport keys (or key-encrypting keys)' list item in "Types of Keys" on page 19.

## Key Forms

A key that is protected under the master key is in *operational form*, which means ICSF can use it in cryptographic functions on the system.

When you store a key with a file or send it to another system, the key is enciphered under a transport key rather than the master key because, for security reasons, the key should no longer be active on the system. When ICSF enciphers a key under a transport key, the key is not in operational form and cannot be used to perform cryptographic functions.

When a key is enciphered under a transport key, the sending system considers the key in *exportable form*. The receiving system considers the key in *importable form*. When a key is reenciphered from under a transport key to under a system's master key, it is in operational form again.

Enciphered keys appear in three forms. The form you need depends on how and when you use a key.

- **Operational** key form is used at the local system. Many callable services can *use* an operational key form.  
The key token build, key generate, key import, data key import, clear key import, multiple clear key import, secure key import, and multiple secure key import callable services can *create* an operational key form.
- **Exportable** key form is transported to another cryptographic system. It can only be passed to another system. The ICSF callable services cannot use it for cryptographic functions. The key generate, data key export, and key export callable services produce the exportable key form.
- **Importable** key form can be transformed into operational form on the local system. The key import callable service (CSNBKIM) and the Data key import callable service (CSNBDKM) can *use* an importable key form. Only the key generate callable service (CSNBKGN) can *create* an importable key form. The secure key import (CSNBSKI) and multiple secure key import (CSNBSKM) callable services can convert a clear key into an importable key form.

For more information about the key types, see either “Functions of the Symmetric Cryptographic Keys” on page 15 or the *z/OS Cryptographic Services ICSF Administrator's Guide*. See “Key Forms and Types Used in the Key Generate Callable Service” on page 67 for more information about key form.

### DES Key Flow

The conversion from one key to another key is considered to be a one-way flow. An operational key form cannot be turned back into an importable key form. An exportable key form cannot be turned back into an operational or importable key form. The flow of ICSF key forms can only be in one direction:

IMPORTABLE -to> OPERATIONAL -to> EXPORTABLE

## Key Token

ICSF supports two types of symmetric key tokens: fixed-length and variable-length. The fixed-length token is a 64-byte field composed of a key value and control information in the control vector. The variable-length token is composed of a key value and control information in the associated data section of the token. The control information is assigned to the key when ICSF creates the key. The key token can be either an internal key token, an external key token, or a null key token. Through the use of key tokens, ICSF can:

- Support continuous operation across a master key change

- Control use of keys in cryptographic services

If the first byte of the key identifier is X'01', the key identifier is interpreted as an **internal key token**. An internal key token is a token that can be used only on the ICSF system that created it (or another ICSF system with the same host master key). It contains a key that is encrypted under the master key.

An application obtains an internal key token by using one of the callable services such as those listed here. The callable services are described in detail in Chapter 5, “Managing Symmetric Cryptographic Keys.”

- CKDS Key record read and CKDS key record read2
- Clear key import
- Data key import
- Key generate and Key generate2
- Key import
- Key part import and Key part import2
- Key token build and Key token build2
- Multiple secure key import
- Multiple clear key import
- Secure key import and secure key import2
- Symmetric key import2

The master keys may be dynamically changed between the time that you invoke a service, such as the key import callable service to obtain a key token, and the time that you pass the key token to the encipher callable service. When a change to the master key occurs, ICSF reenciphers the caller's key from under the old master key to under the new master key. A Return Code of 0 with a reason code of 10000 notifies you that ICSF reenciphered the key. For information on reenciphering the CKDS or the PKDS, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

**Attention:** If an internal key token held in user storage is not used while the master key is changed twice, the internal key token is no longer usable. (See “Other Considerations” on page 22 for additional information.)

For debugging information, see Appendix B, “Key Token Formats” for the format of an internal key token.

If the first byte of the key identifier is X'02', the key identifier is interpreted as an **external key token**. By using the external key token, you can exchange keys between systems. It contains a key that is encrypted under a key-encrypting key.

An external key token contains an encrypted key and control information to allow compatible cryptographic systems to:

- Have a standard method of exchanging keys
- Control the use of keys through the control vector
- Merge the key with other information needed to use the key

An application obtains the external key token by using one of the callable services such as these listed. They are described in detail in Chapter 5, “Managing Symmetric Cryptographic Keys.”

- Key generate
- Key export
- Data key export
- Symmetric key export

For debugging information, see Appendix B, “Key Token Formats” for the format of an external key token.

If the first byte of the key identifier is X'00', the key identifier is interpreted as a **null key token**. Use the null key token to import a key from a system that cannot produce external key tokens. That is, if you have an 8- to 16-byte key that has been encrypted under an importer key, but is not imbedded within a token, place the encrypted key in a null key token and then invoke the key import callable service to get the key in operational form.

For debugging information, see Appendix B, “Key Token Formats” for the format of a null key token.

## Key Wrapping

ICSF supports two methods of wrapping the key value in a fixed-length symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant.

The key value in a symmetric key token may be wrapped in two ways. The original method has been used by ICSF since it was first released. The key value in DES key tokens are encrypted using triple DES encryption and key parts are encrypted separately. The key value in AES tokens are encrypted using AES encryption and cipher block chaining mode.

The enhanced method of key wrapping, introduced in HCR7780, is ANSI X9.24 compliant. The key value of all keys are bundled with other token data and encrypted using triple DES or AES encryption and cipher block chaining mode. The enhanced method requires a z196 or IBM zEnterprise EC12 with a CEX3C or CEX4C.

Your installation's system programmer can, while customizing installation options data set as described in the *z/OS Cryptographic Services ICSF System Programmer's Guide*, use the DEFAULTWRAP parameter to specify the default key wrapping for symmetric keys. Application programs can override this default method using the WRAP-ENH (use enhanced method) and WRAP-ECB (use original ECB key-wrapping method) rule array keywords.

**Note:** Variable-length tokens are wrapped using the AESKW wrapping method defined in ANSI X9.102 and are not affected by the DEFAULTWRAP setting.

## Control Vector for DES Keys

For DES keys, a unique control vector exists for each type of key the master key enciphers. The cryptographic feature exclusive ORs the master key with the control vector associated with the type of key the master key will encipher. The control vector ensures that an operational key is only used in cryptographic functions for which it is intended. For example, the control vector for an input PIN-encrypting key ensures that such a key can be used only in the Encrypted PIN translate and Encrypted PIN verify functions.

## Types of Keys

The cryptographic keys are grouped into these categories based on the functions they perform.

- **DES master key.** The DES master key is a double-length (128 bits) or triple-length (192 bits) key used only to encrypt other keys. The ICSF

administrator installs and changes the DES master key (see *z/OS Cryptographic Services ICSF Administrator's Guide* for details). The administrator does this by using the Master Key Entry panels or the optional Trusted Key Entry (TKE) workstation.

The master key always remains in a secure area in the cryptographic facility.

It is used only to encipher and decipher keys. Other keys also encipher and decipher keys and are mostly used to protect cryptographic keys you transmit on external links. These keys, while on the system, are also encrypted under the master key.

- **AES master key.** The AES master key is a 32-byte (256 bits) key used only to encrypt other keys. The ICSF administrator installs and changes the AES master key (see *z/OS Cryptographic Services ICSF Administrator's Guide* for details). The administrator does this by using the Master Key Entry panels or the optional Trusted Key Entry (TKE) workstation (TKE V5.3).

The master key always remains in a secure area in the cryptographic facility.

It is used only to encipher and decipher keys. Other keys also encipher and decipher keys and are mostly used to protect cryptographic keys you transmit on external links. These keys, while on the system, are also encrypted under the master key.

- **AES Data-encrypting keys.** The AES data-encrypting keys are 128-, 192- and 256-bits keys that protect data privacy. If you intend to use a data-encrypting key for an extended period, you can store it in the CKDS so that it will be reenciphered if the master key is changed.
- **AES Cipher keys.** The AES cipher keys are 128-, 192- and 256-bit keys that protect data privacy. If you intend to use a cipher key for an extended period, you can store it in the CKDS so that it will be reenciphered if the master key is changed.
- **DES Data-encrypting keys.** The DES data-encrypting keys are single-length (64-bit), double-length (128-bit), or triple-length (192-bit) keys that protect data privacy. Single-length data-encrypting keys can also be used to encode and decode data and authenticate data sent in messages. If you intend to use a data-encrypting key for an extended period, you can store it in the CKDS so that it will be reenciphered if the master key is changed.

You can use single-length data-encrypting keys in the encipher, decipher, encode, and decode callable services to manage data and also in the MAC generation and MAC verification callable services. Double-length and triple-length data-encrypting keys can be used in the encipher and decipher callable services for more secure data privacy. DATAC is also a double-length data encrypting key.

Single-length data-encrypting keys can be exported and imported using the ANSI X9.17 key management callable services.

- **DES Data-translation keys.** The data-translation keys are single-length (64 bits) keys used for the ciphertext translate callable service as either the input or the output data-transport key.

**Restriction:** Only single-length data-translation keys are supported on the IBM @server zSeries 900.

- **AES Cipher text translation keys.** The cipher-translation keys are 128-, 192- or 256-bit keys used for the cipher text translate2 callable service as either the input or the output cipher-transport key.
- **DES Cipher text translation keys.** The cipher-translation keys are double-length keys used for the cipher text translate2 callable service as either the input or the output cipher-transport key.

- **CIPHER keys.** These consist of CIPHER, ENCIPHER and DECIPHER keys. They are single and double length keys for enciphering and deciphering data.

**Note:** Double-length CIPHER, ENCIPHER and DECIPHER keys are not supported on the IBM @server zSeries 900 and require a PCIXCC or CCA Crypto Express Coprocessor.

- **HMAC keys.** HMAC keys are variable-length (80 - 2048 bits) keys used to generate and verify MACs using the key-hash MAC algorithm.
- **MAC keys.** The MAC keys are single- and double-length (64 and 128 bits) keys used for the callable services that generate and verify MACs.

With a PCIXCC or CCA Crypto Express coprocessor, MAC and MACVER can be single or double length keys.

- **PIN keys.** The personal identification number (PIN) is a basis for verifying the identity of a customer across financial industry networks. PIN keys are used in cryptographic functions to generate, translate, and verify PINs, and protect PIN blocks. They are all double-length (128 bits) keys. PIN keys are used in the Clear PIN generate, Encrypted PIN verify, and Encrypted PIN translate callable services.

For installations that do not support double-length 128-bit keys, effective single-length keys are provided. For a single-length key, the left key half of the key equals the right key half.

“Managing Personal Authentication” on page 59 gives an overview of the PIN algorithms you need to know to write your own application programs.

- **AES Transport keys (or key-encrypting keys).** Transport keys are also known as key-encrypting keys. They are used to protect AES and HMAC keys when you distribute them from one system to another.

There are two types of AES transport keys:

- Exporter key-encrypting key protects keys of any type that are sent from your system to another system. The exporter key at the originator is the same key as the importer key of the receiver.
- Importer key-encrypting key protects keys of any type that are sent from another system to your system. It also protects keys that you store externally in a file that you can import to your system at another time. The importer key at the receiver is the same key as the exporter key at the originator.

**Note:** A key-encrypting key should be as strong or stronger than the key it is wrapping.

- **DES Transport keys (or key-encrypting keys).** Transport keys are also known as key-encrypting keys. They are double-length (128 bits) DES keys used to protect keys when you distribute them from one system to another.

There are several types of transport keys:

- *Exporter or OKEYXLAT key-encrypting key* protects keys of any type that are sent from your system to another system. The exporter key at the originator is the same key as the importer key of the receiver.
- *Importer or IKEYXLAT key-encrypting key* protects keys of any type that are sent from another system to your system. It also protects keys that you store externally in a file that you can import to your system at another time. The importer key at the receiver is the same key as the exporter key at the originator.
- *NOCV Importers and Exporters* are key-encrypting keys used to transport keys with systems that do not recognize key-encrypting key variants. There are some requirements and restrictions for the use of NOCV key-encrypting keys:



- On CCF systems, installation of NOCV enablement keys on the CKDS is required.
- On Cryptographic Coprocessor systems, use of NOCV IMPORTERs and EXPORTERs is controlled by access control points.
- Only programs in system or supervisor state can use the NOCV key-encrypting key in the form of tokens in callable services. Any problem program may use NOCV key-encrypting key with labelnames from the CKDS.
- NOCV key-encrypting key on the CKDS should be protected by RACF.
- NOCV key-encrypting key can be used to encrypt single or double length keys with standard CVs for key types DATA, DATAC, DATAM, DATAMV, DATAXLAT, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, single-length MAC, single-length MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER .
- With PCIXCCs or CCA Crypto Express coprocessors, NOCV key-encrypting keys can be used with triple length DATA keys. Since DATA keys have 0 CVs, processing will be the same as if the key-encrypting keys are standard key-encrypting keys (not the NOCV key-encrypting key).

**Note:** Transport keys replace local, remote, and cross keys used by PCF.

You use key-encrypting keys to protect keys that are transported using any of these services: data key export, key export, key import, clear key import, multiple clear key import, secure key import, multiple secure key import, key generate, and key translate.

For installations that do not support double-length key-encrypting keys, effective single-length keys are provided. For an effective single-length key, the clear key value of the left key half equals the clear key value of the right key half.

- **ANSI X9.17 key-encrypting keys.** These bidirectional key-encrypting keys are used exclusively in ANSI X9.17 key management. They are either single-length (64 bits) or double-length (128 bits) keys used to protect keys when you distribute them from one system to another according to the ANSI X9.17 protocol.

**Note:** ANSI X9.17 keys are only supported on the IBM @server zSeries 900.

- **Key-Generating Keys.** Key-generating keys are double-length keys used to derive unique-key-per-transaction keys.

## Other Considerations

These are considerations for keys held in the cryptographic key data set (CKDS) or by applications.

- ICSF ensures that keys held in the CKDS are reenciphered during the master key change. Keys with a long life span (more than one master key change) should be stored in the CKDS.
- Keys enciphered under the host DES master key and held by applications are automatically reenciphered under a new master key as they are used. Keys with a short life span (for example, VTAM SLE data keys) do not need to be stored in the CKDS. However, if you have keys with a long life span and you do not store them in the CKDS, they should be enciphered under the importer key-encrypting key. The importer key-encrypting key itself should be stored in the CKDS.

Table 3 on page 23 describes the key types.



You can build, generate, import, or export key types DECIPHER, ENCIPHER, CIPHER, CVARDEC, and CVARPINE on a CCF system, but they are not usable on CCF systems. They will be usable by ICSF if running on z990/z890 and later systems with a PCIXCC or CCA Crypto Express coprocessor.

*Table 3. Descriptions of Key Types*

Key Type	Meaning
AESDATA	Data encrypting key. Use the AES 128-, 192- or 256-bit key to encipher and decipher data.
AESTOKEN	May contain an AES key.
AKEK	Single-length or double-length, bidirectional key-encrypting key used for the ANSI X9.17 key management callable services. AKEK keys are only supported on the IBM @server zSeries 900.
CIPHER	<ul style="list-style-type: none"> <li><b>DES:</b> This single or double-length key is used to encrypt or decrypt data. It can be used in the Encipher and Decipher callable services.</li> <li><b>z900 only:</b> This is a single-length key and cannot be used in the Encipher and Decipher services.</li> <li><b>AES:</b> This 128-, 192- or 256-bit key is used to encrypt or decrypt data. It can be used in the Symmetric Algorithm Decipher and Symmetric Algorithm Encipher callable services.</li> </ul> <p>This 128-, 192, or 256-bit key is used to translate cipher text from one key to another. It can be used in the Cipher Text Translate2 service.</p>
CIPHERXI	DES input cipher text translation key. Use this double-length key to reencipher text from one data-encrypting key to another.
CIPHERXL	DES cipher text translation key. Use this double-length key to reencipher text from one data-encrypting key to another.
CIPHERXO	DES output cipher text translation key. Use this double-length key to reencipher text from one data-encrypting key to another.
CLRAES	Data encrypting key. The key value is not encrypted. Use this AES 128-, 192- or 256-bit key to encipher and decipher data.
CLRDES	Data encrypting key. The key value is not encrypted. Use this DES single-length, double-length, or triple-length key to encipher and decipher data.
CVARDEC	The TSS Cryptographic variable decipher verb uses a CVARDEC key to decrypt plaintext by using the Cipher Block Chaining (CBC) method. This is a single-length key.
CVARENC	Cryptographic variable encipher service uses a CVARENC key to encrypt plaintext by using the Cipher Block Chaining (CBC) method. This is a single-length key.
CVARPINE	Used to encrypt a PIN value for decryption in a PIN-printing application. This is a single-length key.
CVARXCVL	Used to encrypt special control values in DES key management. This is a single-length key.
CVARXCVR	Used to encrypt special control values in DES key management. This is a single-length key.
DATA	Data encrypting key. Use this DES single-length, double-length, or triple-length key to encipher and decipher data. Use the AES 128-, 192- or 256-bit key to encipher and decipher data.
DATAAC	Used to specify a DATA-class key that will perform in the Encipher and Decipher callable services, but not in the MAC Generate or MAC Verify callable services. This is a double-length key. Only available with a PCIXCC or CCA Crypto Express coprocessor.

Table 3. Descriptions of Key Types (continued)

Key Type	Meaning
DATAM	Double-length MAC generation key. Used to generate a message authentication code.
DATAMV	Double-length MAC verification key. Used to verify a message authentication code.
DATAXLAT	Data translation key. Use this single-length key to reencipher text from one DATA key to another. DATAXLAT keys are only supported on the IBM @server zSeries 900.
DECIPHER	This single or double-length DES key is used to decrypt data. It can be used in the Decipher callable service.  <b>z900 only:</b> This is a single-length key and cannot be used in the Decipher service.
DKYGENKY	Used to generate a diversified key based on the key-generating key. This is a double-length key.
ENCIPHER	This single or double-length DES key is used to encrypt data. It can be used in the Encipher callable service.  <b>z900 only:</b> This is a single-length key and cannot be used in the Encipher service.
EXPORTER	Exporter key-encrypting key. Use this double-length DES key or 128-, 192-, or 256-bit AES key to convert a key from the operational form into exportable form.
IKEYXLAT	Used to decrypt an input key in the Key Translate callable service. This is a double-length key.
IMPORTER	Importer key-encrypting key. Use this double-length DES key or 128-, 192- or 256-bit AES key to convert a key from importable form into operational form.
IMP-PKA	Double-length limited-authority importer key used to encrypt PKA private key values in PKA external tokens.
IPINENC	Double-length input PIN-encrypting key. PIN blocks received from other nodes or automatic teller machine (ATM) terminals are encrypted under this type of key. These encrypted PIN blocks are the input to the Encrypted PIN translate, Encrypted PIN verify, and Clear PIN Generate Alternate services. If an encrypted PIN block is contained in the output of the SET Block Decompose service, it may be encrypted by an IPINENC key.
KEYGENKY	Used to generate a key based on the key-generating key. This is a double-length key.
MAC	Single, double-length, or variable-length MAC generation key. Use this key to generate a message authentication code.  <b>z900 only:</b> This is a single-length key.
MACVER	Single, double-length, or variable-length MAC verification key. Use this key to verify a message authentication code.  <b>z900 only:</b> This is a single-length key.
OKEYXLAT	Used to encrypt an output key in the Key Translate callable service. This is a double-length key.

Table 3. Descriptions of Key Types (continued)

Key Type	Meaning
OPINENC	Output PIN-encrypting key. Use this double-length output key to translate PINs. The output PIN blocks from the Encrypted PIN translate, Encrypted PIN generate, and Clear PIN generate alternate callable services are encrypted under this type of key. If an encrypted PIN block is contained in the output of the SET Block Decompose service, it may be encrypted by an OPINENC key.
PINGEN	PIN generation key. Use this double-length key to generate PINs.
PINVER	PIN verification key. Use this double-length key to verify PINs.
SECMSG	Used to encrypt PINs or keys in a secure message. This is a double-length key.
TOKEN	A key token that may contain a key.

## Clear Keys

A clear key is the base value of a key, and is not encrypted under another key. Encrypted keys are keys whose base value has been encrypted under another key.

There are four callable services you can use to convert a clear key to an encrypted key:

- To convert a clear key to an encrypted *data* key in operational form, use either the Clear Key Import callable service or the Multiple Clear Key Import callable service.
- To convert a clear key to an encrypted key of any type, in operational or importable form, use either the Secure Key Import callable service or the Multiple Secure Key Import callable service.

**Note:** The Secure Key Import and Multiple Secure Key Import callable services can only execute in special secure mode.

Clear key DATA tokens can be stored in the CKDS. These tokens can only be used by symmetric key decipher and symmetric key encipher callable services for the DES and AES algorithms.

## Key Strength and Wrapping of Key

Key strength is measured as “bits of security” as described in the documentation of NIST and other organizations. Each individual key will have its “bits of security” computed, then the different key types (AES, DES, ECC, RSA, HMAC) can then have their relative strengths compared on a single scale. When the raw value of a particular key falls between discreet values of the NIST table, the lower value from the table will be used as the “bits of security”.

The following tables show some examples of the restrictions due to key strength.

When wrapping an HMAC key with an AES key-encrypting key, the strength of the AES key-encrypting key depends on the attributes of the HMAC key.

Table 4. AES EXPORTER strength required for exporting an HMAC key under an AES EXPORTER

Key-usage field 2 in the HMAC key	Minimum strength of AES EXPORTER to adequately protect the HMAC key
SHA-256, SHA-384, SHA-512	256 bits
SHA-224	192 bits
SHA-1	128 bits

Table 5. Minimum RSA modulus length to adequately protect an AES key

Bit length of AES key to be exported	Minimum strength of RSA wrapping key to adequately protect the AES key
128	3072
192	7860
256	15360

## Key Strength and Key Wrapping Access Control Points

In order to comply with cryptographic standards, including ANSI X9.24 Part 1 and PCI-HSM, ICSF provides a way to ensure that a key is not wrapped with a key weaker than itself. ICSF provides a set of access control points in the ICSF role to control the wrapping of keys. ICSF administrators can use these access control points to meet an installation's individual requirements.

There are new and existing access control points that control the wrapping of keys by master and key-encrypting keys. These access control points will either prohibit the wrapping of a key by a key of weaker strength or will return a warning (return code 0, reason code non-zero) when a key is wrapped by a weaker key. All of these ACPs are disabled by default in the ICSF role.

The processing of callable services will be affected by these access control points. Here is a description of the access control points, the wrapping they control, and the effect on services. These access control points apply to symmetric and asymmetric keys.

When the **Prohibit weak wrapping - Transport keys** access control point is enabled, any service that attempts to wrap a key with a weaker transport key will fail.

When the **Prohibit weak wrapping - Master keys** access control point is enabled, any service that wraps a key under a master key will fail if the master key is weaker than the key being wrapped.

When the **Warn when weak wrap - Transport keys** access control point is enabled, any service that attempts to wrap a key with a weaker transport key will succeed with a warning reason code.

When the **Warn when weak wrap - Master keys** access control point is enabled, any service that attempts to wrap a key with a weaker master key will succeed with a warning reason code.

24-byte DATA keys with a zero control vector can be wrapped with a 16-byte key, the DES master key, or a key-encrypting key, which violates the wrapping requirements. The **Prohibit weak wrapping - Transport keys** and **Prohibit weak**

**wrapping – Master keys** access control points do not cause services to fail for this case. The **Disallow 24-byte DATA wrapped with 16-byte Key** access control point does control this wrapping. When enabled, services will fail. The **Warn when weak wrap – Transport keys** and **Warn when weak wrap – Master keys** access control points will cause the warning to be returned when the access control points are enabled.

When the **RKX/TBC – Disallow triple-length MAC key** access control point is enabled, CSNDRKX will fail to import a triple-length MAC key under a double-length key-encrypting key. CSNBTBC will not wrap a triple-length MAC key under a double-length key-encrypting key. The **Prohibit weak wrapping – Transport keys** and **Prohibit weak wrapping – Master keys** access control points do not cause services to fail for this case. The **Warn when weak wrap – Transport keys** and **Warn when weak wrap – Master keys** access control points will cause the warning to be returned when the ACPs are enabled.

If the **Prohibit Weak Wrap** access control point is enabled, RSA private keys may not be wrapped using a weaker DES key-encrypting key. Enabling the **Allow weak DES wrap of RSA private key** access control points will override this restriction.

## DES Master Key

Since ICSF only allows a 16-byte DES master key to be loaded, ICSF can't be compliant for key strength for 24-byte operational keys wrapped by the DES master key. Starting with ICSF release HCR77A0, a 24-byte master key can be loaded. Only cryptographic coprocessors with the October, 2012 licensed internal code support this key length. The **DES master key – 24-byte key** access control point must be enabled in the ICSF role. See the *z/OS Cryptographic Services ICSF Administrator's Guide* for more details.

---

## Generating and Managing Symmetric Keys

Using ICSF, you can generate keys using either the *key generator utility program* or the *key generate callable service*. The dynamic CKDS update callable services allow applications to directly manipulate the CKDS. ICSF provides callable services that support DES and AES key management as defined by the IBM Common Cryptographic Architecture (CCA) and by the ANSI X9.17 standard.

The next few topics describe the key generating and management options ICSF provides.

### Key Generator Utility Program

The key generator utility program generates data, data-translation, MAC, PIN, and key-encrypting keys, and enciphers each type of key under a specific master key variant. When the KGUP generates a key, it stores it in the cryptographic key data set (CKDS).

**Note:** If you specify CLEAR, KGUP uses the random number generate and secure key import callable services rather than the key generate service.

You can access KGUP using ICSF panels. The KGUP path of these panels helps you create the JCL control statements to control the key generator utility program. When you want to generate a key, you can enter the ADD control statement and information, such as the key type on the panels. For a detailed description of the key generator utility program and how to use it to generate keys, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

## Common Cryptographic Architecture DES Key Management Services

ICSF provides callable services that support CCA key management for DES keys.

### **Clear Key Import Callable Service (CSNBCKI and CSNECKI)**

This service imports a clear DATA key that is used to encipher or decipher data. It accepts a clear key and enciphers the key under the host master key, returning an encrypted DATA key in operational form in an internal key token.

### **Control Vector Generate Callable Service (CSNBCVG and CSNECVG)**

The control vector generate callable service builds a control vector from keywords specified by the *key\_type* and *rule\_array* parameters.

### **Control Vector Translate Callable Service (CSNBCVT and CSNECVT)**

The control vector translate callable service changes the control vector used to encipher an external key. Use of this service requires the optional PCI Cryptographic Coprocessor.

### **Cryptographic Variable Encipher Callable Service (CSNBCVE and CSNECVE)**

The cryptographic variable encipher callable service uses a CVARENC key to encrypt plaintext by using the Cipher Block Chaining (CBC) method. You can use this service to prepare a mask array for the control vector translate service. The plaintext must be a multiple of eight bytes in length.

### **Data Key Export Callable Service (CSNBCKX and CSNEDCKX)**

This service reenciphers a DATA key from encryption under the master key to encryption under an exporter key-encrypting key, making it suitable for export to another system.

### **Data Key Import Callable Service (CSNBCKM and CSNEDCKM)**

This service imports an encrypted source DES DATA key and creates or updates a target internal key token with the master key enciphered source key.

### **Diversified Key Generate Callable Service (CSNBCKG and CSNEDCKG)**

The diversified key generate service generates a key based on the key-generating key, the processing method, and the parameter supplied. The control vector of the key-generating key also determines the type of target key that can be generated.

### **Key Export Callable Service (CSNBKEX and CSNEKEX)**

This service reenciphers any type of key (except an AKEK or IMP-PKA key) from encryption under a master key variant to encryption under the same variant of an exporter key-encrypting key, making it suitable for export to another system.

### **Key Generate Callable Service (CSNBKGN and CSNEKGN)**

The key generate callable service generates data, data-translation, MAC, PIN, and key-encrypting keys. It generates a single key or a pair of keys. Unlike the key generator utility program, the key generate service does not store the keys in the CKDS where they can be saved and maintained. The key generate callable service returns the key to the application program that called it. The application program can then use a dynamic CKDS update service to store the key in the CKDS.



When you call the key generate callable service, include parameters specifying information about the key you want generated. Because the form of the key restricts its use, you need to choose the form you want the generated key to have. You can use the *key\_form* parameter to specify the form. The possible forms are:

- **Operational**, if the key is used for cryptographic operations on the local system. Operational keys are protected by master key variants and can be stored in the CKDS or held by applications in internal key tokens.
- **Importable**, if the key is stored with a file or sent to another system. Importable keys are protected by importer key-encrypting keys.
- **Exportable**, if the key is transported or exported to another system and imported there for use. Exportable keys are protected by exporter key-encrypting keys and cannot be used by ICSF callable service.

Importable and exportable keys are contained in external key tokens. For more information on key tokens, refer to “Key Token” on page 17.

### **Key Import Callable Service (CSNBKIM and CSNEKIM)**

This service reenciphers a key (except an AKEK) from encryption under an importer key-encrypting key to encryption under the master key. The reenciphered key is in the operational form.

### **Key Part Import Callable Service (CSNBKPI and CSNEKPI)**

This service combines clear key parts of any key type and returns the combined key value either in an internal token or as an update to the CKDS.

### **Key Test Callable Service (CSNBKYT, CSNEKYT, CSNBKYTX, and CSNEKYTX)**

This service generates or verifies a secure cryptographic verification pattern for keys. A parameter indicates the action you want to perform.

The key to test can be in the clear or encrypted under a master key. The key test extended callable service works on keys encrypted under a KEK.

For generating a verification pattern, the service creates and returns a random number with the verification pattern. For verifying a pattern, you supply the random number from the call to the service that generated the pattern.

### **Key Token Build Callable Service (CSNBKTB and CSNEKTB)**

The key token build callable service is a utility you can use to create skeleton key tokens for AKEKs as input to the key generate or key part import callable service. You can also use this service to build CCA key tokens for all key types ICSF supports. You can also use this service to build CCA key tokens for all key types ICSF supports.

### **Key Translate Callable Service (CSNBKTR and CSNEKTR)**

This service uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment.

### **Key Translate2 Callable Service (CSNBKTR2 and CSNEKTR2)**

This service uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment.

### **Multiple Clear Key Import Callable Service (CSNBCKM and CSNECKM)**

This service imports a single-length, double-length, or triple-length clear DATA key that is used to encipher or decipher data. It accepts a clear key and enciphers the key under the host master key, returning an encrypted DATA key in operational form in an internal key token.

### **Multiple Secure Key Import Callable Service (CSNBSKM and CSNESKM)**

This service enciphers a single-length, double-length, or triple-length clear key under the host master key or under an importer key-encrypting key. The clear key can then be imported as any of the possible key types. Triple-length keys can only be imported as DATA keys. This service can be used only when ICSF is in special secure mode and does not allow the import of an AKEK.

### **Prohibit Export Callable Service (CSNBPEX and CSNEPEX)**

This service modifies an operational key so that it cannot be exported. This callable service does not support NOCV key-encrypting keys, DATA, MAC, or MACVER keys with standard control vectors (for example, control vectors supported by the Cryptographic Coprocessor Feature).

### **Prohibit Export Extended Callable Service (CSNBPEXX and CSNEPEXX)**

This service updates the control vector in the external token of a key in exportable form so that the receiver node can import the key but not export it. When the key import callable service imports such a token, it marks the token as non-exportable. The key export callable service does not allow export of this token.

### **Random Number Generate Callable Service (CSNBRNG, CSNERNG, CSNBRNGL, and CSNERNGL)**

The random number generate callable service creates a random number value to use in generating a key. The callable service uses cryptographic hardware to generate a random number for use in encryption.

### **Remote Key Export Callable Service (CSNDRKX and CSNFRKX)**

The remote key export callable service uses the trusted block to generate or export DES keys for local use and for distribution to an ATM or other remote device.

### **Restrict Key Attribute Callable Service (CSNBRKA and CSNERKA)**

This service modifies an AES operational key so that it cannot be exported.

### **Secure Key Import Callable Service (CSNBSKI and CSNESKI)**

This service enciphers a clear key under the host master key or under an importer key-encrypting key. The clear key can then be imported as any of the possible key types. This service can be used only when ICSF is in special secure mode and does not allow the import of an AKEK.

**Note:** The PKA encrypt, PKA decrypt, symmetric key generate, symmetric key import, and symmetric key export callable services provide a way of distributing DES DATA keys protected under a PKA key. See Chapter 3, "Introducing PKA Cryptography and Using PKA Callable Services," on page 83 for additional information.



### **Symmetric Key Export Callable Service (CSNDSYX and CSNFSYX)**

This service transfers an application-supplied symmetric key (a DATA key) from encryption under the DES host master key to encryption under an application-supplied RSA public key. (There are two types of PKA public key tokens: RSA and DSS. This callable service can use only the RSA type.) The application-supplied DATA key must be an ICSF DES internal key token or the label of such a token in the CKDS. The symmetric key import callable service can import the PKA-encrypted form at the receiving node.

### **Symmetric Key Generate Callable Service (CSNDSYG and CSNFSYG)**

This service generates a symmetric key (that is, a DATA key) and returns it encrypted using DES and encrypted under an RSA public key token. (There are two types of PKA public key tokens: RSA and DSS. This callable service can use only the RSA type.)

The DES-encrypted key can be an internal token encrypted under a host DES master key, or an external form encrypted under a KEK. (You can use the symmetric key import callable service to import the PKA-encrypted form.)

### **Symmetric Key Import Callable Service (CSNDSYI and CSNFSYI)**

This service imports a symmetric (DES) DATA key enciphered under an RSA public key. (There are two types of PKA private key tokens: RSA and DSS. This callable service can use only the RSA type.) This service returns the key in operational form, enciphered under the DES master key.

### **Transform CDMF Key Callable Service (CSNBTCK and CSNETCK)**

**Restriction:** This service is only available on the IBM @server zSeries 900.

It changes a CDMF DATA key in an internal or external token to a transformed shortened DES key. It ignores the input internal DES token markings and marks the output internal token for use in the DES. You need to use this service only if you have a CDMF or DES-CDMF system that needs to send CDMF-encrypted data to a DES-only system. The CDMF or DES-CDMF system must generate the key, shorten it, and pass it to the DES-only system.

If the input DATA key is in an external token, the operational KEK must be marked as DES or SYS-ENC. The service fails for an external DATA key encrypted under a KEK that is marked as CDMF.

### **Trusted Block Create Callable Service (CSNDTBC and CSNFTBC)**

This service creates and activates a trusted block under two step process.

### **Unique Key Derive Callable Service (CSFBUKD and CSFEUKD)**

Unique Key Derive will perform the key derivation process as defined in ANSI X9.24 Part 1, Using a Base Derivation Key and Derivation Data as inputs.

### **User Derived Key Callable Service (CSFUDK and CSFUDK6)**

**Restriction:** This service is only available on the IBM @server zSeries 900.

This service generates a single-length or double-length MAC key, or updates an existing user-derived key. A single-length MAC key can be used to compute a Message Authentication Code (MAC) following the ANSI X9.9, ANSI X9.19, or the Europay, MasterCard, Visa (EMV) Specification MAC processing rules. A

double-length MAC key can be used to compute a MAC following the ANSI X9.19 optional double MAC processing rule or the EMV rules.

## **Common Cryptographic Architecture AES Key Management Services**

ICSF provides callable services that support CCA key management for AES keys.

### **Key Generate Callable Service (CSNBKGN and CSNEKGN)**

The key generate callable service generates AES data keys. It generates a single operational key. Unlike the key generator utility program, the key generate service does not store the keys in the CKDS where they can be saved and maintained. The key generate callable service returns the key to the application program that called it. The application program can then use a dynamic CKDS update service to store the key in the CKDS.

### **Key Generate2 Callable Service (CSNBKGN2 and CSNEKGN2)**

The service generates AES keys. It generates one operational key or an operational key pair. The key generate callable service returns the key to the application program that called it. The application program can then use a dynamic CKDS update service to store the key in the CKDS.

### **Key Part Import2 Callable Service (CSNBKPI2 and CSNEKPI2)**

This service combines clear key parts of any AES key type and returns the combined key value either in an internal token or as an update to the CKDS.

### **Key Test2 Callable Service (CSNBKYT2 and CSNEKYT2)**

This service generates or verifies a secure cryptographic verification pattern for AES keys. A parameter indicates the action you want to perform.

### **Key Token Build Callable Service (CSNBKTB and CSNEKTB)**

The key token build callable service is a utility you can use to create clear AES key tokens, secure AES key tokens and skeleton secure AES key tokens for use with other callable services. You can also use this service to build CCA key tokens for all key types ICSF supports. You can also use this service to build CCA key tokens for all key types ICSF supports.

### **Multiple Clear Key Import Callable Service (CSNBCKM and CSNECKM)**

This service imports a 128-, 192- or 256-bit clear DATA key that is used to encipher or decipher data. It accepts a clear key and enciphers the key under the host master key, returning an encrypted DATA key in operational form in an internal key token.

### **Multiple Secure Key Import Callable Service (CSNBSKM and CSNESKM)**

This service enciphers 128-, 192- or 256-bit clear DATA key under the host master key. This service can be used only when ICSF is in special secure mode.

### **Restrict Key Attribute Callable Service (CSNBRKA and CSNERKA)**

This service modifies an AES operational key so that it cannot be exported.

### **Secure Key Import2 Callable Service (CSNBSKI2 and CSNESKI2)**

This service enciphers a variable length clear AES key under the host master key. This service can be used only when ICSF is in special secure mode.

### **Symmetric Key Export Callable Service (CSNDSYX and CSNFSYX)**

Use the symmetric key export callable service to transfer an application-supplied AES DATA key from encryption under a master key to encryption under an application-supplied RSA public key or AES EXPORTER key. The application-supplied key must be an ICSF AES internal key token or the label of such a token in the CKDS. The Symmetric Key Import or Symmetric Key Import2 callable services can import the key encrypted under the RSA public key or AES EXPORTER at the receiving node.

### **Symmetric Key Generate Callable Service (CSNDSYG and CSNFSYG)**

This service generates a symmetric DATA key and returns it encrypted under the host AES master key and encrypted under an RSA public key token. (There are two types of PKA public key tokens: RSA and DSS. This callable service can use only the RSA type.)

The AES-encrypted key can only be an internal token encrypted under a host AES master key. You can use the symmetric key import callable service to import the PKA-encrypted form.

### **Symmetric Key Import Callable Service (CSNDSYI and CSNFSYI)**

This service imports a symmetric (AES) DATA key enciphered under an RSA public key. (There are two types of PKA private key tokens: RSA and DSS. This callable service can use only the RSA type.) This service returns the key in operational form, enciphered under the AES master key.

### **Symmetric Key Import2 Callable Service (CSNDSYI2 and CSNFSYI2)**

This service imports an AES key enciphered under an RSA public key. (There are two types of PKA private key tokens: RSA and DSS. This callable service can use only the RSA type.) This service returns the key in operational form, enciphered under the AES master key.

## **Common Cryptographic Architecture HMAC Key Management Services**

ICSF provides callable services that support CCA key management for HMAC keys. HMAC keys are stored in the cryptographic key data set (CKDS).

### **Key Generate2 callable service (CSNBKGN2 and CSNEKGN2)**

The service generates HMAC keys. It generates operational key or operational key pair. The key generate callable service returns the key to the application program that called it. The application program can then use a dynamic CKDS update service to store the key in the CKDS.

### **Key Part Import2 callable service (CSNBKPI2 and CSNEKPI2)**

This service combines clear key parts of any HMAC key type and returns the combined key value either in an internal token or as an update to the CKDS.

### **Key Test2 callable service (CSNBKYT2 and CSNEKYT2)**

This service generates or verifies a secure cryptographic verification pattern for HMAC keys. A parameter indicates the action you want to perform.

### **Key Token Build2 callable service (CSNBKTB2 and CSNEKTB2)**

This service is a utility you can use to create skeleton HMAC key tokens for use with other callable services.

### **Restrict Key Attribute callable service (CSNBRKA and CSNERKA)**

This service modifies an HMAC operational key so that it cannot be exported.

### **Secure Key Import2 callable service (CSNBSKI2 and CSNESKI2)**

This service enciphers a variable length clear HMAC key under the host master key. This service can be used only when ICSF is in special secure mode.

### **Symmetric Key Export Callable Service (CSNDSYX and CSNFSYX)**

This service transfers an application-supplied symmetric key (HMAC key) from encryption under the AES host master key to encryption under an application-supplied RSA public key. (There are two types of PKA public key tokens: RSA and DSS. This callable service can use only the RSA type.) The application-supplied key must be an ICSF internal key token or the label of such a token in the CKDS. The symmetric key import callable service can import the PKA-encrypted form at the receiving node.

### **Symmetric Key Import2 Callable Service (CSNDSYI2 and CSNFSYI2)**

This service imports an HMAC key enciphered under an RSA public key. (There are two types of PKA private key tokens: RSA and DSS. This callable service can use only the RSA type.) This service returns the key in operational form, enciphered under the AES master key.

## **ECC Diffie-Hellman Key Agreement Models**

### **Token Agreement Scheme**

The caller must have both the required key tokens and both Parties identifiers including a randomly generated nonce. Combine the exchanged nonce and Party Info into the party identifier. (Both parties must combine this information in the same format.) Then call the ECC Diffie-Hellman callable service. Specify a skeleton token or the label of a skeleton token as the output key identifier as a container for the computed symmetric key material. Note, both parties must specify the same key type in their skeleton key tokens.

- Specify rule array keyword DERIV01 to denote the Static Unified Model key agreement scheme.
- Specify an ECC token as the private key identifier containing this party's ECC public-private key pair.
- Optionally specify a private KEK key identifier, if the key pair is in an external key token.
- Specify an ECC token as the public key identifier containing other party's ECC public key part.
- Specify a skeleton token as the output key identifier as a container for the computed symmetric key material.
- Optionally specify an output KEK key identifier, if the output key is to be in an external key token.
- Specify the combined party info (including nonce) as the party identifier.
- Specify the desired size of the key to be derived (in bits) as the key bit length.

### **Obtaining the Raw "Z" value**

To use a key agreement scheme that differs from the above, one may obtain the raw shared secret "Z" and skip the key derivation step. The caller must then derive the final key material using a method of their choice. Do not specify any party info.

- Specify rule array keyword "PASSTHRU" to denote no key agreement scheme.
- Specify an ECC token as the private key identifier containing this party's ECC public-private key pair.
- Optionally specify a private KEK key identifier, if the key pair is in an external key token.
- Specify an ECC token as the public key identifier containing other party's ECC public key part.
- The output key identifier be populated with the resulting shared secret material.

## Improved remote key distribution

**Note:** This improved remote key distribute support is only available on the z9 EC, z9 BC, z10 EC and z10 BC servers.

New methods have been added for securely transferring symmetric encryption keys to remote devices, such as Automated Teller Machines (ATMs), PIN-entry devices, and point of sale terminals. These methods can also be used to transfer symmetric keys to another cryptographic system of any type, such as a different kind of Hardware Security Module (HSM) in an IBM or non-IBM computer server. This change is especially important to banks, since it replaces expensive human operations with network transactions that can be processed quickly and inexpensively. This method supports a variety of requirements, fulfilling the new needs of the banking community while simultaneously making significant interoperability improvements to related cryptographic key-management functions.

For the purposes of this description, the ATM scenario will be used to illustrate operation of the new methods. Other uses of this method are also valuable.

### Remote Key Loading

Remote key loading refers to the process of installing symmetric encryption keys into a remotely located device from a central administrative site. This encompasses two phases of key distributions.

- Distribution of initial key encrypting keys (KEKs) to a newly installed device. A KEK is a type of symmetric encryption key that is used to encrypt other keys so they can be securely transmitted over unprotected paths.
- Distribution of operational keys or replacement KEKs, enciphered under a KEK currently installed in the device.

**Old remote key loading example:** Use an ATM as an example of the remote key loading process. A new ATM has none of the bank's keys installed when it is delivered from the manufacturer. The process of getting the first key securely loaded is difficult. This has typically been done by loading the first KEK into each ATM manually, in multiple cleartext key parts. Using dual control for key parts, two separate people must carry key part values to the ATM, then load each key part manually. Once inside the ATM, the key parts are combined to form the actual KEK. In this manner, neither of the two people has the entire key, protecting the key value from disclosure or misuse. This method is labor-intensive and error-prone, making it expensive for the banks.

**New remote key loading methods:** New remote key loading methods have been developed to overcome some of the shortcomings of the old manual key loading methods. These new methods define acceptable techniques using public key cryptography to load keys remotely. Using these new methods, banks will be able to load the initial KEKs without sending people to the remote device. This will

reduce labor costs, be more reliable, and be much less expensive to install and change keys. The new cryptographic features added provide new methods for the creation and use of the special key forms needed for remote key distribution of this type. In addition, they provide ways to solve long-standing barriers to secure key exchange with non-IBM cryptographic systems.

Once an ATM is in operation, the bank can install new keys as needed by sending them enciphered under a KEK installed previously. This is straightforward in concept, but the cryptographic architecture in ATMs is often different from that of the host system sending the keys, and it is difficult to export the keys in a form understood by the ATM. For example, cryptographic architectures often enforce key-usage restrictions in which a key is bound to data describing limitations on how it can be used - for encrypting data, for encrypting keys, for operating on message authentication codes (MACs), and so forth. The encoding of these restrictions and the method used to bind them to the key itself differs among cryptographic architectures, and it is often necessary to translate the format to that understood by the target device prior to a key being transmitted. It is difficult to do this without reducing security in the system; typically it is done by making it possible to arbitrarily change key-usage restrictions. The methods described here provide a mechanism through which the system owner can securely control these translations, preventing the majority of attacks that could be mounted by modifying usage restrictions.

A new data structure called a *trusted block* is defined to facilitate the remote key loading methods. The trusted block is the primary vehicle supporting these new methods.

### **Trusted block**

The trusted block is the central data structure to support all remote key loading functions. It provides great power and flexibility, but this means that it must be designed and used with care in order to have a secure system. This security is provided through several features of the design.

- A two step process is used to create a trusted block.
- The trusted block includes cryptographic protection that prevents any modification when it is created.
- A number of fields in the rules of a trusted block offer the ability to limit how the block is used, reducing the risk of it being used in unintended ways or with unintended keys.

The trusted block is the enabler which requires secure approval for its creation, then enables the export or generation of DES and TDES keys in a wide variety of forms as approved by the administrators who created the trusted block. For added security, the trusted blocks themselves can be created on a separate system, such as an xSeries server with an IBM 4764 Cryptographic Coprocessor, locked in a secure room. The trusted block can subsequently be imported into the zSeries server where they will be used to support applications.

There are two CCA callable services to manage and use trusted blocks: Trusted Block Create (CSNDTBC and CSNETBC) and Remote Key Export (CSNDRKX and CSNFRKX). The Trusted Block Create service creates a trusted block, and the Remote Key Export service uses a trusted block to generate or export DES keys according to the parameters in the trusted block. The trusted block consists of a header followed by several sections. Some elements are required, while others are optional.



Figure 1 shows the contents of a trusted block. The elements shown in the table give an overview of the structure and do not provide all of the details of a trusted block.

Structure version information	
Public key	Modulus
	Exponent
	Attributes
Trusted block protection information	MAC key
	MAC
	Flags
	MKVP
	Activation/Expiration dates
Public key name (optional)	Label
Rules	Rule 1
	Rule 2
	Rule 3
	...
	Rule N
Application defined data	Data defined and understood only by the application using the trusted block

Figure 1. Overview of trusted block contents

Here is a brief description of the elements that are depicted.

**Structure version information** - This identifies the version of the trusted block structure. It is included so that code can differentiate between this trusted block layout and others that may be developed in the future.

**Public key** - This contains the RSA public key and its attributes. For distribution of keys to a remote ATM, this will be the root certification key for the ATM vendor, and it will be used to verify the signature on public-key certificates for specific individual ATMs. In this case, the Trusted Block will also contain Rules that will be used to generate or export symmetric keys for the ATMs. It is also possible for the Trusted Block to be used simply as a trusted public key container, and in this case the Public Key in the block will be used in general-purpose cryptographic functions such as digital signature verification. The public key attributes contain information on key usage restrictions. This is used to securely control what operations will be permitted to use the public key. If desired, the public key can be restricted to use for only digital signature operations, or for only key management operations.

**Trusted block protection information** - This topic contains information that is used to protect the Trusted Block contents against modification. According to the method in ISO 16609, a CBC-mode MAC is calculated over the Trusted Block using a randomly-generated triple-DES (TDES) key, and the MAC key itself is encrypted and embedded in the block. For the internal form of the block, the MAC key is encrypted with a randomly chosen fixed-value variant of the PKA master key. For the external form, the MAC key is encrypted with a fixed variant of a key-encrypting key. The MKVP field contains the master key verification pattern for the PKA master key that was used, and is filled with binary zeros if the trusted block is in external format. Various flag fields contain these boolean flags.

- **Active flag** - Contained within the flags field of the required trusted block information section, this flag indicates whether the trusted block is active and ready for use by other callable services. Combined with the use of two separate access control points, the active flag is used to enforce dual control over creation of the block. A person whose active role is authorized to create a trusted block in inactive form creates the block and defines its parameters. An inactive trusted block can only be used to make it active. A person whose active role is authorized to activate an inactive trusted block must approve the block by changing its status to active. See Figure 3 on page 42. The Remote\_Key\_Export callable service can only use an internal active trusted block to generate or export DES keys according to the parameters defined in the trusted block.
- **Date checking flag** - Contained within the optional activation and expiration date subsection of the required trusted block information subsection, this flag indicates whether the coprocessor checks the activation and expiration dates for the trusted block. If the date checking flag is on, the coprocessor compares the activation and expiration dates in the optional subsection to the coprocessor internal real time clock, and processing terminates if either date is out of range. If this flag is off or the activation and expiration dates subsection is not defined, the device does no date checking. If this flag is off and the activation and expiration dates subsection is defined, date checking can still be performed outside of the device if required. The date checking flag enables use of the trusted block in systems where the coprocessor clock is not set.

**Trusted block name** - This field optionally contains a text string that is a name (key label) for the trusted block. It is included in the block for use by an external system such as a host computer, and not by the card itself. In the zSeries system, the label can be checked by RACF to determine if use of the block is authorized. It is possible to disable use of trusted blocks that have been compromised or need to be removed from use for other reasons by publishing a revocation list containing the key names for the blocks that must not be used. Code in the host system could check each trusted block prior to it being used in the cryptographic coprocessor, to ensure that the name from that block is not in the revocation list.

**Expiration date and activation dates** - The trusted block can optionally contain an expiration date and an activation date. The activation date is the first day on which the block can be used, and the expiration date is the last day when the block can be used. If these dates are present, the date checking flag in the trusted block will indicate whether the coprocessor should check the dates using its internal real-time clock. In the case of a system that does set the coprocessor clock, checking would have to be performed by an application program prior to using the trusted block. This is not quite as secure, but it is still valuable, and storing the dates in the block itself is preferable to making the application store it somewhere else and maintain the association between the separate trusted block and activation and expiration dates.



**Application-defined data** - The trusted block can hold data defined and understood only by the host application program. This data is included in the protected contents of the trusted block, but it is not used or examined in any way by the coprocessor. By including its own data in the trusted block, an application can guarantee that the data is not changed in any way, since it is protected in the same way as the other trusted block contents.

**Rules** - A variable number of rules can be included in the block. Each rule contains information on how to generate or export a symmetric key, including values for variants to be used in order to provide keys in the formats expected by systems with differing cryptographic architectures. Use of the rules are described in the topics covering key generation and export using the RKX function. This table summarizes the required and optional values of each rule.

Field name	Required field	Description
Rule ID	Yes	Specifies the 8-character name of the rule
Operation	Yes	Indicates whether this rule generates a new key or exports an existing key
Generated key length	Yes	Specifies the length of the key to be generated
Key-check algorithm ID	Yes	Specifies which algorithm to use to compute the optional key-check value (KCV). Options are <ul style="list-style-type: none"> <li>• No KCV</li> <li>• Encrypt zeros with the key</li> <li>• Compute MDC-2 hash of the key</li> </ul>
Symmetric-encrypted output format	Yes	Specifies the format of the required symmetric-encrypted key output. Options are: <ul style="list-style-type: none"> <li>• CCA key token</li> <li>• RKX key token</li> </ul>
Asymmetric-encrypted output format	Yes	Specifies the format of the optional asymmetric-encrypted key output (key is encrypted with RSA). Options are: <ul style="list-style-type: none"> <li>• No asymmetric-encrypted key output</li> <li>• Encrypt in PKCS1.2 format</li> <li>• Encrypt in RSAOAEP format</li> </ul>
Transport-key variant	No	Specifies the variant to apply to the transport key prior to it being used to encrypt the key being generated or exported
Export key CV	No	Specifies the CCA CV to apply to the transport key prior to it being used to encrypt the key being generated or exported. The CV defines permitted uses for the exported key.
Export key length limits	No	Defines the minimum and maximum lengths of the key that can be exported with this rule.
Output key variant	No	Specifies the variant to apply to the generated or exported key prior to it being encrypted.
Export-key rule reference	No	Specifies the rule ID for the rule that must have been used to generate the key being exported, if that key is an RKX key token.

Field name	Required field	Description
Export-key CV restrictions	No	Defines masks and templates to use to restrict the possible CV values that a source key can have when being exported with RKX. Only applies if the key is a CCA key token. This can control the types of CCA keys that can be processed using the rule.
Export-key label template	No	Specifies the <i>key label</i> of the key token that contains the source key to be exported. A key label is a name used to identify a key. The rule can optionally contain a key label template, which will be matched against the host-supplied key label, using a wildcard (*) so that the template can match a set of related key labels. The operation will only be accepted if the supplied label matches the wildcard template in the rule.

## Changes to the CCA API

These changes have been made to the CCA API to support remote key loading using trusted blocks:

- A new Trusted Block Create (CSNDTBC and CSNETBC) callable service has been developed to securely create trusted blocks under dual control.
- A new Remote Key Export (CSNDRKX and CSNFRKX) callable service has been developed to generate or export DES and TDES keys under control of the rules contained in a trusted block.
- The Digital Signature Verify (CSNDDSV) callable service has been enhanced so that, in addition to verifying ordinary CCA RSA keys, it can use the RSA public key contained in a trusted block to verify digital signatures.
- The PKA Key Import (CSNDPKI) callable service has been enhanced so it can import an RSA key into the CCA domain. In addition, the verb can import an external format trusted block into an internal format trusted block, ready to be used in the local system.
- The PKA Key Token Change (CSNDKTC and CSNFKTC) callable service has been enhanced so that it can update trusted blocks to the current PKA master key when the master key is changed. A trusted block contains an embedded MAC key enciphered under the PKA master key. When the PKA master key is changed, the outdated MAC key and the trusted block itself need to be updated to reflect the current PKA master key.
- The MAC Generate (CSNBMGN) and MAC Verify (CSNBMVR) callable services have been enhanced to add ISO 16609 TDES MAC support in which the text will be CBC-TDES encrypted using a double-length key and the MAC will be extracted from the last block.
- The PKA key storage callable services support trusted blocks.

## The RKX key token

CCA normally uses key tokens that are designed solely for the purposes of protecting the key value and carrying metadata associated with the key to control its use by CCA cryptographic functions. The remote key loading design introduces a new type of key token called an RKX key token. The purpose of this token is somewhat different, and its use is connected directly with the Remote Key Export callable service added to CCA of the remote key loading design.

The RKX key token uses a special structure that binds the token to a specific trusted block, and allows sequences of Remote Key Export calls to be bound together as if they were an atomic operation. This allows a series of related

key-management operations to be performed using the Remote Key Export callable service. These capabilities are made possible by incorporating these three features into the RKX key token structure:

- The key is enciphered using a variant of the MAC key that is in the trusted block. A fixed, randomly-derived variant is applied to the key prior to it being used. As a result, the enciphered key is protected against disclosure since the trusted block MAC key is itself protected at all times.
- The structure includes the rule ID contained in the trusted block rule that was used to create the key. A subsequent call to the Remote Key Export callable service can use this key with a trusted block rule that references this rule ID, effectively chaining use of the two rules together securely.
- A MAC is computed over the encrypted key and the rule ID, using the same MAC key that is used to protect the trusted block itself. This MAC guarantees that the key and the rule ID cannot be modified without detection, providing integrity and binding the rule ID to the key itself. In addition, the MAC will only verify if the RKX key token is used with the same trusted block that created the token, thus binding the key to that specific trusted block.

This figure shows a simplified conceptual view of the RKX key token structure.

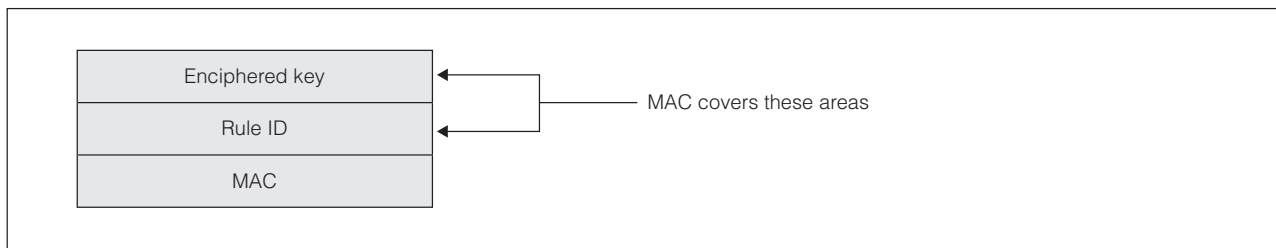
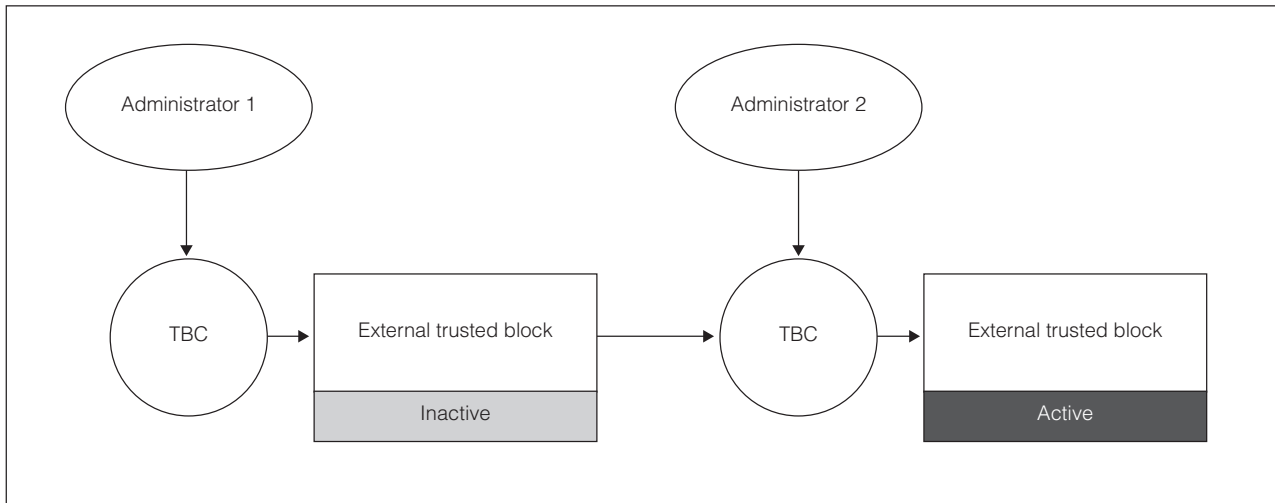


Figure 2. Simplified RKX key-token structure

## Using trusted blocks

These examples illustrate how trusted blocks are used with the new and enhanced CCA callable services.

**Creating a trusted block:** This figure illustrates the steps used to create a trusted block.



*Figure 3. Trusted block creation*

A two step process is used to create a trusted block. Trusted blocks are structures that could be abused to circumvent security if an attacker could create them with undesirable settings, and the requirement for two separate and properly authorized people makes it impossible for a single dishonest employee to create such a block. A trusted block cannot be used for any operations until it is in the active state. Any number of trusted blocks can be created in order to meet different needs of application programs.

**Exporting keys with Remote\_Key\_Export:** This figure shows the process for using a trusted block in order to export a DES or TDES key. This representation is at a very high level in order to illustrate the basic flow.

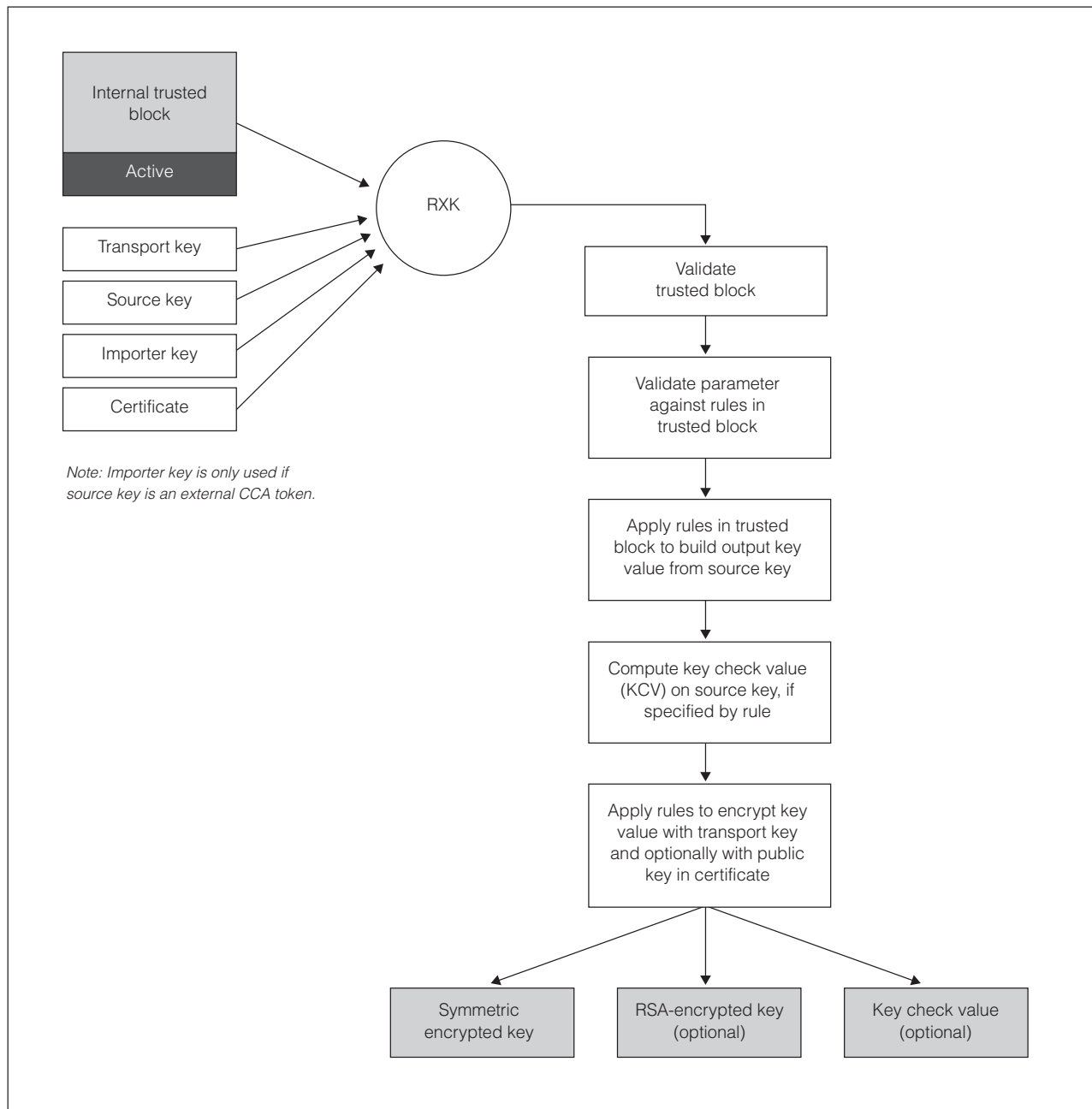


Figure 4. Exporting keys using a trusted block

The Remote Key Export callable service is called with these main parameters:

- A trusted block, in the active state, defines how the export operation is to be processed, including values to be used for things such as variants to apply to the keys.
- The key to be exported, shown previously as the source key. The source key takes one of two forms:
  1. A CCA DES key token
  2. An RKX key token

- A key-encrypting key, shown in the figure as the importer key. This is only used if the source key is an external CCA DES key token, encrypted under a KEK. In this case, the KEK is the key needed to obtain the cleartext value of the source key.
- A transport key, either an exporter KEK or an RKX key token, used to encrypt the key being exported.
- An optional public key certificate which, if included, contains the certified public key for a specific ATM. The certificate is signed with the ATM vendor's private key, and its corresponding public key must be contained in the trusted block so that this certificate can be validated. The public key contained in the certificate can be used to encrypt the exported key.

The processing steps are simple at a high level, but there are many options and significant complexity in the details.

- The trusted block itself is validated. This includes several types of validation.
  - Cryptographic validation using the MAC that is embedded in the block, in which the MAC key is decrypted using the coprocessor's master key, and the MAC is then verified using that key. This verifies the block has not been corrupted or tampered with, and it also verifies that the block is for use with this coprocessor since it will only succeed if the master key is correct.
  - Consistency checking and field validation, in which the validity of the structure itself is checked, and all values are verified to be within defined ranges.
  - Fields in the trusted block are checked to see if all requirements are met for use of this trusted block. One check which is always required is to ensure that the trusted block is in the active state prior to continuing. Another check, which is optional based on the contents of the trusted block, is to ensure the operation is currently allowed by comparing the date of the coprocessor real-time clock to the activation and expiration dates defined in the trusted block.
- Input parameters to the Remote Key Export callable service are validated against rules defined for them within the trusted block. For example:
  - The rule can restrict the length of the key to be exported.
  - The rule can restrict the control vector values for the key to be exported, so only certain key types can be exported with that rule.
- When the export key is decrypted, the rules embedded in the trusted block are then used to modify that key to produce the desired output key value. For example, the trusted block can contain a variant to be exclusive-ORed with the source key prior to when that key is encrypted. Many non-IBM cryptographic systems use variants to provide key separation to restrict a key from improper use.
- A key check value (KCV) can be optionally computed for the source key. If the KCV is computed, the trusted block allows for one of two key check algorithms to be used: (1) encrypting binary zeros with the key, or (2) computing an MDC-2 hash of the key. The KCV is returned as output from the Remote Key Export function.
- The export key, which could possibly be modified with a variant according to the rules in the trusted block, is enciphered with the transport key. The rules can specify that the key be created in one of two formats: (1) a CCA key token, or (2) the new RKX key token, described previously. With proper selection of rule options, the CCA key token can create keys that can be used in non-CCA systems. The key value can be extracted from the CCA key token resulting in a generic encrypted key, with variants and other options as defined in the rule.

Two optional fields in the trusted block may modify the transport key prior to it being used to encrypt the source key:

- The trusted block can contain a CCA control vector (CV) to be exclusive-ORed with the transport key prior to it being used to encrypt the export key. This exclusive-OR process is the standard way CCA applies a CV to a key.
- In addition to the CV described previously, the trusted block can also contain a variant to be exclusive-ORed with the transport key prior to its use.

If a variant and CV are both present in the trusted block, the variant is applied first, then the CV.

- The export key can optionally be encrypted with the RSA public key identified by the certificate parameter of the Remote Key Export callable service, in addition to encrypting it with the transport key as described previously. These two encrypted versions of the export key are provided as separate outputs of the Remote Key Export callable service. The trusted block allows a choice of encrypting the key in either PKCS1.2 format or PKCSOAEP format.

**Generating keys with Remote\_Key\_Export:** This figure shows the process for using a trusted block to generate a new DES or TDES key. This representation is at a very high level in order to illustrate the basic flow.



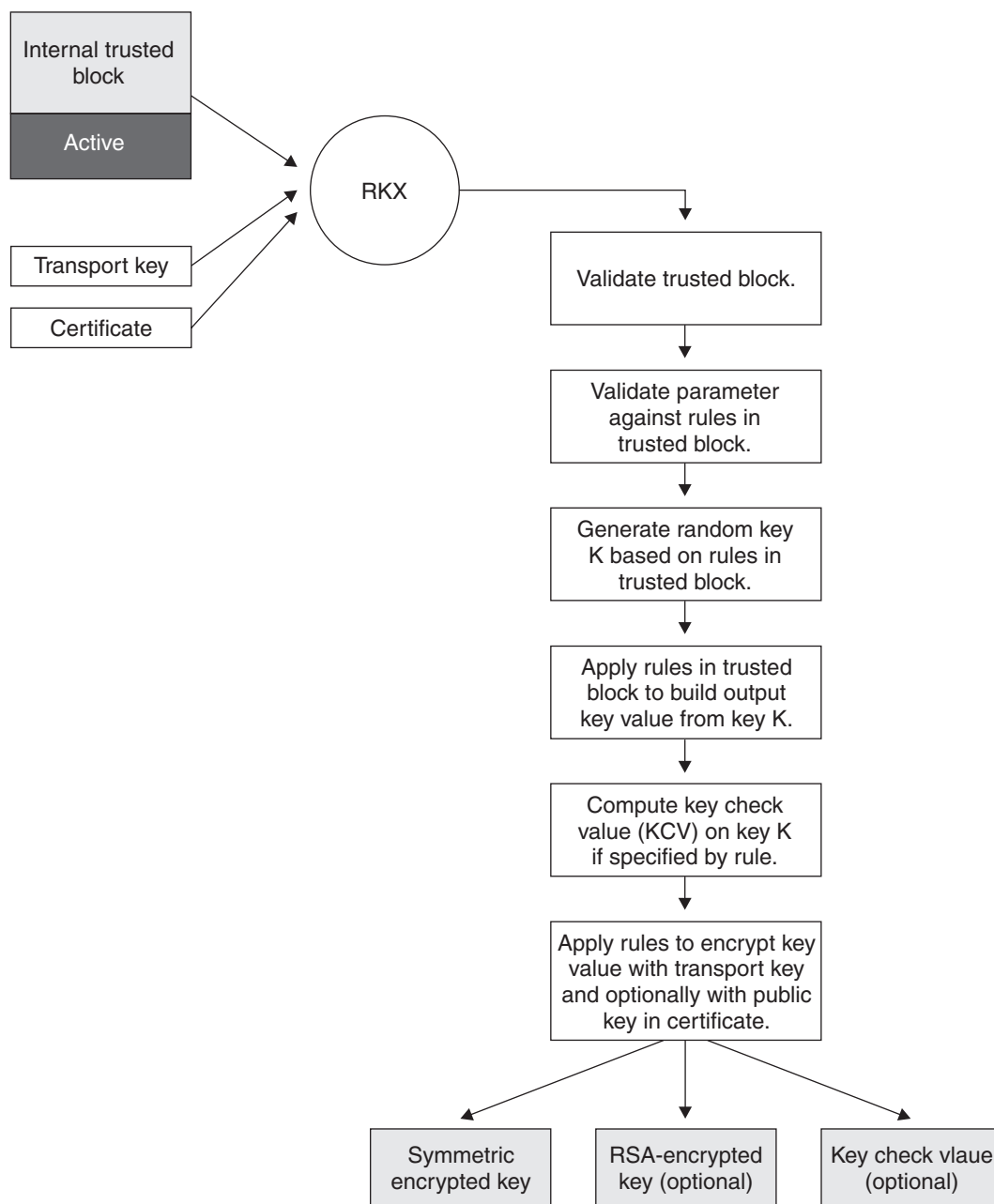


Figure 5. Generating keys using a trusted block

For key generation, the Remote Key Export callable service is called with these main parameters:

- A trusted block, in the internal active state, which defines how the key generation operation is to be processed, including values to be used for things such as variants to apply to the keys. The generated key is encrypted by a variant of the MAC key contained in a trusted block.
- An optional public key certificate which, if included, contains the certified public key for a specific ATM. The certificate is signed with the ATM vendor's private key, and its corresponding public key must be contained in the trusted block so that this certificate can be validated. The public key contained in the certificate can be used to encrypt the generated key.

The processing steps are simple at a high level, but there are many options and significant complexity in the details. Most of the processing steps are the same as those described previously for key export. Therefore, only those processing steps that differ are described here in detail.

- Validation of the trusted block and input parameters is done as described for export previously.
- The DES or TDES key to be returned by the Remote Key Export callable service is randomly generated. The trusted block indicates the length for the generated key.
- The output key value is optionally modified by a variant as described previously for export, and then encrypted in the same way as for export using the Transport key and optionally the public key in the certificate parameter.
- The key check value (KCV) is optionally computed for the generated key using the same method as for an exported key.

### Remote key distribution scenario

The new and modified CCA functions for remote key loading are used together to create trusted blocks, and then generate or export keys under the control of those trusted blocks. This figure summarizes the flow of the CCA functions to show how they are used:

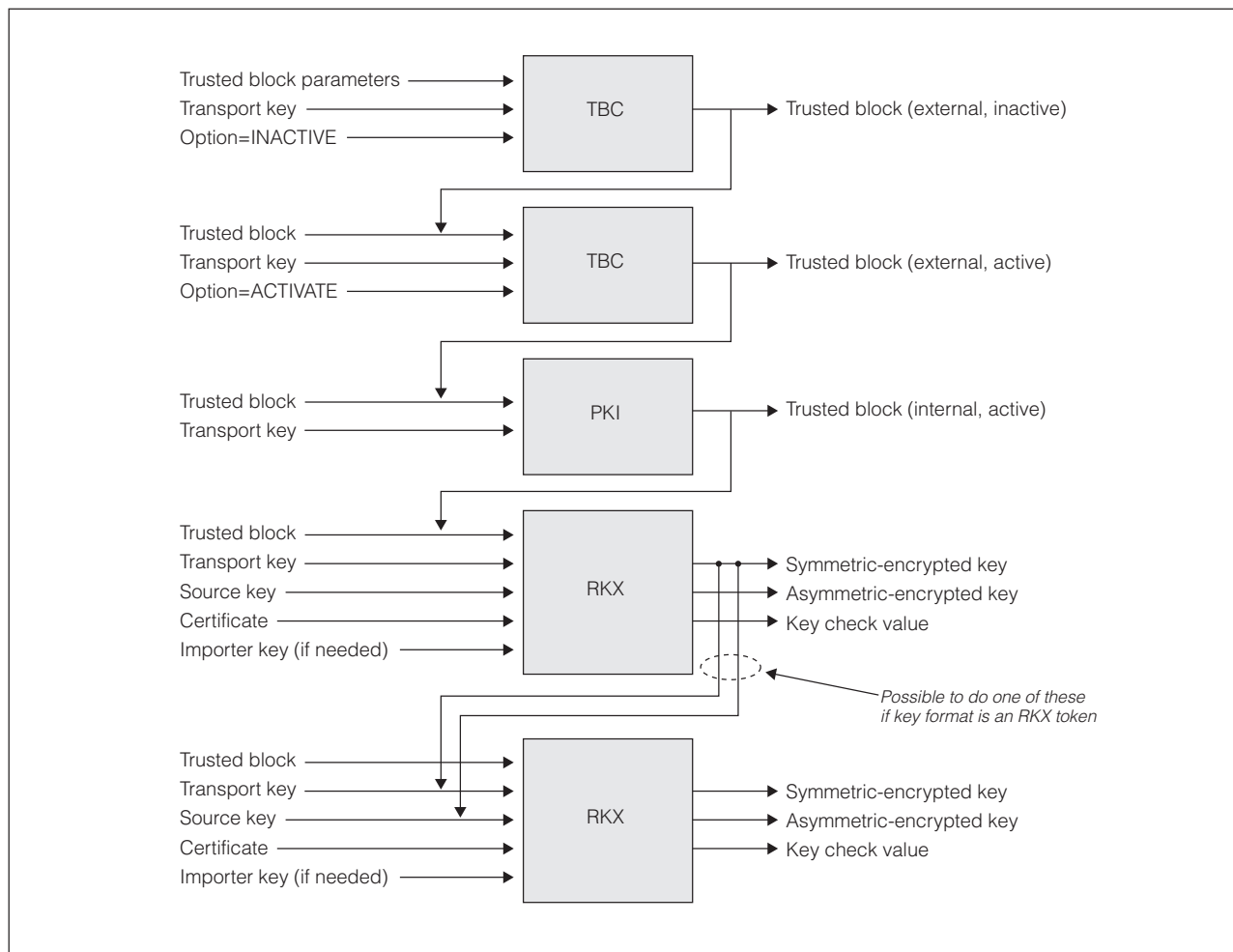


Figure 6. Typical flow of callable services for remote key export

**Usage example:** The scenario described shows how these functions might be combined in a real-life application to distribute a key to an ATM and keep a copy for local use. Some of the terminology used reflects typical terms used in ATM networks. The example illustrates a fairly complex real-world key distribution scenario, in which these values are produced.

- A TMK (Terminal Master Key), which is the root KEK used by the ATM to exchange other keys, is produced in two forms: (1) encrypted under the ATM public key, so it can be sent to the ATM, and (2) as an RKX key token that will be used in subsequent calls to the Remote Key Export callable service to produce other keys.
- A key-encrypting key KEK1 that is encrypted under the TMK in a form that can be understood by the ATM.
- A PIN-encrypting key PINKEY be used by the ATM to encrypt customer-entered PINs and by the host to verify those PINs. The PINKEY is produced in two forms: (1) encrypted under KEK1 in a form that can be understood by the ATM, and (2) as a CCA internal DES key token with the proper PIN-key CV, encrypted under the CCA DES master key and suitable for use with the coprocessor.

It takes seven steps to produce these keys using the Remote Key Export callable service. These steps use a combination of five rules contained in a single trusted block. The rules in this example are referred to as GENERAT1, GENERAT2, EXPORT1, EXPORT2, and EXPORT3.

1. Use the Remote Key Export callable service with rule ID "GENERAT1" to generate a TMK for use with the ATM. The key will be output in two forms:
  - a.  $e_{Pu}(TMK)$ : Encrypted under the ATM public key, supplied in the certificate parameter, CERT
  - b.  $RKX(TMK)$ : As an RKX key token, suitable for subsequent input to the CSNDRKX callable service
2. Use the Remote Key Export callable service with rule ID "GENERAT2" to generate a key-encrypting key (KEK1) as an RKX key token,  $RKX(KEK1)$
3. Use the Remote Key Export callable service with rule ID "GENERAT2" to generate a PIN key (PINKEY) as an RKX key token:  $RKX(PINKEY)$ .
4. Use the Remote Key Export callable service with rule ID "EXPORT1 " to export KEK1 encrypted under the TMK as a CCA DES key token using a variant of zeros applied to the TMK. This produces  $e_{TMK}(KEK1)$ .
5. Use the Remote Key Export callable service with rule ID "EXPORT2 " to export PINKEY encrypted under KEK1 as a CCA token using a variant of zeros applied to KEK1. This produces  $e_{KEK1}(PINKEY)$ .
6. Use the Remote Key Export callable service with rule ID "EXPORT3 " to export PINKEY under KEK2, an existing CCA key-encrypting key on the local server. This produces  $e_{KEK2}(PINKEY)$ , with the CCA control vector for a PIN key.
7. Use the Key Import callable service to import the PINKEY produced in step 6 into the local system as an operational key. This produces  $e_{MK}(PINKEY)$ , a copy of the key encrypted under the local DES master key (MK) and ready for use by CCA PIN API functions.

## Remote key distribution benefits

CCA support for remote key loading solves one new problem, and one long-standing problem. This support allows the distribution of initial keys to ATMs and other remote devices securely using public-key techniques, in a flexible way that can support a wide variety of different cryptographic architectures. They also make it far easier and far more secure to send keys to non-CCA systems when

those keys are encrypted with a triple-DES key-encrypting key. These changes make it easier for customers to develop more secure systems.

## Diversifying keys

CCA supports several methods for diversifying a key using the diversified key generate callable service. Key-diversification is a technique often used in working with smart cards. In order to secure interactions with a population of cards, a "key-generating key" is used with some data unique to a card to derive ("diversify") keys for use with that card. The data is often the card serial number or other quantity stored on the card. The data is often public, and therefore it is very important to handle the key-generating key with a high degree of security lest the interactions with the whole population of cards be placed in jeopardy.

In the current implementation, several methods of diversifying a key are supported: **CLR8-ENC**, **TDES-ENC**, **TDES-DEC**, **SESS-XOR**, **TDES-XOR**, **TDESEMV2** and **TDESEMV4**. The first two methods triple-encrypt data using the generating\_key to form the diversified key. The diversified key is then multiply-enciphered by the master key modified by the control vector for the output key. The TDES-DEC method is similar except that the data is triple-decrypted.

The **SESS-XOR** method provides a means for modifying an existing DATA, DATAC, MAC, DATAM, or MACVER, DATAMV single- or double-length key. The provided data is exclusive-ORed into the clear value of the key. This form of key diversification is specified by several of the credit card associations.

The **TDES-ENC** and **TDES-DEC** methods permit the production of either another key-generating key, or a final key. Control-vector bits 19 – 22 associated with the key-generating key specify the permissible type of final key. (See DKYGENKY in Figure 11 on page 899.) Control-vector bits 12 – 14 associated with the key-generating key specify if the diversified key is a final key or another in a series of key-generating keys. Bits 12 – 14 specify a counter that is decreased by one each time the diversified key generate service is used to produce another key-generating key. For example, if the key-generating key that you specify has this counter set to B'010', then you must specify the control vector for the generated\_key with a DKYGENKY key type having the counter bits set to B'001' and specifying the same final key type in bits 19 – 22. Use of a generating\_key with bits 12 – 14 set to B'000' results in the creation of the final key. Thus you can control both the number of diversifications required to reach a final key, and you can closely control the type of the final key.

The **TDESEMV2**, **TDESEMV4**, and **TDES-XOR** methods also derive a key by encrypting supplied data including a transaction counter value received from an EMV smart card. The processes are described in detail at "Visa and EMV-related smart card formats and processes" on page 954. Refer to "Working with Europay–MasterCard–Visa smart cards" on page 466 to understand the various verbs you can use to operate with EMV smart cards.

## Callable Services for Dynamic CKDS Update

ICSF provides the dynamic CKDS update services that allow applications to directly manipulate both the DASD copy and in-storage copy of the current CKDS.

**Note:** Applications using the dynamic CKDS update callable services can run concurrently with other operations that affect the CKDS, such as KGUP, CKDS conversion, REFRESH, and dynamic master key change. An operation

can fail if it needs exclusive or shared access to the same DASD copy of the CKDS that is held shared or exclusive by another operation. ICSF provides serialization to prevent data loss from attempts at concurrent access, but your installation is responsible for the effective management of concurrent use of competing operations. Consult your system administrator or system programmer for your installation guidelines.

The syntax of the CKDS key record create, CKDS key record read, and CKDS key record write services is identical with the same services provided by the Transaction Security System security application programming interface. Key management applications that use these common interface verbs can run on both systems without change.

Additional versions of CKDS key record create, CKDS key record read, and CKDS key record write (introduced in HCR7780) must be used for variable-length key tokens. These are the CKDS Key Record Create2, CKDS Key Record Read2, and CKDS Key Record Write2 callable services. These services also support existing DES and AES tokens.

### **CKDS Key Record Create Callable Service (CSNBKRC and CSNEKRC)**

This service accepts a key label and creates a null key record in both the DASD copy and in-storage copy of the CKDS. The record contains a key token set to binary zeros and is identified by the key label passed in the call statement. The key label must be unique.

Prior to updating a key record using either the dynamic CKDS update services or KGUP, that record must already exist in the CKDS. You can use either the CKDS key record create service, KGUP, or your key entry hardware to create the initial record in the CKDS.

### **CKDS Key Record Create2 Callable Service (CSNBKRC2 and CSNEKRC2)**

This service accepts a key label and optionally, a symmetric key token, and creates a key record in both the DASD copy and in-storage copy of the CKDS. The record contains the supplied key token or a null key token and is identified by the key label passed in the call statement. The key label must be unique.

This service must be used with variable-length key tokens. This service supports existing DES and AES key tokens.

### **CKDS Key Record Delete Callable Service (CSNBKRD and CSNEKRD)**

This service accepts a unique key label and deletes the associated key record from both the in-storage and DASD copies of the CKDS. This service deletes the entire record, including the key label from the CKDS.

### **CKDS Key Record Read Callable Service (CSNBKRR and CSNEKRR)**

This service copies an internal key token from the in-storage CKDS to the application storage, where it may be used directly in other cryptographic services. Key labels specified with this service must be unique.

### **CKDS Key Record Read2 Callable Service (CSNBKRR2 and CSNEKRR2)**

This service copies an internal key token from the in-storage CKDS to the application storage, where it may be used directly in other cryptographic services. Key labels specified with this service must be unique.

This service must be used with variable-length key tokens. This service supports existing DES and AES key tokens.

### **CKDS Key Record Write Callable Service (CSNBKRW and CSNEKRW)**

This service accepts an internal key token and a label and writes the key token to the CKDS record identified by the key label. The key label must be unique. Application calls to this service write the key token to both the DASD copy and in-storage copy of the CKDS, so the record must already exist in both copies of the CKDS.

### **CKDS Key Record Write2 Callable Service (CSNBKRW2 and CSNEKRW2)**

This service accepts an internal key token and a label and writes the key token to the CKDS record identified by the key label. The key label must be unique. Application calls to this service write the key token to both the DASD copy and in-storage copy of the CKDS, so the record must already exist in both copies of the CKDS.

This service must be used with variable-length key tokens. This service supports existing DES and AES key tokens.

### **Coordinated KDS Administration Callable Service (CSFCRC and CSFCRC6)**

This service is used to perform the following operations; coordinated CKDS change-mk, coordinated CKDS refresh, coordinated PKDS change-mk, coordinated PKDS refresh, and coordinated TKDS change-mk.

While this service is performing a coordinated change-mk operation, dynamic KDS update services may continue to run in parallel. During a coordinated refresh operation, dynamic KDS update services may continue to be enabled, however they will be temporarily suspended internally until the coordinated refresh completes. If this can not be tolerated, it is recommended to disable dynamic KDS update services when using this service.

In a sysplex environment, this callable service is executed from a single ICSF instance, and the operation is coordinated across all sysplex members sharing the same active KDS. This removes the need for KDS refresh or KDS change-mk operations to be performed locally on every ICSF instance sharing the same active KDS in a sysplex environment.

## **Callable Services that Support Secure Sockets Layer (SSL)**

The Secure Sockets Layer (SSL) protocol, developed by Netscape Development Corporation, provides communications privacy over the Internet. Client/server applications can use the SSL protocol to provide secure communications and prevent eavesdropping, tampering, or message forgery.



ICSF provides callable services that support the RSA-encryption and RSA-decryption of PKCS 1.2-formatted symmetric key data to produce symmetric session keys. These session keys can then be used to establish an SSL session between the sender and receiver.

### **PKA Decrypt Callable Service (CSNDPKD)**

The PKA decrypt callable service uses the corresponding private RSA key to unwrap the RSA-encrypted key and deformat the key value. This service then returns the clear key value to the application.

### **PKA Encrypt Callable Service (CSNDPKE)**

The PKA encrypt callable service encrypts a supplied clear key value under an RSA public key. Currently, the supplied key can be formatted using the PKCS 1.2 or ZERO-PAD methods prior to encryption.

## **System Encryption Algorithm**

**Note:** This topic only applies to systems with the Cryptographic Coprocessor Feature.

ICSF uses either the DES or AES algorithm or the Commercial Data Masking Facility (CDMF) to encipher and decipher data. The CDMF defines a scrambling technique for data confidentiality. It is a substitute for those customers prohibited from receiving IBM products that support DES data confidentiality services. The CDMF data confidentiality algorithm is composed of two processes: a key shortening process and a standard DES process to encipher and decipher data.

Your system can be one of these:

- DES
- CDMF
- DES-CDMF

A DES system protects data using a single-length, double-length, or triple-length DES data-encrypting key and the DES algorithm.

A CDMF system protects data using a single-length DES data-encrypting key and the CDMF. You input a standard single-length data-encrypting key to the encipher (CSNBENC) and decipher (CSNBDEC) callable services. The single-length data-encrypting key that is intended to be passed to the CDMF is called a CDMF key. Cryptographically, it is indistinguishable from a DES data-encrypting key. Prior to the key being used to encipher or decipher data, however, the Cryptographic Coprocessor Feature hardware cryptographically shortens the key of the CDMF process. This transformed, shortened data-encrypting key can be used only in the DES. (It must never be used in the CDMF; this would result in a double shortening of the key.) When used with the DES, a transformed, shortened data-encrypting key produces results identical to those that the CDMF would produce using the original single-length key.

A DES-CDMF system protects data using either the DES or the CDMF. The default is DES.

ICSF provides functions to mark internal IMPORTER, EXPORTER, and DATA key tokens with **data encryption algorithm bits**. IMPORTER and EXPORTER KEKs are marked when they are installed in operational form in ICSF. Your cryptographic key administrator does this. (See *z/OS Cryptographic Services ICSF Administrator's Guide* for details.) Whenever a DATA key is imported or generated in concert with a marked KEK, this marking is transferred to the DATA key token, unless the



token copying function of the callable service is used to override the KEK marking with the marking of the key token passed. These data encryption algorithm bits internally drive the DES or CDMF for the ICSF encryption services. External key tokens are not marked with these data encryption algorithm bits.

IMPORTER and EXPORTER KEKs can have data encryption algorithm bit markings of CDMF (X'80'), DES (X'40'), or SYS-ENC (X'00'). DATA keys generated or imported with marked KEKs will also be marked. A CDMF-marked KEK will transfer a data encryption algorithm bit marking of CDMF (X'80') to the DATA key token. A DES-marked KEK will transfer a data encryption algorithm bit marking of DES (X'00') to the DATA key token. A SYS-ENC-marked KEK will transfer a CDMF (X'80') marking to the DATA key token on a CDMF system, and a DES (X'00') marking to the DATA key token on DES-CDMF and DES systems.

To accomplish token copying of data encryption algorithm marks, a valid internal token of the same key type must be provided in the target key identifier field of the service. The token must have the proper token mark to be copied.

**Notes:**

1. For the multiple secure key import callable service the token markings on the KEK are ignored. In this case, the algorithm choice specified in the rule array determines the markings on the DATA key.
2. Propagation of data encryption algorithm bits and token copying are only performed when the ICSF callable service is performed on the Cryptographic Coprocessor Feature. The PCI Cryptographic Coprocessor, PCI X Cryptographic Coprocessor, and CCA Crypto Express coprocessors do not perform these functions.

Table 6 summarizes the data encryption algorithm bits by key type, and the algorithm they drive in the ICSF encryption services.

*Table 6. Summary of Data Encryption Standard Bits*

Algorithm	Key Type	Bits
CDMF	DATA	X'80'
	KEK	X'80'
DES	DATA	X'00'
	KEK	X'40'
System Default Algorithm	KEK	X'00'

For PCF users, your system programmer specifies a default encryption mode of DES or CDMF when installing ICSF. (See *z/OS Cryptographic Services ICSF System Programmer's Guide* for details.)

## ANSI X9.17 Key Management Services

**Restriction:** ANSI X9.17 keys and ANSI key management services are only supported on the IBM @server zSeries 900.

The ANSI X9.17 key management standard defines a process for protecting and exchanging DES keys. The ANSI X9.17 standard defines methods for generating, exchanging, using, storing, and destroying these keys. ANSI X9.17 keys are protected by the processes of *notarization* and *offsetting*, instead of control vectors. In addition to providing services to support these processes, ICSF also defines and uses an optional process of *partial notarization*.

Offsetting involves exclusive-ORing a key-encrypting key with a counter. The counter, a 56-bit binary number that is associated with a key-encrypting key and contained in certain ANSI X9.17 messages, prevents either a replay or an out-of-sequence transmission of a message. When the associated AKEK is first used, the application initializes the counter. With each additional use, the application increments the counter.

Notarization associates the identities of a pair of communicating parties with a cryptographic key. The notarization process cryptographically combines a key with two 16-byte quantities, the origin identifier and the destination identifier, to produce a notarized key. The notarization process is completed by offsetting the AKEK with a counter.

ICSF makes it possible to divide the AKEK notarization process into two steps. In the first step, partial notarization, the AKEK is cryptographically combined with the origin and destination identifiers and returned in a form that can be stored in the CKDS or application storage. In the second step, the partially notarized AKEK is exclusive OR-ed with a binary counter to complete the notarization process. Partial notarization improves performance when you use an AKEK for many cryptographic service messages, each with a different counter. For details of the partial notarization calculations, refer to “ANSI X9.17 Partial Notarization Method” on page 949.

ICSF provides these callable services to support the ANSI X9.17 key management standard. Except where noted, these callable services have the identical syntax as the Transaction Security System verbs of the same name. With few exceptions, key management applications that use these common callable services, or verbs, can be executed on either system without change. Internal tokens cannot be interchanged; external tokens can be.

### **Key Generate Callable Service Used to Generate an AKEK (CSNBKGN)**

The key generate callable service, described in “Key Generate Callable Service (CSNBKGN and CSNEKGN)” on page 28, can also be used to generate an AKEK in the operational form. It generates either an 8-byte or 16-byte AKEK and places it in a skeleton key token created by the key token build callable service. The length of the AKEK is determined by the key length keyword specified when building the key token.

### **ANSI X9.17 EDC Generate Callable Service (CSNAEGN and CSNGEGN)**

This service generates an ANSI X9.17 error detection code on an arbitrary length string.

### **ANSI X9.17 Key Export Callable Service (CSNAKEX and CSNGKEX)**

This service uses the ANSI X9.17 protocol to export a DATA key or a pair of DATA keys, with or without an AKEK. It also provides the ability to convert a single supplied DATA key or combine two supplied DATA keys into a MAC key.

### **ANSI X9.17 Key Import Callable Service (CSNAKIM and CSNGKIM)**

This service uses the ANSI X9.17 protocol to import a DATA key or a pair of DATA keys, with or without an AKEK. It also provides the ability to convert a single supplied DATA key or combine two supplied DATA keys into a MAC key. The syntax is identical to the Transaction Security System verb, with these exceptions:

- Keys cannot be imported directly into the CKDS.

### **ANSI X9.17 Key Translate Callable Service (CSNAKTR and CSNGKTR)**

This service translates one or two DATA keys or an AKEK from encryption under one AKEK to encryption under another AKEK, using the ANSI X9.17 protocol.

### **ANSI X9.17 Transport Key Partial Notarize Callable Service (CSNATKN and CSNGTKN)**

This service preprocesses or partially notarizes an AKEK with origin and destination identifiers. The partially notarized key is supplied to the ANSI X9.17 key export, ANSI X9.17 key import, or ANSI X9.17 key translate callable service to complete the notarization process. The syntax is identical to the Transaction Security System verb except that:

- The callable service does not update the CKDS.

---

## **Enciphering and Deciphering Data**

The encipher and decipher callable services protect data off the host. ICSF protects sensitive data from disclosure to people who do not have authority to access it. Using algorithms that make it difficult and expensive for an unauthorized user to derive the original clear data within a practical time period assures privacy.

To protect data, ICSF can use the Data Encryption Standard (DES) algorithm to encipher or decipher data or keys. The algorithm is documented in the Federal Information Processing Standard #46. On z900 systems, ICSF also supports the CDMF encryption mode. See “System Encryption Algorithm” on page 47 for more information. The Advanced Encryption Standard (AES) algorithm can also be used to encipher or decipher data or keys. The algorithm is documented in the Federal Information Processing Standard #192.

These services can be used to protect data.

- Decipher Callable Service (CSNBDEC, CSNBDEC1, CSNEDEC and CSNEDEC1)  
The decipher callable service uses encrypted DES data-encrypting keys to decipher data.
- Encipher Callable Service (CSNBENC, CSNBENC1, CSNEENC and CSNEENC1)  
The encipher callable service uses encrypted DES data-encrypting keys to encipher data.
- Symmetric Algorithm Decipher Callable Service (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1)  
The symmetric algorithm decipher callable service uses encrypted AES data-encrypting keys to decipher data.
- Symmetric Algorithm Encipher Callable Service (CSNBSAE, CSNBSAE1, CSNESAE and CSNESAE1)  
The symmetric algorithm Encipher callable service uses encrypted AES data-encrypting keys to encipher data.
- Symmetric Key Decipher Callable Service (CSNBSYD, CSNBSYD1, CSNESYD and CSNESYD1)  
The symmetric key decipher callable service uses clear and encrypted AES and DES data-encrypting keys to decipher data.
- Symmetric Key Encipher Callable Service (CSNBSEY, CSNBSEY1, CSNESY and CSNESY1)

The symmetric key encipher callable service uses clear and encrypted AES and DES data-encrypting keys to encipher data.

---

## Encoding and Decoding Data (CSNBECO, CSNEECO, CSNBDCO, and CSNEDCO)

The encode and decode callable services perform functions with clear keys. Encode enciphers 8 bytes of data using the electronic code book (ECB) mode of the DES and a clear key. Decode does the inverse of the encode service. These services are available only on a DES-capable system. (See “System Encryption Algorithm” on page 52 for more information.)

---

## Translating Ciphertext (CSNBCTT or CSNBCTT1 and CSNECTT or CSNECTT1)

**Restriction:** These services are only available on the IBM @server zSeries 900.

ICSF also provides a ciphertext translate callable service. It decipheres encrypted data (ciphertext) under one encryption key and reenciphers it under another key without having the data appear in the clear outside the cryptographic feature. Such a function is useful in a multiple node network, where sensitive data is passed through multiple nodes prior to it reaching its final destination. Different nodes use different keys in the process. For more information about different nodes, see “Using the Ciphertext Translate and Ciphertext Translate2 Callable Service” on page 70.

The keys cannot be used for the encipher and decipher callable services. (See “System Encryption Algorithm” on page 52 for more information.)

---

## Translating Ciphertext (CSNBCTT2 or CSNBCTT3 and CSNECTT2 or CSNECTT3)

**Restriction:** These services are only available on the IBM zEnterprise EC12 or later servers.

ICSF provides a ciphertext translate callable service. It decipheres encrypted data (ciphertext) under one encryption key and reenciphers it under another key without having the data appear in the clear outside the cryptographic feature. Such a function is useful in a multiple node network, where sensitive data is passed through multiple nodes prior to reaching its final destination. Different nodes use different keys in the process. For more information about different nodes, see “Using the Ciphertext Translate and Ciphertext Translate2 Callable Service” on page 70.

The translate keys cannot be used for the encipher and decipher callable services.

---

## Managing Data Integrity and Message Authentication

To ensure the integrity of transmitted messages and stored data, ICSF provides:

- Message authentication code (MAC)
- Several hashing functions, including modification detection code (MDC), SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, RIPEMD-160 and MD5.

(See Chapter 9, “Using Digital Signatures,” on page 557 for an alternate method of message authentication using digital signatures.)

The choice of callable service depends on the security requirements of the environment in which you are operating. If you need to ensure the authenticity of the sender and also the integrity of the data, consider message authentication code processing. If you need to ensure the integrity of transmitted data in an environment where it is not possible for the sender and the receiver to share a secret cryptographic key, consider hashing functions, such as the modification detection code process.

## Message Authentication Code Processing

The process of verifying the integrity and authenticity of transmitted messages is called *message authentication*. Message authentication code (MAC) processing allows you to verify that a message was not altered or a message was not fraudulently introduced onto the system. You can check that a message you have received is the same one sent by the message originator. The message itself may be in clear or encrypted form. The comparison is performed within the cryptographic feature. Since both the sender and receiver share a secret cryptographic key used in the MAC calculation, the MAC comparison also ensures the authenticity of the message.

In a similar manner, MACs can be used to ensure the integrity of data stored on the system or on removable media, such as tape.

ICSF provides support for both single-length and double-length MAC generation and MAC verification keys. With the ANSI X9.9-1 single key algorithm, use the single-length MAC and MACVER keys.

ICSF provides support for the use of data-encrypting keys in the MAC generation and verification callable services, and also the use of a MAC generation key in the MAC verification callable service. This support permits ICSF MAC services to interface more smoothly with non-CCA key distribution system, including those implementing the ANSI X9.17 protocol.

### **HMAC Generation Callable Service (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1)**

When a message is sent, an application program can generate an authentication code for it using the HMAC generation callable service. The callable service computes the message authentication code using FIPS-198 Keyed-Hash Message Authentication Code method.

### **HMAC Verification Callable Service (CSNBHMGV or CSNBHMGV1 and CSNEHMGV or CSNEHMGV1)**

When the receiver gets the message, an application program calls the HMAC verification callable service. The callable service verifies a MAC by generating another MAC and comparing it with the MAC received with the message. If the two codes are the same, the message sent was the same one received. A return code indicates whether the MACs are the same.

The MAC verification callable service can use FIPS-198 Keyed-Hash Message Authentication Code method.

### **MAC Generation Callable Service (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)**

When a message is sent, an application program can generate an authentication code for it using the MAC generation callable service. The callable service computes the message authentication code using one of these methods:

- Using the ANSI X9.9-1 single key algorithm, a single-length MAC generation key or data-encrypting key, and the message text.
- Using the ANSI X9.19 optional double key algorithm, a double-length MAC generation key and the message text.
- Using Europay, MasterCard and Visa (EMV) padding rules with a single-length MAC key or double-length MAC key and the message text.
- Using ISO 16609 algorithm with a double-length MAC or a double-length DATA key and the message text.

ICSF allows a MAC to be the leftmost 32 or 48 bits of the last block of the ciphertext or the entire last block (64 bits) of the ciphertext. The originator of the message sends the message authentication code with the message text.

### **MAC Verification Callable Service (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)**

When the receiver gets the message, an application program calls the MAC verification callable service. The callable service verifies a MAC by generating another MAC and comparing it with the MAC received with the message. If the two codes are the same, the message sent was the same one received. A return code indicates whether the MACs are the same.

The MAC verification callable service can use either of these methods to generate the MAC for authentication:

- The ANSI X9.9-1 single key algorithm, a single-length MAC verification or MAC generation key (or a data-encrypting key), and the message text.
- The ANSI X9.19 optional double key algorithm, a double-length MAC verification or MAC generation key and the message text.
- Using Europay, MasterCard and Visa (EMV) padding rules with a single-length MAC key or double-length MAC key and the message text.
- Using ISO 16609 algorithm with a double-length MAC or a double-length DATA key and the message text.

The method used to verify the MAC should correspond with the method used to generate the MAC.

### **Symmetric MAC Generate Callable Service (CSNBSMG, CSNBSMG1, CSNESMG and CSNESMG1)**

This service supports generating a MAC using a clear AES key. The algorithms supported are CBC-MAC and XCBC-MAC (AES-XCBC-MAC-96 and AES-XCBC-PRF-128)

### **Symmetric MAC Verify Callable Service (CSNBSMV, CSNBSMV1, CSNESMV and CSNESMV1)**

This service supports verifying a MAC using a clear AES key. The algorithms supported are CBC-MAC and XCBC-MAC (AES-XCBC-MAC-96 and AES-XCBC-PRF-128)

## **Hashing Functions**

Hashing functions include one-way hash generation and modification detection code (MDC) processing.



## One-Way Hash Generate Callable Service (CSNBOWH or CSNBOWH1 and CSNEOWH or CSNEOWH1)

This service hashes a supplied message. Supported hashing methods include:

- SHA-1<sup>2</sup>
- SHA-224
- SHA-256
- SHA-384
- SHA-512
- RIPEMD-160
- MD5

## MDC Generation Callable Service (CSNBMDG or CSNBMDG1 and CSNEMDG or CSNEMDG1)

The modification detection code (MDC) provides a form of support for data integrity. The MDC allows you to verify that data was not altered during transmission or while in storage. The originator of the data ensures that the MDC is transmitted with integrity to the intended receiver of the data. For instance, the MDC could be published in a reliable source of public information. When the receiver gets the data, an application program can generate an MDC, and compare it with the original MDC value. If the MDC values are equal, the data is accepted as unaltered. If the MDC values differ the data is assumed to be bogus.

Supported hashing methods through the MDC generation callable service are:

- MDC-2
- MDC-4
- PADMDC-2
- PADMDC-4

In a similar manner, MDCs can be used to ensure the integrity of data stored on the system or on removable media, such as tape.

When data is sent, an application program can generate a modification detection code for it using the MDC generation callable service. The callable service computes the modification detection code by encrypting the data using a publicly-known cryptographic one-way function. The MDC is a 128-bit value that is easy to compute for specific data, yet it is hard to find data that will result in a given MDC.

Once an MDC has been established for a file, the MDC generate service can be run at any other time on the file. The resulting MDC can then be compared with the previously established MDC to detect deliberate or inadvertent modification.

---

## Managing Personal Authentication

The process of validating personal identities in a financial transaction system is called *personal authentication*. The personal identification number (PIN) is the basis for verifying the identity of a customer across the financial industry networks. ICSF checks a customer-supplied PIN by verifying it using an algorithm. The financial industry needs functions to generate, translate, and verify PINs. These functions prevent unauthorized disclosures when organizations handle personal identification numbers.

---

2. The Secure Hash Algorithm (SHA) is also called the Secure Hash Standard (SHS), which Federal Information Processing Standard (FIPS) Publication 180 defines.



ICSF supports these algorithms for generating and verifying personal identification numbers:

- IBM 3624
- IBM 3624 PIN offset
- IBM German Bank Pool
- IBM German Bank Pool PIN Offset (GBP-PINO)
- VISA PIN validation value
- Interbank

With ICSF, you can translate PIN blocks from one format to another. ICSF supports these formats:

- ANSI X9.8
- ISO formats 0, 1, 2, 3
- VISA formats 1, 2, 3, 4
- IBM 4704 Encrypting PINPAD format
- IBM 3624 formats
- IBM 3621 formats
- ECI formats 1, 2, 3

With the capability to translate personal identification numbers into different PIN block formats, you can use personal identification numbers on different systems.

## Verifying Credit Card Data

The Visa International Service Association (VISA) and MasterCard International, Incorporated have specified a cryptographic method to calculate a value that relates to the personal account number (PAN), the card expiration date, and the service code. The VISA card-verification value (CVV) and the MasterCard card-verification code (CVC) can be encoded on either track 1 or track 2 of a magnetic striped card and are used to detect forged cards. Because most online transactions use track-2, the ICSF callable services generate and verify the CVV<sup>3</sup> by the track-2 method.

The VISA CVV generate callable service calculates a 1- to 5-byte value through the DES-encryption of the PAN, the card expiration date, and the service code using two data-encrypting keys or two MAC keys. The VISA CVV service verify callable service calculates the CVV by the same method, compares it to the CVV supplied by the application (which reads the credit card's magnetic stripe) in the *CVV\_value*, and issues a return code that indicates whether the card is authentic.

### Clear PIN Encrypt Callable Service (CSNBCPE and CSNECPE)

To format a PIN into a PIN block format and encrypt the results, use the Clear PIN Encrypt callable service. You can also use this service to create an encrypted PIN block for transmission. With the RANDOM keyword, you can have the service generate random PIN numbers. Use of this service requires a PCIXCC or CCA Crypto Express coprocessor. An enhanced PIN security mode on PCICC, PCIXCC, and CCA Crypto Express coprocessors, is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. See “Clear PIN Encrypt (CSNBCPE and CSNECPE)” on page 476 for more information.

---

3. The VISA CVV and the MasterCard CVC refer to the same value. CVV is used here to mean both CVV and CVC.

## **Clear PIN Generate Alternate Callable Service (CSNBCPA and CSNECPA)**

To generate a clear VISA PIN validation value from an encrypted PIN block, call the clear PIN generate alternate callable service. This service also supports the IBM-PINO algorithm to produce a 3624 offset from a customer selected encrypted PIN.

An enhanced PIN security mode is available for extracting PINs from encrypted PIN blocks. This mode only applies on PCICC, PCIXCC, and CCA Crypto Express coprocessors, when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. See “Clear PIN Generate Alternate (CSNBCPA and CSNECPA)” on page 485 for more information.

**Note:** The PIN block must be encrypted under either an input PIN-encrypting key (IPINENC) or output PIN-encrypting key (OPINENC). Using an IPINENC key requires NOCV keys to be enabled in the CKDS. Functions other than VISA PIN validation value generation require the optional PCICC, PCIXCC or CCA Crypto Express coprocessor.

## **Clear PIN Generate Callable Service (CSNBPGN and CSNEPGN)**

To generate personal identification numbers, call the Clear PIN generate callable service. Using a PIN generation algorithm, data used in the algorithm, and the PIN generation key, the callable service generates a clear PIN, a PIN verification value, or an offset. The callable service can only execute in special secure mode, which is described in “Special Secure Mode” on page 10.

## **CVV Key Combine Callable Service (CSNBCKC and CSNECKC)**

This callable service combines 2 single-length CCA internal key tokens into 1 double-length CCA key token containing a CVVKEY-A key type. This combined double-length key satisfies current VISA requirements and eases translation between TR-31 and CCA formats for CVV keys.

The callable service name for AMODE(64) is CSNECKC.

## **Encrypted PIN Generate Callable Service (CSNBEPG and CSNEEPG)**

To generate personal identification numbers, call the Encrypted PIN generation callable service. Using a PIN generation algorithm, data used in the algorithm, and the PIN generation key, the callable service generates a PIN and using a PIN block format and the PIN encrypting key, formats and encrypts the PIN. Use of this service requires an optional PCICC, PCIXCC, or CCA Crypto Express coprocessor. An enhanced PIN security mode on PCICC, PCIXCC, and CCA Crypto Express coprocessors, is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. See “Encrypted PIN Generate (CSNBEPG and CSNEEPG)” on page 497 for more information.

## **Encrypted PIN Translate Callable Service (CSNBPTR and CSNEPTR)**

To translate a PIN from one PIN-encrypting key to another or from one PIN block format to another or both, call the Encrypted PIN translation callable service. You must identify the input PIN-encrypting key that originally enciphers the PIN. You also need to specify the output PIN-encrypting key that you want the callable service to use to encipher the PIN. If you want to change the PIN block format, specify a different output PIN block format from the input PIN block format. An enhanced PIN security mode, on PCICC, PCIXCC, and CCA Crypto Express coprocessors, is available for formatting an encrypted PIN block into IBM 3621

format or IBM 3624 format. The enhanced security mode is also available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. See “Encrypted PIN Translate (CSNBPTR and CSNEPTR)” on page 502 for more information.

### **Encrypted PIN Verify Callable Service (CSNBPVR and CSNEPVR)**

To verify a supplied PIN, call the Encrypted PIN verify callable service. You need to specify the supplied enciphered PIN, the PIN-encrypting key that enciphers it, and other relevant data. You must also specify the PIN verification key and PIN verification algorithm. It compares the two personal identification numbers; if they are the same, it verifies the supplied PIN. See Chapter 8, “Financial Services,” on page 465 for additional information.

An enhanced PIN security mode, on PCICC, PCIXCC, and CCA Crypto Express coprocessors, is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. See “Encrypted PIN Verify (CSNBPVR and CSNEPVR)” on page 509 for more information.

### **PIN Change/Unblock Callable Service (CSNBPCU and CSNEPCU)**

To support PIN change algorithms specified in the VISA Integrated Circuit Card Specification, call the PIN change/unblock callable service. The service can be executed on z890/z990 and later machines.

An enhanced PIN security mode, on PCICC, PCIXCC, and CCA Crypto Express coprocessors, is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. See “PIN Change/Unblock (CSNBPCU and CSNEPCU)” on page 516 for more information.

### **Transaction Validation Callable Service (CSNBTRV and CSNETRV)**

To support generation and validation of American Express card security codes, call the transaction validation callable service. The service can be executed on z890/z990 and later machines.

---

## **ANSI TR-31 key block support**

A TR-31 key block is a format defined by the American National Standards Institute (ANSI) to support the interchange of keys in a secure manner with key attributes included in the exchanged data. The TR-31 key block format has a set of defined key attributes that are securely bound to the key so that they can be transported together between any two systems that both understand the TR-31 format. ICSF enables applications to convert a CCA token to a TR-31 key block for export to another party, and to convert an imported TR-31 key block to a CCA token. This enables you to securely exchange keys and their attributes with non-CCA systems.

Although there is often a one-to-one correspondence between TR-31 key attributes and the attributes defined by CCA, there are also cases where the correspondence is many-to-one or one-to-many. Because there is not always a one-to-one mapping between the key attributes defined by TR-31 and those defined by CCA, the TR-31 Export callable service and the TR-31 Import callable service provide rule array keywords that enable an application to specify the attributes to attach to the exported or imported key.

The TR-31 key block format defines a header section. The header contains metadata about the key, including its usage attributes. The header can also be extended with optional blocks, which can either have standardized content or proprietary information. Callable services are also provided for retrieving standard header or optional block information from a TR-31 key block without importing the key and for building an optional block.

The TR-31 key block support requires a z196 or IBM zEnterprise EC12 with a CEX3C or CEX4C and the Sept. 2011 or later LIC. Only DES/TDES keys can be transported in TR-31 key blocks. There is no support for transporting AES keys.

### **TR-31 Export Callable Service (CSNBT31X and CSNET31X)**

The TR-31 Export callable service converts a CCA token to TR-31 format for export to another party. Since there is not always a one-to-one mapping between the key attributes defined by TR-31 and those defined by CCA, the caller may need to specify the attributes to attach to the exported key through the rule array.

### **TR-31 Import Callable Service (CSNBT31I and CSNET31I)**

The TR-31 Import callable service converts a TR-31 key block to a CCA token. Since there is not always a one-to-one mapping between the key attributes defined by TR-31 and those defined by CCA, the caller may need to specify the attributes to attach to the imported key through the rule array.

### **TR-31 Parse Callable Service (CSNBT31P and CSNET31P)**

The TR-31 Parse callable service retrieves standard header information from a TR-31 key block without importing the key. This callable service can be used with the TR-31 Optional Data Read callable service to obtain both the standard header fields and any optional data blocks from the key block.

### **TR-31 Optional Data Read Callable Service (CSNBT31R and CSNET31R)**

A TR-31 key block can hold optional fields which are securely bound to the key block using the integrated MAC. The optional blocks may either contain information defined in the TR-31 standard, or they may contain proprietary data. A separate range of optional block identifiers is reserved for use with proprietary blocks. Applications can call the TR-31 Optional Data Read callable service to obtain lists of the optional block identifiers and optional block lengths, and to obtain the data for a particular optional block. This callable service is often used in conjunction with the TR-31 Parse Callable Service which can be used to determine the number of optional blocks in the TR-31 token.

### **TR-31 Optional Data Build Callable Service (CSNBT31O and CSNET31O)**

The TR-31 Optional Data Build callable service constructs the optional block data structure for a TR-31 key block. It builds the structure by adding one optional block with each call, until your entire set of optional blocks have been added. With each call, the application program provides a single optional block by specifying its ID, its length, and its data. Each subsequent call appends the current optional block to any pre-existing blocks.

---

## Secure Messaging

These services will assist applications in encrypting secret information such as clear keys and PIN blocks in a secure message. These services will execute within the secure boundary of the PCICC, PCIXCC, or CCA Crypto Express coprocessor.

The Secure Messaging for Keys callable service encrypts a text block, including a clear key value decrypted from an internal or external DES token.

The Secure Messaging for PINs callable service encrypts a text block, including a clear PIN block recovered from an encrypted PIN block.

---

## Trusted Key Entry (TKE) Support

The Trusted Key Entry (TKE) workstation is an optional feature. It offers an alternative to clear key entry. You can use the TKE workstation to load:

- DES master key, AES master key, PKA master keys, and operational keys in a *secure* way. CCF only supports Operational Transport and PIN keys. On the PCIXCC/CEX2C, all operational keys can be loaded with TKE V4.1 or higher. AES master key and AES operational keys can be loaded with TKE V5.3. On the CEX3C, all operational keys can be loaded with TKE 6.0 or later. On the CEX4C, all operational keys can be loaded with TKE 7.2 or later.
- DES-MK and ASYM-MK master keys on the PCICC, PCIXCC, or CCA Crypto Express coprocessors.
- AES master keys are only on z9 and z10 systems running with the Nov. 2008 or later licensed internal code (LIC).

You can load keys remotely and for multiple PCICCs, PCIXCCs, or CCA Crypto Express coprocessors. The TKE workstation eases the administration for using one cryptographic coprocessor as a production machine and as a test machine at the same time, while maintaining security and reliability.

The TKE workstation can be used for enabling/disabling access control points for callable services executed on cryptographic coprocessors. See Appendix H, "Access Control Points and Callable Services," on page 961 for additional information.

For complete details about the TKE workstation see *z/OS Cryptographic Services ICSF TKE Workstation User's Guide*.

TKE Version 4.0 or higher is required if using a PCIXCC/CEX2C.

TKE Version 6.0 or higher is required if using a CEX3C.

TKE Version 7.2 or higher is required if using a CEX4C.

On z890, z990 z9 EC, z9 BC, z10 EC and z10 BC systems running with May 2004 or higher version of Licensed Internal Code or an z9 EC, z9 BC, z10 EC and z10 BC with MCL 029 Stream J12220 or higher of Licensed Internal Code, you must enable TKE commands for each PCIXCC or CCA Crypto Express coprocessor from the Support Element. This is true for new TKE users and those upgrading from TKE V4.0 to V4.1, V4.2 or V5.x when the new LIC is installed. See *Support Element Operations Guide* and *z/OS Cryptographic Services ICSF TKE Workstation User's Guide*, SA23-2211 for more information.

---

## Utilities

ICSF provides these utilities.

### **Character/Nibble Conversion Callable Services (CSNBXBC and CSNBXCB)**

The character/nibble conversion callable services are utilities that convert a binary string to a character string and vice versa.

### **Code Conversion Callable Services (CSNBXEA and CSNBXAE)**

The code conversion callable services are utilities that convert EBCDIC data to ASCII data and vice versa.

### **X9.9 Data Editing Callable Service (CSNB9ED)**

The data editing callable service is a utility that edits an ASCII text string according to the editing rules of ANSI X9.9-4.

### **ICSF Query Algorithm Callable Service (CSFIQA)**

The callable service provides information regarding the cryptographic and hash algorithms available.

### **ICSF Query Facility Callable Service (CSFIQF)**

The callable service provides ICSF status information, as well as coprocessor information.

## Typical Sequences of ICSF Callable Services

Sample sequences in which the ICSF callable services might be called are shown in Table 7.

Table 7. Combinations of the Callable Services

Combination A (DATA keys only)	Combination B
<ol style="list-style-type: none"> <li>1. Random number generate</li> <li>2. Clear key import or multiple clear key import</li> <li>3. Encipher/decipher</li> <li>4. Data key export or key export (optional step)</li> </ol>	<ol style="list-style-type: none"> <li>1. Random number generate</li> <li>2. Secure key import or multiple secure key import</li> <li>3. Any service</li> <li>4. Data key export for DATA keys, or key export in the general case (optional step)</li> </ol>
Combination C	Combination D
<ol style="list-style-type: none"> <li>1. Key generate (OP form only)</li> <li>2. Any service</li> <li>3. Key export (optional)</li> </ol>	<ol style="list-style-type: none"> <li>1. Key generate (OPEX form)</li> <li>2. Any service</li> </ol>
Combination E	Combination F
<ol style="list-style-type: none"> <li>1. Key generate (IM form only)</li> <li>2. Key import</li> <li>3. Any service</li> <li>4. Key export (optional)</li> </ol>	<ol style="list-style-type: none"> <li>1. Key generate (IMEX form)</li> <li>2. Key import</li> <li>3. Any service</li> </ol>
Combination G	Combination H
<ol style="list-style-type: none"> <li>1. Key generate</li> <li>2. Key record create</li> <li>3. Key record write</li> <li>4. Any service (passing label of the key just generated)</li> </ol>	<ol style="list-style-type: none"> <li>1. Key import</li> <li>2. Key record create</li> <li>3. Key record write</li> <li>4. Any service (passing label of the key just generated)</li> </ol>
Combination I	
<ol style="list-style-type: none"> <li>1. Key token build to create key token skeleton</li> <li>2. Key generate to OP form of AKEK using key token skeleton</li> <li>3. Use AKEK in any ANSI X9.17 service</li> </ol>	
<b>Notes:</b> <ol style="list-style-type: none"> <li>1. An example of “any service” is CSNBENC.</li> <li>2. These combinations exclude services that can be used on their own; for example, key export or encode, or using key generate to generate an exportable key.</li> <li>3. These combinations do not show key communication, or the transmission of any output from an ICSF callable service.</li> <li>4. Combination I is not available on the IBM @server zSeries 990.</li> </ol>	

The key forms are described in “Key Generate (CSNBKGN and CSNEKGN)” on page 142.



---

## Key Forms and Types Used in the Key Generate Callable Service

The key generate callable service is the most complex of all the ICSF callable services. This topic provides examples of the key forms and key types used in the key generate callable service.

### Generating an Operational Key

To generate an operational key, choose one of these methods:

- **For operational keys**, call the key generate callable service (CSNBKGN). Table 36 on page 151 and Table 37 on page 151 show the key type and key form combinations for a single key and for a key pair.
- **For operational keys**, call the random number generate callable service (CSNBRNG) and specify the *form* parameter as RANDOM. Specify ODD parity for a random number you intend to use as a key. Then pass the generated value to the secure key import callable service (CSNBSKI) with a required key type. The required key type is now in operational form.

This method requires a cryptographic unit to be in special secure mode. For more information about special secure mode, see “Special Secure Mode” on page 10.

- **For data-encrypting keys**, call the random number generate callable service (CSNBRNG) and specify the *form* parameter as ODD. Then pass the generated value to the clear key import callable service (CSNBCKI) or the multiple clear key import callable service (CSNBCKM). The DATA key type is now in operational form.

You cannot generate a PIN verification (PINVER) key in operational form because the originator of the PIN generation (PINGEN) key generates the PINVER key in exportable form, which is sent to you to be imported.

### Generating an Importable Key

To generate an importable key form, call the key generate callable service (CSNBKGN).

If you want a DATA, MAC, PINGEN, DATAM, or DATAC key type in importable form, obtain it directly by generating a single key. If you want any other key type in importable form, request a key pair where either the first or second key type is importable (IM). Discard the generated key form that you do not need.

### Generating an Exportable Key

To generate an exportable key form, call the key generate callable service (CSNBKGN).

If you want a DATA, MAC, PINGEN, DATAM, or DATAC key type in exportable form, obtain it directly by generating a single key. If you want any other key type in exportable form, request a key pair where either the first or second key type is exportable (EX). Discard the generated key form that you do not need.

### Examples of Single-Length Keys in One Form Only

Key Form	Key 1
-------------	----------

OP	DATA	Encipher or decipher data. Use data key export or key export to send encrypted key to another cryptographic partner. Then communicate the ciphertext.
----	------	---

OP	MAC	MAC generate. Because no MACVER key exists, there is no secure communication of the MAC with another cryptographic partner.
IM	DATA	Key Import, and then encipher or decipher. Then key export to communicate ciphertext and key with another cryptographic partner.
EX	DATA	You can send this key to a cryptographic partner, but you can do nothing with it directly. Use it for the key distribution service. The partner could then use key import to get it in operational form, and use it as in OP DATA above.

## Examples of OPIM Single-Length, Double-Length, and Triple-Length Keys in Two Forms

The first two letters of the key form indicate the form that key type 1 parameter is in, and the second two letters indicate the form that key type 2 parameter is in.

Key Form	Type 1	Type 2	
OPIM	DATA	DATA	Use the OP form in encipher. Use key export with the OP form to communicate ciphertext and key with another cryptographic partner. Use key import at a later time to use encipher or decipher with the same key again.
OPIM	MAC	MAC	Single-length MAC generation key. Use the OP form in MAC generation. You have no corresponding MACVER key, but you can call the MAC verification service with the MAC key directly. Use the key import callable service and then compute the MAC again using the MAC verification callable service, which compares the MAC it generates with the MAC supplied with the message and issues a return code indicating whether they compare.

## Examples of OPEX Single-Length, Double-Length, and Triple-Length Keys in Two Forms

Key Form	Type 1	Type 2	
OPEX	DATA	DATA	Use the OP form in encipher. Send the EX form and the ciphertext to another cryptographic partner.
OPEX	MAC	MAC	Single-length MAC generation key. Use the OP form in both MAC generation and MAC verification. Send the EX form to a cryptographic partner to be used in the MAC generation or MAC verification services.
OPEX	MAC	MACVER	Single-length MAC generation and MAC verification keys. Use the OP form in MAC generation. Send the EX form to a cryptographic partner where it will be put into key import, and then MAC verification, with the message and MAC that you have also transmitted.
OPEX	PINGEN	PINVER	Use the OP form in Clear PIN generate. Send the EX form to a cryptographic partner where it is put into key import, and then Encrypted PIN verify, along with an IPINENC key.
OPEX	IMPORTER	EXPORTER	Use the OP form in key import, key generate, or secure key import. Send the EX form to a cryptographic partner where it is used in key export, data key export, or key generate, or put in the CKDS.
OPEX	EXPORTER	IMPORTER	Use the OP form in key export, data key export,

or key generate. Send the EX form to a cryptographic partner where it is put into the CKDS or used in key import, key generate or secure key import.

When you and your partner have the OPEX IMPORTER EXPORTER, OPEX EXPORTER IMPORTER pairs of keys in “Examples of OPEX Single-Length, Double-Length, and Triple-Length Keys in Two Forms” on page 68 installed, you can start key and data exchange.

## Examples of IMEX Single-Length and Double-Length Keys in Two Forms

Key Form	Type 1	Type 2	
IMEX	DATA	DATA	Use the key import callable service to import IM form and use the OP form in encipher. Send the EX form to a cryptographic partner.
IMEX	MAC	MACVER	Use the key import callable service to import the IM form and use the OP form in MAC generate. Send the EX form to a cryptographic partner who can verify the MAC.
IMEX	IMPORTER	EXPORTER	Use the key import callable service to import the IM form and send the EX form to a cryptographic partner. This establishes a new IMPORTER/EXPORTER key between you and your partner.
IMEX	PINGEN	PINVER	Use the key import callable service to import the IM form and send the EX form to a cryptographic partner. This establishes a new PINGEN/PINVER key between you and your partner.

## Examples of EXEX Single-Length and Double-Length Keys in Two Forms

For the keys shown in this list, you are providing key distribution services for other nodes in your network, or other cryptographic partners. Neither key type can be used in your installation.

Key Form	Type 1	Type 2	
EXEX	DATA	DATA	Send the first EX form to a cryptographic partner with the corresponding IMPORTER and send the second EX form to another cryptographic partner with the corresponding IMPORTER. This exchange establishes a key between two partners.
EXEX	MAC	MACVER	
EXEX	IMPORTER	EXPORTER	
EXEX	OPINENC	IPINENC	

## Generating AKEKs

**Restriction:** AKEKs are only supported on the IBM @server zSeries 800 and the IBM @server zSeries 900.

AKEKs are bidirectional and are OP-form-only keys that can be used in both import and export. Prior to using the key generate callable service to create an AKEK, you need to use the key token build callable service to create a key token for receiving the AKEK. The steps involved in this process are:

1. Use the key token build callable service with these parameter values:

Parameter	Value
-----------	-------

- |                   |                                      |
|-------------------|--------------------------------------|
| <b>Key_type</b>   | AKEK                                 |
| <b>Rule_array</b> | INTERNAL NO-KEY {SINGLE or DOUBLE-O} |
2. Use the key generate callable service with these parameter values:

<b>Parameter</b>	<b>Value</b>
<b>Key_form</b>	OP
<b>Key_type_1</b>	TOKEN
<b>Generated_key_identifier_1</b>	The skeleton key token created in step 1

## Using the Ciphertext Translate and Ciphertext Translate2 Callable Service

**Restriction:** The ciphertext translate callable service does not work in CDMF-only systems (see “System Encryption Algorithm” on page 52).

**Restriction:** The ciphertext translate2 callable service is only available on the IBM zEnterprise EC12 and later servers.

This topic describes a scenario using the encipher, ciphertext translate, and decipher callable services with four network nodes: A, B, C, and D. You want to send data from your network node A to a destination node D. You cannot communicate directly with node D, and nodes B and C are situated between you. You do not want nodes B and C to decipher your data.

At node A, you use the Encipher callable service. Node D uses the Decipher callable service.

Node B and C will use the ciphertext translate callable service. Consider the keys that are needed to support this process:

1. At your node, generate one key in two forms: OPEX DATA DATAXLAT
2. Send the exportable DATAXLAT key to node B.
3. Node B and C need to share a DATAXLAT key, so generate a **different key** in two forms: EXEX DATAXLAT DATAXLAT.
4. Send the first exportable DATAXLAT key to node B.
5. Send the second exportable DATAXLAT key to node C.
6. Node C and node D need to share a DATAXLAT key and a DATA key. Node D can generate one key in two forms: OPEX DATA DATAXLAT.
7. Node D sends the exportable DATAXLAT key to node C.

The communication process is shown as:

Node:	A	B	C	D
Callable				
Service:	Encipher	Ciphertext Translate	Ciphertext Translate	Decipher
Keys:	DATA	DATAXLAT	DATAXLAT	DATAXLAT
Key Pairs:	___ = ___	___ = ___	___ = ___	

**Note:** Where a DATA type key is specified in the preceding diagram, a CIPHER key can also be used. Where a DATAXLAT type key is specified, a CIPHER TRANSLATE type key can be used.

Therefore, you need three keys, each in two different forms. You can generate two of the keys at node A, and node D can generate the third key. Note that the key used in the decipher callable service at node D is **not** the same key used in the encipher callable service at node A.

## Summary of Callable Services

Table 8 lists the callable services described in this publication, and their corresponding verbs. The figure also references the topic that describes the callable service.

Table 8. Summary of ICSF Callable Services

Verb	Service Name	Function
<b>Chapter 5, “Managing Symmetric Cryptographic Keys”</b>		
CSNBCKI CSNECKI	Clear key import	Imports an 8-byte clear DATA key, enciphers it under the master key, and places the result into an internal key token. CSNBCKI converts the clear key into operational form as a DATA key.
CSNBCVG CSNECVG	Control vector generate	Builds a control vector from keywords specified by the <i>key_type</i> and <i>rule_array</i> parameters.
CSNBCVT CSNECVT	Control vector translate	Changes the control vector used to encipher an external key.
CSNBCVE CSNECVE	Cryptographic variable encipher	Uses a CVARENC key to encrypt plaintext by using the Cipher Block Chaining (CBC) method. The plaintext must be a multiple of eight bytes in length.
CSNBDKX CSNEDKX	Data key export	Converts a DATA key from operational form into exportable form.
CSNBDKM CSNEDKM	Data key import	Imports an encrypted source DES single- or double-length DATA key and creates or updates a target internal key token with the master key enciphered source key.
CSNBDKG CSNEDKG	Diversified key generate	Generates a key based upon the key-generating key, the processing method, and the parameter data that is supplied.
CSNBEDH CSNEEDH	ECC Diffie-Hellman	Creates symmetric key material from a pair of ECC keys using the Elliptic Curve Diffie-Hellman protocol and the static unified model key agreement scheme or “Z” data (the “secret” material output from D-H process).
CSNBKEX CSNEKEX	Key export	Converts any key from operational form into exportable form. (However, this service does not export a key that was marked non-exportable when it was imported.)
CSNBKGN CSNEKGN	Key generate	Generates a 64-bit, 128-bit, or 192-bit odd parity key, or a pair of keys; and returns them in encrypted forms (operational, exportable, or importable). CSNBKGN does not produce keys in plaintext.

Table 8. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSNBKGN2 CSNEKGN2	Key generate2	Generates a variable-length HMAC or AES key or a pair of keys; and returns them in encrypted forms (operational, exportable, or importable).
CSNBKIM CSNEKIM	Key import	Converts any key from importable form into operational form.
CSNBKPI CSNEKPI	Key part import	Combines the clear key parts of any key type and returns the combined key value in an internal key token or an update to the CKDS.
CSNBKPI2 CSNEKPI2	Key part import2	Combines the clear key parts of an HMAC or AES key and returns the combined key value in an internal key token or an update to the CKDS.
CSNBKYT CSNEKYT CSNBKYTX CSNEKYTX	Key test	Generates or verifies (depending on keywords in the rule array) a secure verification pattern for keys. CSNBKYT and CSNEKYT require the tested key to be in the clear or encrypted under the master key. CSNBKYTX and CSNEKYTX also allow the tested key to be encrypted under a key-encrypting key.
CSNBKYT2 CSNEKYT2	Key test2	Generates or verifies (depending on keywords in the rule array) a secure verification pattern for keys. CSNBKYT2 and CSNEKYT2 allow the tested key to be in the clear or encrypted under the master key or a key-encrypting key.
CSNBKTB CSNEKTB	Key token build	Builds an internal or external token from the supplied parameters. You can use this callable service to build an internal token for an AKEK for input to the key generate and key part import callable services. You can also use this service to build CCA key tokens for all key types ICSF supports. You can also use this service to build CCA key tokens for all key types ICSF supports.
CSNBKTB2 CSNEKTB2	Key token build2	Builds an internal clear key or skeleton token from the supplied parameters. You can use this callable service to build an internal clear key token for any key type for input to the key test2 callable service. You can use this callable service to build a skeleton token for input to the key generate2 and key part import2 callable services.
CSNBKTR CSNEKTR	Key translate	Uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment.
CSNBKTR2 CSNEKTR2	Key translate2	Uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment.

Table 8. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSNBCKM CSNECKM	Multiple clear key import	Imports a single-, double-, or triple-length clear DATA key, enciphers it under the master key, and places the result into an internal key token. CSNBCKM converts the clear key into operational form as a DATA key.
CSNBCKM CSNESKM	Multiple secure key import	Enciphers a single-, double-, or triple-length clear key under the master key or an input importer key, and places the result into an internal or external key token as any key type. Triple-length keys can only be imported as DATA keys.  This service executes only in special secure mode.
CSNDPKD CSNFPKD	PKA decrypt	Uses an RSA private key to decrypt the RSA-encrypted key value and return the clear key value to the application.
CSNDPKE CSNFPKE	PKA encrypt	Encrypts a supplied clear key value under an RSA public key.
CSNBPEX CSNEPEX	Prohibit export	Modifies an operational key so that it cannot be exported.
CSNBPEXX CSNEPEXX	Prohibit export extended	Changes the external token of a key in exportable form so that it can be imported at the receiver node but not exported from that node.
CSNBRKA CSNERKA	Restrict Key Attribute	Modifies an operational variable-length key so that it cannot be exported.
CSNBRNG CSNERNG CSNBRNGL CSNERNGL	Random number generate	Generates an 8-byte random number or a random number with a user-specified length. The output can be specified in three forms of parity: RANDOM, ODD, and EVEN.
CSNDRKX CSNFRKX	Remote key export	Generates or exports DES keys for local use and for distribution to an ATM or other remote device. RKX uses a special structure to hold encrypted symmetric keys in a way that binds them to the trusted block and allows sequences of RKX calls to be bound together as if they were an atomic operation.
CSNBSKI CSNESKI	Secure key import	Enciphers a clear key under the master key, and places the result into an internal or external key token as any key type.  This service executes only in special secure mode.
CSNBSKI2 CSNESKI2	Secure key import2	Enciphers a variable-length clear HMAC or AES key under the master key and places the result into an internal key token.  This service executes only in special secure mode.



Table 8. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSNDSYX CSNFSYX	Symmetric key export	Transfers an application-supplied symmetric key from encryption under the host master key to encryption under an application-supplied RSA public key or AES EXPORTER key. The application-supplied key must be an internal key token or the label in the CKDS of a DES DATA, AES DATA, or variable-length symmetric key token.
CSNDSYG CSNFSYG	Symmetric key generate	Generates a symmetric DATA key and returns the key in two forms: enciphered under the DES master key or KEK and under a PKA public key.
CSNDSYI CSNFSYI	Symmetric key import	Imports a symmetric key enciphered under an RSA public key into operational form enciphered under a host master key.
CSNDSYI2 CSNFSYI2	Symmetric key import2	Imports a symmetric key enciphered under an RSA public key or AES EXPORTER key into operational form enciphered under a host master key.
CSNBTCK CSNETCK	Transform CDMF key	Changes a CDMF DATA key in an internal or external token to a transformed shortened DES key.
CSNDTBC CSNETBC	Trusted block create	Creates a trusted block in a two step process. The block will be in external form, encrypted under an IMP-PKA transport key. This means that the MAC key contained within the trusted block will be encrypted under the IMP-PKA key.
CSNBT31X CSNET31X	TR-31 Export	Converts a CCA token to TR-31 format for export to another party.
CSNBT31I CSNET31I	TR-31 Import	Converts a TR-31 key block to a CCA token.
CSNBT31P CSNET31P	TR-31 Parse	Retrieves standard header information from a TR-31 key block without importing the key.
CSNBT31R CSNET31R	TR-31 Optional Data Read	Obtains lists of the optional block identifiers and optional block lengths, and obtains the data for a particular optional block.
CSNBT31O CSNET31O	TR-31 Optional Data Build	Constructs the optional block data structure for a TR-31 key block.
CSFUDK CSFUDK6	Unique Key Derive	Generates single-length or double-length MAC keys, or updates an existing user derived key.

Table 8. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSNBUKD CSNEUKD	User Derived Key	Derives a key using a base derivation key and derivation data. The following key types can be derived: <ul style="list-style-type: none"> <li>• CIPHER</li> <li>• ENCIPHER</li> <li>• DECIPHER</li> <li>• MAC</li> <li>• MACVER</li> <li>• IPINENC</li> <li>• OPINENC</li> <li>• DATA token containing a PIN Key</li> </ul>
<b>Chapter 6, "Protecting Data"</b>		
CSNBCTT CSNECTT CSNBCTT1 CSNECTT1	Ciphertext translate	Translates the user-supplied ciphertext from one key and enciphers the ciphertext to another key. (This is for DES encryption only.)  CSNBCTT and CSNECTT require the ciphertext to reside in the caller's primary address space.  CSNBCTT1 and CSNECTT1 allow the ciphertext to reside in the caller's primary address space or in a z/OS data space.
CSNBCTT2 CSNBCTT3 CSNECTT2 CSNECTT3	Ciphertext translate2	Translates the user-supplied ciphertext from one key and enciphers the ciphertext to another key. Supports both AES and DES algorithms.  CSNBCTT2 and CSNECTT2 require the ciphertext to reside in the caller's primary address space.  CSNBCTT3 and CSNECTT3 allow the ciphertext to reside in the caller's primary address space or in a z/OS data space.
CSNBDEC CSNEDEC CSNBDEC1 CSNEDEC1	Decipher	Deciphers data using either the CDMF or the cipher block chaining mode of the DES. (The method depends on the token marking or keyword specification.) The result is called plaintext.  CSNBDEC and CSNEDEC require the plaintext and ciphertext to reside in the caller's primary address space.  CSNBDEC1 and CSNEDEC1 allow the plaintext and ciphertext to reside in the caller's primary address space or in a z/OS data space.
CSNBDCO CSNEDCO	Decode	Decodes an 8-byte string of data using the electronic code book mode of the DES. (This is for DES encryption only.)

Table 8. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSNBENC CSNEENC CSNBENC1 CSNEENC1	Encipher	<p>Enciphers data using either the CDMF or the cipher block chaining mode of the DES. (The method depends on the token marking or keyword specification.) The result is called ciphertext.</p> <p>CSNBENC and CSNEENC require the plaintext and ciphertext to reside in the caller's primary address space.</p> <p>CSNBENC1 and CSNEENC1 allow the plaintext and ciphertext to reside in the caller's primary address space or in a z/OS data space.</p>
CSNBECO CSNEECO	Encode	<p>Encodes an 8-byte string of data using the electronic code book mode of the DES. (This is for DES encryption only.)</p>
CSNBSAD CSNESAD CSNBSAD1 CSNESAD1	Symmetric algorithm decipher	<p>Deciphers data using the AES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes.</p> <p>CSNBSAD and CSNESAD require the plaintext and ciphertext to reside in the caller's primary address space.</p> <p>CSNBSAD1 and CSNESAD1 allows the plaintext and ciphertext to reside in the caller's primary address space or in a z/OS data space.</p>
CSNBSAE CSNESAE CSNBSAE1 CSNESAE1	Symmetric algorithm encipher	<p>Enciphers data using the AES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes.</p> <p>CSNBSAE and CSNESAE require the plaintext and ciphertext to reside in the caller's primary address space.</p> <p>CSNBSAE1 and CSNESAE1 allows the plaintext and ciphertext to reside in the caller's primary address space or in a z/OS data space.</p>
CSNBSYD CSNBSYD1 CSNESYD CSNESYD1	Symmetric key decipher	<p>Deciphers data using the AES or DES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes. Only clear keys are supported.</p> <p>CSNBSYD and CSNESYD require the plaintext and ciphertext to reside in the caller's primary address space.</p> <p>CSNBSYD1 and CSNESYD1 allow the plaintext and ciphertext to reside in the caller's primary address space or in a z/OS data space.</p>

Table 8. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSNBSYE CSNBSYE1 CSNESYE CSNESYE1	Symmetric key encipher	<p>Enciphers data using the AES or DES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes. Only clear keys are supported.</p> <p>CSNBSYE and CSNESYE require the plaintext and ciphertext to reside in the caller's primary address space.</p> <p>CSNBSYE1 and CSNESYE1 allows the plaintext and ciphertext to reside in the caller's primary address space or in a z/OS data space.</p>
<b>Chapter 7, "Verifying Data Integrity and Authenticating Messages"</b>		
CSNBHMG CSNEHMG CSNBHMG1 CSNEHMG1	HMAC generation	<p>Generates message authentication code (MAC) for a text string that the application program supplies. The MAC is computed using the FIPS-198 Keyed-Hash Message Authentication Code algorithm.</p> <p>CSNBHMG and CSNEHMG require data to reside in the caller's primary address space.</p> <p>CSNBHMG1 and CSNEHMG1 allow data to reside in the caller's primary address space or in a z/OS data space.</p>
CSNBHMGV CSNEHMGV CSNBHMGV1 CSNEHMGV1	HMAC verification	<p>Verifies message authentication code (MAC) for a text string that the application program supplies. The MAC is computed using the FIPS-198 Keyed-Hash Message Authentication Code algorithm.</p> <p>CSNBHMGV and CSNEHMGV requires data to reside in the caller's primary address space.</p> <p>CSNBHMGV1 and CSNEHMGV1 allows data to reside in the caller's primary address space or in a z/OS data space.</p>
CSNBMGN CSNEMGN CSNBMGN1 CSNEMGN1	MAC generate	<p>Generates a 4-, 6-, or 8-byte message authentication code (MAC) for a text string that the application program supplies. The MAC is computed using the ANSI X9.9-1 algorithm, ANSI X9.19 optional double key algorithm the EMV padding rules or the ISO 16609 TDES algorithm.</p> <p>CSNBMGN and CSNEMGN require data to reside in the caller's primary address space.</p> <p>CSNBMGN1 and CSNEMGN1 allow data to reside in the caller's primary address space or in a z/OS data space.</p>

Table 8. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSNBMVR CSNEMVR CSNBMVR1 CSNEMVR1	MAC verify	<p>Verifies a 4-, 6-, or 8-byte message authentication code (MAC) for a text string that the application program supplies. The MAC is computed using the ANSI X9.9-1 algorithm, ANSI X9.19 optional double key algorithm, the EMV padding rules or the ISO 16609 TDES algorithm.</p> <p>CSNBMVR and CSNEMVR require data to reside in the caller's primary address space.</p> <p>CSNBMVR1 and CSNEMVR1 allow data to reside in the caller's primary address space or in a z/OS data space.</p>
CSNBMDG CSNEMDG CSNBMDG1 CSNEMDG1	MDC generate	<p>Generates a 128-bit modification detection code (MDC) for a text string that the application program supplies.</p> <p>CSNBMDG and CSNEMDG require data to reside in the caller's primary address space.</p> <p>CSNBMDG1 and CSNEMDG1 allow data to reside in the caller's primary address space or in a z/OS data space.</p>
CSNBOWH CSNEOWH CSNBOWH1 CSNEOWH1	One way hash generate	Generates a one-way hash on specified text.
CSNBSMG, CSNESMG CSNBSMG1 CSNESMG1	Symmetric MAC Generate	<p>Use the symmetric MAC generate callable service to generate a 96- or 128-bit message authentication code (MAC) for an application-supplied text string using a clear AES key.</p> <p>CSNBSMG1 allows data to reside in the caller's primary address space or in a z/OS data space.</p>
CSNBSMV, CSNESMV CSNBSMV1 CSNESMV1	Symmetric MAC Verify	<p>Use the symmetric MAC verify callable service to verify a 96- or 128-bit message authentication code (MAC) for an application-supplied text string using a clear AES key.</p> <p>CSNBSMV1 allows data to reside in the caller's primary address space or in a z/OS data space.</p>
<b>Chapter 8, "Financial Services"</b>		
CSNBCPE CSNECPE	Clear PIN encrypt	Formats a PIN into a PIN block format and encrypts the results.

Table 8. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSNBPGN CSNEPGN	Clear PIN generate	Generates a clear personal identification number (PIN), a PIN verification value (PVV), or an offset using one of these algorithms: IBM 3624 (IBM-PIN or IBM-PINO) IBM German Bank Pool (GBP-PIN or GBP-PINO) VISA PIN validation value (VISA-PVV) Interbank PIN (INBK-PIN)  This service executes only in special secure mode.
CSNBPCPA CSNECPA	Clear PIN generate alternate	Generates a clear VISA PIN validation value (PVV) from an input encrypted PIN block. The PIN block may have been encrypted under either an input or output PIN encrypting key. The IBM-PINO algorithm is supported to produce a 3624 offset from a customer selected encrypted PIN.
CSNBCKC CSNECKC	CVV Key Combine	Combines two single-length CCA internal key tokens into 1 double-length CCA key token containing a CVVKEY-A key type.
CSNBEPG CSNEEPG	Encrypted PIN generate	Generates and formats a PIN and encrypts the PIN block.
CSNBPTR CSNEPTR	Encrypted PIN translate	Reenciphers a PIN block from one PIN-encrypting key to another and, optionally, changes the PIN block format. UKPT keywords are supported.
CSNBPVR CSNEPVR	Encrypted PIN verify	Verifies a supplied PIN using one of these algorithms: IBM 3624 (IBM-PIN or IBM-PINO) IBM German Bank Pool (GBP-PIN or GBP-PINO) VISA PIN validation value (VISA-PVV) Interbank PIN (INBK-PIN)  UKPT keywords are supported.
CSNBPCU CSNEPCU	PIN Change/Unblock	Supports the PIN change algorithms specified in the VISA Integrated Circuit Card Specification; only available on a z890 or Requires May 2004 or later version of Licensed Internal Code (LIC).
CSNBSKY CSNESKY	Secure messaging for keys	Encrypts a text block, including a clear key value decrypted from an internal or external DES token.
CSNBSPN CSNESPN	Secure messaging for PINs	Encrypts a text block, including a clear PIN block recovered from an encrypted PIN block.
CSNDSBC CSNFSBC	SET block compose	Composes the RSA-OAEP block and the DES-encrypted block in support of the SET protocol.
CSNDSBD CSNFSBD	SET block decompose	Decomposes the RSA-OAEP block and the DES-encrypted block to provide unencrypted data back to the caller.

Table 8. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSNBTRV CSNETRV	Transaction Validation	Supports the generation and validation of American Express card security codes; only available on a z890 or Requires May 2004 or later version of Licensed Internal Code (LIC).
CSNBCSG CSNECSG	VISA CVV service generate	Generates a VISA Card Verification Value (CVV) or a MasterCard Card Verification Code (CVC).
CSNBCSV CSNECSV	VISA CVV service verify	Verifies a VISA Card Verification Value (CVV) or a MasterCard Card Verification Code (CVC).
<b>Chapter 11, "Key Data Set Management"</b>		
CSNBKRC CSNEKRC	CKDS key record create	Adds a key record containing a key token set to binary zeros to both the in-storage and DASD copies of the CKDS.
CSNBKRC2 CSNEKRC2	CKDS key record create2	Adds a key record containing a key token to both the in-storage and DASD copies of the CKDS.
CSNBKRD CSNEKRD	CKDS key record delete	Deletes a key record from both the in-storage and DASD copies of the CKDS.
CSNBKRR CSNEKRR	CKDS key record read	Copies an internal key token from the in-storage copy of the CKDS to application storage.
CSNBKRR2 CSNEKRR2	CKDS key record read2	Copies an internal key token from the in-storage copy of the CKDS to application storage.
CSNBKRW CSNEKRW	CKDS key record write	Writes an internal key token to the CKDS record specified in the key label parameter. Updates both the in-storage and DASD copies of the CKDS currently in use.
CKDSCSNBKRW2 CSNEKRW2	CKDS key record write2	Writes an internal key token to the CKDS record specified in the key label parameter. Updates both the in-storage and DASD copies of the CKDS currently in use.
CSFCRC CSFCRC6	Coordinated KDS Administration	Performs a CKDS refresh or CKDS reencipher and change master key operation while allowing applications to update the CKDS. In a sysplex environment, this callable service performs a coordinated sysplex-wide refresh or change master key operation from a single ICSF instance.
<b>Chapter 12, "Utilities"</b>		
CSNBXBC or CSNBXCB	Character/nibble conversion	Converts a binary string to a character string or vice versa.
CSNBXEA or CSNBXAE	Code conversion	Converts EBCDIC data to ASCII data or vice versa.
CSFIQA CSFIQA6	ICSF Query Algorithm	Use this utility to retrieve information about the cryptographic and hash algorithms available. You can control the amount of data that is returned by passing in different <i>rule_array</i> keywords.



Table 8. Summary of ICSF Callable Services (continued)

Verb	Service Name	Function
CSFIQF CSFIQF6	ICSF Query Service	Provides ICSF status, as well as coprocessor information.
CSNB9ED	X9.9 data editing	Edits an ASCII text string according to the editing rules of ANSI X9.9-4.
<b>Chapter 13, “Trusted Key Entry Workstation Interfaces”</b>		
CSFPCI	PCI interface	Puts a request to a specific coprocessor queue and removes the corresponding response when complete. Only the Trusted Key Entry (TKE) workstation uses this service.
CSFPKSC	PKSC interface	Puts a request to a specific cryptographic module and removes the corresponding response when complete. Only the Trusted Key Entry (TKE) workstation uses this service.
<b>Chapter 14, “Managing Keys According to the ANSI X9.17 Standard”</b>		
CSNAEGN CSNGEGN	ANSI X9.17 EDC generate	Generates an ANSI X9.17 error detection code on an arbitrary length string using the special MAC key (x'0123456789ABCDEF').
CSNAKEX CSNGKEX	ANSI X9.17 key export	Uses the ANSI X9.17 protocol to export a DATA key or a pair of DATA keys with or without an AKEK. Supports the export of a CCA IMPORTER or EXPORTER KEK. Converts a single DATA key or combines two DATA keys into a single MAC key.
CSNAKIM CSNGKIM	ANSI X9.17 key import	Uses the ANSI X9.17 protocol to import a DATA key or a pair of DATA keys with or without an AKEK. Supports the import of a CCA IMPORTER or EXPORTER KEK. Converts a single DATA key or combines two DATA keys into a single MAC key.
CSNAKTR CSNGKTR	ANSI X9.17 key translate	Uses the ANSI X9.17 protocol to translate, in a single service call, either one or two DATA keys or a single KEK from encryption under one AKEK to encryption under another AKEK. Converts a single DATA key or combines two DATA keys into a single MAC key.
CSNATKN CSNGTKN	ANSI X9.17 transport key partial notarize	Permits the preprocessing of an AKEK with origin and destination identifiers to create a partially notarized AKEK.



---

## Chapter 3. Introducing PKA Cryptography and Using PKA Callable Services

The preceding topic focused on DES cryptography or secret-key cryptography. This is symmetric—senders and receivers use the same key (which must be exchanged securely in advance) to encipher and decipher data.

Public key cryptography does not require exchanging a secret key. It is asymmetric—the sender and receiver each have a pair of keys, a public key and a different but corresponding private key.

You can use PKA support to exchange symmetric secret keys securely and to compute digital signatures for authenticating messages to users. You can also use public key cryptography in support of secure electronic transactions over open networks, using SET protocols.

---

### PKA Key Algorithms

Public key cryptography uses a key pair consisting of a public key and a private key. The PKA public key uses one of the following algorithms:

- **Rivest-Shamir-Adleman (RSA)**

The RSA algorithm is the most widely used and accepted of the public key algorithms. It uses three quantities to encrypt and decrypt text: a public exponent (PU), a private exponent (PR), and a modulus (M). Given these three and some cleartext data, the algorithm generates ciphertext as follows:

$$\text{ciphertext} = \text{cleartext}^{\text{PU}} \pmod{M}$$

Similarly, this operation recovers cleartext from ciphertext:

$$\text{cleartext} = \text{ciphertext}^{\text{PR}} \pmod{M}$$

An RSA key consists of an exponent and a modulus. The private exponent must be secret, but the public exponent and modulus need not be secret.

- **Digital Signature Standard (DSS)**

The U.S. National Institute of Standards and Technology (NIST) defines DSS in Federal Information Processing Standard (FIPS) Publication 186.

- **Elliptic Curve Digital Signature Algorithm (ECDSA)**

The ECDSA algorithm uses elliptic curve cryptography (an encryption system based on the properties of elliptic curves) to provide a variant of the Digital Signature Algorithm.

---

### PKA Master Keys

PKA master keys protect private keys.

- On the Cryptographic Coprocessor Feature, there are two PKA master keys: the Signature Master Key (SMK) and the RSA Key Management Master Key (KMMK). The SMK protects PKA private keys used only in digital signature services. The KMMK protects PKA private keys used in digital signature services and in the DES DATA key distribution functions.

- On the PCI Cryptographic Coprocessor, PKA keys are protected by the Asymmetric-Keys Master Key (ASYM-MK). The ASYM-MK is a triple-length DES key used to protect PKA keys.

In order for the PCI Cryptographic Coprocessor to function, the hash pattern of the ASYM-MK must match the hash pattern of the SMK on the Cryptographic Coprocessor Feature. The ICSF administrator installs the PKA master keys on the Cryptographic Coprocessor Feature and the ASYM-MK on the PCI Cryptographic Coprocessor by using either the pass phrase initialization routine, the Clear Master Key Entry panels, or the optional Trusted Key Entry (TKE) workstation.

Prior to PKA services being enabled on the PCI Cryptographic Coprocessor, these conditions must be met:

- The Symmetric-Keys Master Key (SYM-MK) must be installed on the PCI Cryptographic Coprocessor. It must match the Cryptographic Coprocessor Feature DES master key and match the master key that the CKDS was enciphered with.
- The PKA master keys (SMK and KMMK) on the Cryptographic Coprocessor Feature must be installed and valid.
- The ASYM-MK PKA master key on the PCI Cryptographic Coprocessor must be installed and valid.
- The hash pattern of the ASYM-MK on the PCI Cryptographic Coprocessor must match the hash pattern of the SMK on the Cryptographic Coprocessor Feature.
- The PKDS must be initialized with the PKA master keys installed on the Cryptographic Coprocessor Feature.

- On the PCIXCC and CCA Crypto Express coprocessors, PKA keys are protected by the Asymmetric-Keys Master Key (ASYM-MK). The ASYM-MK is a triple-length DES key used to protect PKA private keys. On the PCIXCC and CCA Crypto Express coprocessors, the ASYM-MK protects RSA private keys. On the z196 or IBM zEnterprise EC12 with a CEX3C or CEX4C, there are two PKA master keys: RSA and ECC. The RSA master key is the same as the ASYM-MK. The ECC master key is a 256-bit AES key used to protect ECC private keys.

- In order for PKA services to function on the PCIXCC and CCA Crypto Express coprocessors, the Asymmetric-Keys/RSA and/or ECC master keys must be installed. The ICSF administrator installs the master keys on the PCIXCC and CCA Crypto Express coprocessors by using either the pass phrase initialization routine, the Clear Master Key Entry panels, or the optional Trusted Key Entry (TKE) workstation.

Prior to PKA services being enabled on the PCIXCC or CCA Crypto Express coprocessor, these conditions must be met:

- The ASYM-MK/RSA and/or ECC master keys on the PCIXCC or CCA Crypto Express coprocessor must be installed.
- The PKDS must be initialized with the ASYM-MK/RSA and/or ECC master keys installed on the PCIXCC or CCA Crypto Express coprocessor.

## Operational private keys

RSA and DSS operational private keys are protected under two layers of DES encryption. They are encrypted under an Object Protection Key (OPK) that in turn is encrypted under the SMK/ASYM-MK/RSA or KMMK. ECC operational private keys are protected under two layers of AES encryption. They are encrypted under an AES OPK that in turn is encrypted under the ECC master key. You dynamically generate the OPK for each private key at import time or when the private key is

generated on a PCICC, PCIXCC, or CCA Crypto Express coprocessor. ICSF provides a public key data set (PKDS) for the storage of application PKA keys.

On systems with the Cryptographic Coprocessor Feature, the PKA master keys can't be changed dynamically. The PKA callable services control must be disabled when the master keys are changed. Systems with the PCI Cryptographic Coprocessor can change the SMK/ASYM-MK by loading the new master key and using the PKDS Reencipher utility to reencipher private keys encrypted to the new master key. Private tokens encrypted under the KMMK will only be reenciphered if the KMMK was equal to the SMK. When the reenciphered PKDS is refreshed to become the active PKDS, the PKA callable services control can be enabled.

On systems with a PCIXCC or CCA Crypto Express coprocessor, changing the Asymmetric-keys/RSA master key requires that the PKA callable services control be disabled. The new master key value is loaded, the PKDS is reenciphered and the Change Asymmetric Master Key utility makes the reenciphered PKDS the active PKDS. The PKA callable services control will be enabled automatically.

On systems with the CEX3C or CEX4C, the ECC master key is changed in the same manner as the DES and AES master keys. On systems with the CEX3C (with the September 2011 licensed internal code) or the CEX4C, the RSA master key is changed in the same manner as the DES, AES and ECC master keys.

## Key Strength and Wrapping of Key

Key strength is measured as “bits of security” as described in the documentation of NIST and other organizations. Each individual key will have its “bits of security” computed, then the different key types (AES, DES, ECC, RSA, HMAC) can then have their relative strengths compared on a single scale. When the raw value of a particular key falls between discreet values of the NIST table, the lower value from the table will be used as the “bits of security”.

The following tables show some examples of the restrictions due to key strength.

When wrapping an HMAC key with an AES key-encrypting key, the strength of the AES key-encrypting key depends on the attributes of the HMAC key.

*Table 9. AES EXPORTER strength required for exporting an HMAC key under an AES EXPORTER*

Key-usage field 2 in the HMAC key	Minimum strength of AES EXPORTER to adequately protect the HMAC key
SHA-256, SHA-384, SHA-512	256 bits
SHA-224	192 bits
SHA-1	128 bits

*Table 10. Minimum RSA modulus length to adequately protect an AES key*

Bit length of AES key to be exported	Minimum strength of RSA wrapping key to adequately protect the AES key
128	3072
192	7860
256	15360

## Key Strength and Key Wrapping Access Control Points

In order to comply with cryptographic standards, including ANSI X9.24 Part 1 and PCI-HSM, ICSF provides a way to ensure that a key is not wrapped with a key weaker than itself. ICSF provides a set of access control points in the ICSF role to control the wrapping of keys. ICSF administrators can use these access control points to meet an installation's individual requirements.

There are new and existing access control points that control the wrapping of keys by master and key-encrypting keys. These access control points will either prohibit the wrapping of a key by a key of weaker strength or will return a warning (return code 0, reason code non-zero) when a key is wrapped by a weaker key. All of these ACPs are disabled by default in the ICSF role.

The processing of callable services will be affected by these access control points. Here is a description of the access control points, the wrapping they control, and the effect on services. These access control points apply to symmetric and asymmetric keys.

When the **Prohibit weak wrapping - Transport keys** access control point is enabled, any service that attempts to wrap a key with a weaker transport key will fail.

When the **Prohibit weak wrapping - Master keys** access control point is enabled, any service that wraps a key under a master key will fail if the master key is weaker than the key being wrapped.

When the **Warn when weak wrap - Transport keys** access control point is enabled, any service that attempts to wrap a key with a weaker transport key will succeed with a warning reason code.

When the **Warn when weak wrap - Master keys** access control point is enabled, any service that attempts to wrap a key with a weaker master key will succeed with a warning reason code.

24-byte DATA keys with a zero control vector can be wrapped with a 16-byte key, the DES master key, or a key-encrypting key, which violates the wrapping requirements. The **Prohibit weak wrapping - Transport keys** and **Prohibit weak wrapping - Master keys** access control points do not cause services to fail for this case. The **Disallow 24-byte DATA wrapped with 16-byte Key** access control point does control this wrapping. When enabled, services will fail. The **Warn when weak wrap - Transport keys** and **Warn when weak wrap - Master keys** access control points will cause the warning to be returned when the access control points are enabled.

When the **RKX/TBC - Disallow triple-length MAC key** access control point is enabled, CSNDRKX will fail to import a triple-length MAC key under a double-length key-encrypting key. CSNBTBC will not wrap a triple-length MAC key under a double-length key-encrypting key. The **Prohibit weak wrapping - Transport keys** and **Prohibit weak wrapping - Master keys** access control points do not cause services to fail for this case. The **Warn when weak wrap - Transport keys** and **Warn when weak wrap - Master keys** access control points will cause the warning to be returned when the ACPs are enabled.

If the **Prohibit Weak Wrap** access control point is enabled, RSA private keys may not be wrapped using a weaker DES key-encrypting key. Enabling the **Allow weak DES wrap of RSA private key** access control points will override this restriction.

## RSA Private Key Tokens

The existing RSA private key tokens use a DES object protection key to wrap the private key parts of the key. This wrapping is not compliant for large modulus sizes. New private key sections have been introduced for RSA keys where the object protection key is an AES key. These private key sections are compliant.

---

## PKA Callable Services

The Cryptographic Coprocessor Feature available on the IBM @server zSeries 900, provides RSA and DSS digital signature functions, key management functions, and DES key distribution functions.

The IBM @server zSeries 900 provides the ability to generate RSA keys on the PCI Cryptographic Coprocessor. ICSF provides application programming interfaces to these functions through callable services.

The PCIXCC and CCA Crypto Express coprocessors provide RSA digital signature functions, key management functions, and DES key distribution functions, PIN, MAC and data encryption functions, and application programming interfaces to these functions through callable services. You can also generate RSA keys on the PCIXCC and CCA Crypto Express coprocessors.

The CEX3C and the CEX4C provide support for ECC. Specifically, they provide ECDSA digital signature functions, ECC key management functions, and application programming interfaces to these functions through callable services.

## Callable Services Supporting Digital Signatures

ICSF provides these services that support digital signatures.

### Restrictions:

- DSS is only supported on the IBM @server zSeries 900.
- ECDSA is only supported through the CEX3C and CEX4C cryptographic hardware.

### Digital Signature Generate Callable Service (CSNDDSG and CSNFDSG)

This service generates a digital signature using an RSA, DSS, or ECC private key. It supports these methods of signature generation:

- ANSI X9.30 (DSS)
- ANSI X9.30 (ECDSA)
- ANSI X9.31 (RSA)
- ISO 9796-1 (RSA)
- RSA DSI PKCS 1.0 and 1.1 (RSA)
- Padding on the left with zeros (RSA)

The input text must have been previously hashed using the one-way hash generate callable service or the MDC generation service.



## Digital Signature Verify Callable Service (CSNDDSV and CSNFDSG)

This service verifies a digital signature using an RSA, DSS, or ECC public key. This service supports these methods of signature generation:

- ANSI X9.30 (DSS)
- ANSI X9.30 (ECDSA)
- ANSI X9.31 (RSA)
- ISO 9796-1 (RSA)
- RSA DSI PKCS 1.0 and 1.1 (RSA)
- Padding on the left with zeros (RSA)

The text that is input to this service must be previously hashed using the one-way hash generate callable service or the MDC generation service.

## Callable Services for PKA Key Management

ICSF provides these services for PKA key management.

### PKA Key Generate Callable Service (CSNDPKG and CSNFPKG)

This service generates a PKA internal token for use with the DSS algorithm in digital signature services. You can then use the PKA public key extract callable service to extract a DSS public key token from the internal key token. This service also supports the generation of RSA keys and ECC keys.

Input to the PKA key generate callable service is either a skeleton key token created by the PKA key token build callable service or a valid key token. Upon examination of the input skeleton key token, the PKA key generate service routes the key generation request as follows:

- If the skeleton is for a DSS key token, ICSF routes the request to a Cryptographic Coprocessor Feature.
- If the skeleton is for an RSA key, ICSF routes the request to any available PCICC, PCIXCC, or CCA Crypto Express coprocessor.
- If the skeleton is for a retained RSA key, ICSF routes the request to a PCICC, PCIXCC, or CCA Crypto Express coprocessor where the key is generated and retained for additional security.
- If the skeleton is for an ECC key, ICSF routes the request to any available CEX3C or CEX4C.

### PKA Key Import Callable Service (CSNDPKI and CSNFPKI)

This service imports a PKA private key, which may be RSA or DSS.

The key token to import can be in the clear or encrypted. The PKA key token build utility creates a clear PKA key token. The PKA key generate callable service generates either a clear or an encrypted PKA key token.

### PKA Key Token Build Callable Service (CSNDPKB and CSNFPKB)

The PKA key token build callable service is a utility you can use to create an external PKA key token containing an unenciphered private RSA or DSS key. You can supply this token as input to the PKA key import callable service to obtain an operational internal token containing an enciphered private key. You can also use this service to input a clear unenciphered public ECC, RSA, or DSS key and return the public key in a token format that other PKA services can use directly.

Use this service to build skeleton key tokens for input to the PKA key generate callable service for creation of RSA keys (on the PCICC, PCIXCC, or CCA Crypto Express coprocessor) or ECC keys (on the CEX3C or CEX4C).

### **PKA Key Token Change Callable Service (CSNDKTC and CSNFKTC)**

This service changes PKA key tokens (RSA, DSS, and ECC) or trusted block key tokens, from encipherment under the cryptographic coprocessor's old RSA master key or ECC master key to encipherment under the current cryptographic coprocessor's RSA master key or ECC master key. This callable service only changes private internal tokens. An active PCICC, PCIXCC, or CCA Crypto Express coprocessor is required.

### **PKA Key Translate (CSNDPKT and CSNFPKT)**

This service translates a CCA RSA key token to an external smart card key token. An active CCA Crypto Express coprocessor is required.

### **PKA Public Key Extract Callable Service (CSNDPKX and CSNFPKX)**

This service extracts a PKA public key token from a PKA internal (operational) or external (importable) private key token. It performs no cryptographic verification of the PKA private key token.

## **Callable Services to Update the Public Key Data Set (PKDS)**

The Public Key Data Set (PKDS) is a repository for DSS, ECC, and RSA public and private keys and trusted blocks. An application can store keys in the PKDS and refer to them by label when using any of the callable services which accept public key tokens as input. The PKDS update callable services provide support for creating and writing records to the PKDS and reading and deleting records from the PKDS.

### **Coordinated KDS Administration Callable Service (CSFCRC and CSFCRC6)**

This service performs a dynamic PKDS refresh or a dynamic PKDS change master key operation. This callable service performs the refresh or change master key operation while allowing applications to update the PKDS. In a sysplex environment, this callable service enables an application to perform a coordinated sysplex-wide refresh or change master key operation from a single ICSF instance.

### **PKDS Key Record Create Callable Service (CSNDKRC and CSNFKRC)**

This service accepts an RSA, DSS, or ECC private key token in either external or internal format, or an RSA, DSS, or ECC public key token or trusted blocks and writes a new record to the PKDS. An application can create a null token in the PKDS by specifying a token length of zero. The key label must be unique.

### **PKDS Key Record Delete Callable Service (CSNDKRD and CSNFKRD)**

This service deletes a record from the PKDS. An application can specify that the entire record be deleted, or that only the contents of the record be deleted. If only the contents of the record are deleted, the record will still exist in the PKDS but will contain only binary zeros. The key label must be unique.

**Note:** Retained keys cannot be deleted from the PKDS with this service. See “Retained Key Delete (CSNDRKD and CSNFRKD)” on page 606 for information on deleting retained keys.

## **PKDS Key Record Read Callable Service (CSNDKRR and CSNFKRR)**

This service reads a record from the PKDS and returns the contents of that record to the caller. The key label must be unique.

## **PKDS Key Record Write Callable Service (CSNDKRW and CSNFKRW)**

This service accepts an RSA, DSS, or ECC private key token in either external or internal format, or an RSA, DSS, or ECC public key token or trusted blocks and writes over an existing record in the PKDS. An application can check the PKDS for a null record with the label provided and overwrite this record if it does exist. Alternatively, an application can specify to overwrite a record regardless of the contents of the record.

**Note:** Retained keys cannot be written to the PKDS with the PKDS Key Record Write service, nor can a retained key record in the PKDS be overwritten with this service.

## **Callable Services for Working with Retained Private Keys**

Private keys can be generated, retained, and used within the secure boundary of a PCICC, PCIXCC, or CCA Crypto Express coprocessor. Retained keys are generated by the PKA Key Generate (CSNDPKG) callable service. The private key values of retained keys never appear in any form outside the secure boundary. All retained keys have an entry in the PKDS that identifies the PCICC, PCIXCC, or CCA Crypto Express coprocessor where the retained private key is stored. ICSF provides these callable services to list and delete retained private keys.

### **Retained Key Delete Callable Service (CSNDRKD and CSNFRKD)**

The retained key delete callable service deletes a key that has been retained within a PCICC, PCIXCC, or CCA Crypto Express coprocessor and also deletes the record containing the key token from the PKDS.

### **Retained Key List Callable Service (CSNDRKL and CSNFKRL)**

The retained key list callable service lists the key labels of private keys that are retained within the boundaries of PCICC, PCIXCC, or CCA Crypto Express coprocessor installed on your server.

### **Clearing the retained keys on a coprocessor**

The retained keys on a PCICC, PCIXCC, or CCA Crypto Express coprocessor may be cleared. These are the conditions under which the retained key will be lost:

- If the PCICC, PCIXCC, or CCA Crypto Express coprocessor detects tampering (the intrusion latch is tripped), ALL installation data is cleared: master keys, retained keys for all domains, as well as roles and profiles.
- If the PCICC, PCIXCC, or CCA Crypto Express coprocessor detects tampering (the secure boundary of the card is compromised), it self-destructs and can no longer be used.
- If you issue a command from the TKE workstation to zeroize a domain  
This command zeroizes the data specific to a domain: master keys and retained keys.
- If you issue a command from the Support Element panels to zeroize all domains.

This command zeroizes ALL installation data: master keys, retained keys and access control roles and profiles.

## Callable Services for SET Secure Electronic Transaction

SET is an industry-wide open standard for securing bankcard transactions over open networks. The SET protocol addresses the payment phase of a transaction from the individual, to the merchant, to the acquirer (the merchant's current bankcard processor). It can be used to help ensure the privacy and integrity of real time bankcard payments over the Internet. In addition, with SET in place, everyone in the payment process knows who everyone else is. The card holder, the merchant, and the acquirer can be fully authenticated because the core protocol of SET is based on digital certificates. Each participant in the payment transaction holds a certificate that validates his or her identity. The public key infrastructure allows these digital certificates to be exchanged, checked, and validated for every transaction made over the Internet. The mechanics of this operation are transparent to the application.

Under the SET protocol, every online purchase must be accompanied by a digital certificate which identifies the card-holder to the merchant. The buyer's digital certificate serves as an electronic representation of the buyer's credit card but does not actually show the credit card number to the merchant. Once the merchant's SET application authenticates the buyer's identity, it then decrypts the order information, processes the order, and forwards the still-encrypted payment information to the acquirer for processing. The acquirer's SET application authenticates the buyer's credit card information, identifies the merchant, and arranges settlement. With SET, the Internet becomes a safer, more secure environment for the use of payment cards.

ICSF provides these callable services that can be used in developing SET applications that make use of the S/390 and IBM @server zSeries cryptographic hardware at the merchant and acquirer payment gateway.

### **SET Block Compose Callable Service (CSNDSBC and CSNFSBC)**

The SET Block Compose callable service performs DES encryption of data, OAEP-formatting through a series of SHA-1 hashing operations, and the RSA-encryption of the Optimal Asymmetric Encryption Padding (OAEP) block.

### **SET Block Decompose Callable Service (CSNDSBD and CSNFSBD)**

The SET Block Decompose callable service decrypts both the RSA-encrypted and the DES-encrypted data.

---

## PKA Key Tokens

PKA key tokens contain RSA, DSS or ECC private or public keys. PKA tokens are variable length because they contain either RSA, DSS, or ECC key values, which are variable in length. Consequently, length parameters precede all PKA token parameters. The maximum allowed size is 3500 bytes. PKA key tokens consist of a token header, any required sections, and any optional sections. Optional sections depend on the token type. PKA key tokens can be public or private, and private key tokens can be internal or external. Therefore, there are three basic types of tokens, each of which can contain either RSA, DSS, or ECC information:

- A public key token
- A private external key token
- A private internal key token

Public key tokens contain only the public key. Private key tokens contain the public and private key pair. Table 11 on page 92 summarizes the sections in each type of token.

Table 11. Summary of PKA Key Token Sections

Section	Public External Key Token	Private External Key Token	Private Internal Key Token
Header	X	X	X
RSA, DSS, or ECC private key information		X	X
RSA, DSS, or ECC public key information	X	X	X
Key name (optional)		X	X
Internal information			X

As with DES and AES key tokens, the first byte of a PKA key token contains the token identifier which indicates the type of token.

A first byte of X'1E' indicates an external token with a cleartext public key and optionally a private key that is either in cleartext or enciphered by a transport key-encrypting key. An external key token is in importable key form. It can be sent on the link.

A first byte of X'1F' indicates an internal token with a cleartext public key and a private key that is enciphered by the PKA master key and ready for internal use. An internal key token is in operational key form. A PKA private key token must be in operational form for ICSF to use it. (PKA public key tokens are used directly in the external form.)

Formats for public and private external and internal RSA, DSS, and ECC key tokens begin in "RSA Public Key Token" on page 851.

## PKA Key Management

You can also generate PKA keys in several ways.

- Using the ICSF PKA key generate callable service.
- Using the Transaction Security System PKA key generate verb, or a comparable product from another vendor.

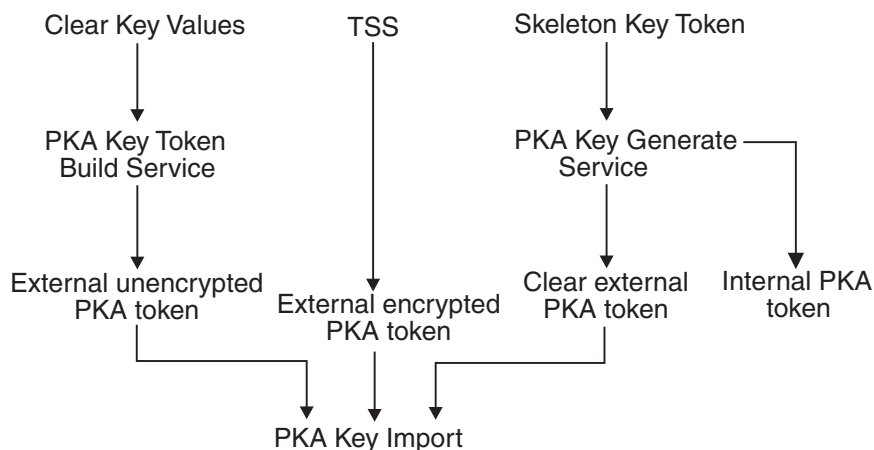


Figure 7. PKA Key Management

I With a PCIXCC or CCA Crypto Express coprocessor, you can use the ICSF PKA key generate callable service to generate internal and external PKA tokens. You can also generate RSA keys on another system. To input a clear RSA key to ICSF, create the token with the PKA key token build callable service and import it using the PKA key import callable service. To input an encrypted RSA key, generate the key on the Transaction Security System and import it using the PKA key import callable service.

In either case, use the PKA key token build callable service to create a skeleton key token as input (see “PKA Key Token Build (CSNDPKB and CSNFPKB)” on page 583).

You can generate DSS keys on another system or on ICSF. You need to supply DSS network quantities to the PKA key generate callable service. If you generate DSS keys on another system, you can import them the same way as RSA keys. If you generate a DSS key on ICSF, you can never export it. You can use it on another ICSF host only if the same PKA master keys are installed on both systems.

The PKA key import callable service uses the clear token from the PKA key token build service or a clear or encrypted token from the Transaction Security System to securely import the key token into operational form for ICSF to use. ICSF does not permit the export of the imported PKA key.

The PKA public key extract callable service builds a public key token from a private key token.

Application RSA, DSS, and ECC public and private keys can be stored in the public key data set (PKDS), a VSAM data set.

---

## Security and Integrity of the Token

PKA private key tokens may optionally have a 64-byte *private\_key\_name* field. If *private\_key\_name* exists, ICSF uses RACROUTE REQUEST=AUTH to verify it prior to using the token in a callable service. For additional security, the processor also validates the entire private key token.

---

## Key Identifier for PKA Key Token

A *key identifier* for a PKA key token is a variable length (maximum allowed size is 3500 bytes) area that contains one of these:

- **Key label** identifies keys that are in the PKDS. Ask your ICSF administrator for the key labels that you can use.
- **Key token** can be either an internal key token, an external key token, or a null key token. Key tokens are generated by an application (for example, using the PKA key generate callable service), or received from another system that can produce external key tokens.

An **internal key token** can be used only on ICSF, because a PKA master key encrypts the key value. Internal key tokens contain keys in operational form only.

An **external key token** can be exchanged with other systems because a transport key that is shared with the other system encrypts the key value. External key tokens contain keys in either exportable or importable form.



A **null key token** consists of 8 bytes of binary zeros. The PKDS Key Record Create service can be used to write a null token to the PKDS. This PKDS record can subsequently be identified as the target token for the PKA key import or PKA key generate service.

The term *key identifier* is used when a parameter could be one of the previously discussed items and to indicate that different inputs are possible. For example, you may want to specify a specific parameter as either an internal key token or a key label. The key label is, in effect, an indirect reference to a stored internal key token.

## Key Label

If the first byte of the key identifier is greater than X'40', the field is considered to be holding a **key label**. The contents of a key label are interpreted as a pointer to a public key data set (PKDS) key entry. The key label is an indirect reference to an internal key token.

A key label is specified on callable services with the *key\_identifier* parameter as a 64-byte character string, left-justified, and padded on the right with blanks. In most cases, the callable service does not check the syntax of the key label beyond the first byte. One exception is the CKDS key record create callable service which enforces the KGUP rules for key labels unless syntax checking is bypassed by a preprocessing exit.

A key label has this form:

Offset	Length	Data
00-63	64	Key label name

## Key Token

A key token is a variable length (maximum allowed size is 3500 bytes) field composed of key value and control information. PKA keys can be either public or private RSA, DSS, or ECC keys. Each key token can be either an internal key token (the first byte of the key identifier is X'1F'), an external key token (the first byte of the key identifier is X'1E'), or a null private key token (the first byte of the key identifier is X'00'). For the format of each token type, refer to Appendix B, "Key Token Formats," on page 835.

An internal key token is a token that can be used only on the ICSF system that created it (or another ICSF system with the same PKA master key). It contains a key that is encrypted under the PKA master key.

An application obtains an internal key token by using one of the callable services such as those listed. The callable services are described in detail in Chapter 10, "Managing PKA Cryptographic Keys."

- PKA key generate
- PKA key import

The PKA Key Token Change callable service can reencipher private internal tokens from encryption under the old ASYM-MK to encryption under the current ASYM-MK. PKDS Reencipher/Activate options are available to reencipher RSA, DSS and ECC internal tokens in the PKDS when the SMK/ASYM-MK keys are changed.

PKA master keys may not be changed dynamically.



For debugging information, see Appendix B, “Key Token Formats” for the format of an internal key token.

If the first byte of the key identifier is X'1E', the key identifier is interpreted as an **external key token**. An external PKA key token contains key (possibly encrypted) and control information. By using the external key token, you can exchange keys between systems.

An application obtains the external key token by using one of the callable services such as those listed. They are described in detail in Chapter 10, “Managing PKA Cryptographic Keys.”

- PKA public key extract
- PKA key token build
- PKA key generate

For debugging information, see Appendix B, “Key Token Formats” for the format of an external key token.

If the first byte of the key identifier is X'00', the key identifier is interpreted as a **null key token**.

For debugging information, see Appendix B, “Key Token Formats” for the format of a null key token.

---

## The Transaction Security System and ICSF Portability

The Transaction Security System PKA verbs from releases prior to 1996 can run only on the Transaction Security System. The PKA96 release of the Transaction Security System PKA verbs generally runs on ICSF without change. As with DES cryptography, you cannot interchange internal PKA tokens but can interchange external tokens.

---

## Summary of the PKA Callable Services

Table 12 lists the PKA callable services, described in this publication, and their corresponding verbs. (The PKA services start with CSNDxxx and have corresponding CSFxxx names.) This table also references the topic that describes the callable service.

*Table 12. Summary of PKA Callable Services*

Verb	Service Name	Function
<b>Chapter 8, “Financial Services”</b>		
CSNDSBC CSNFSBC	SET block compose	Composes the RSA-OAEP block and the DES-encrypted block in support of the SET protocol.
CSNDSBD	SET block decompose	Decomposes the RSA-OAEP block and the DES-encrypted block to provide unencrypted data back to the caller.
<b>Chapter 9, “Using Digital Signatures”</b>		
CSNDDSG CSNFDSG	Digital signature generate	Generates a digital signature using a PKA private key supporting RSA, DSS, and ECDSA algorithms.
CSNDDSV CSNFDSV	Digital signature verify	Verifies a digital signature using a PKA public key supporting RSA, DSS, and ECDSA algorithms.
<b>Chapter 10, “Managing PKA Cryptographic Keys”</b>		

Table 12. Summary of PKA Callable Services (continued)

Verb	Service Name	Function
CSNDPKG CSNFPKG	PKA key generate	Generates a DSS internal token for use in digital signature services, RSA keys (for use on the PCICC, PCIXCC, or CCA Crypto Express coprocessor) and ECC keys (for use on the CEX3C or CEX4C).
CSNDPKI CSNFPKI	PKA key import	Imports a PKA key token containing either a clear PKA key or a PKA key enciphered under a limited authority IMP-PKA KEK.
CSNDPKB CSNFPKB	PKA key token build	Creates an external PKA key token containing a clear private RSA or DSS key. Using this token as input to the PKA key import callable service returns an operational internal token containing an enciphered private key. Using CSNDPKB on a clear public RSA or DSS key, returns the public key in a token format that other PKA services can directly use. CSNDPKB can also be used to create a skeleton token for input to the PKA Key Generate service for the generation of an internal DSS or RSA key token.
CSNDKTC CSNFKTC	PKA key token change	Changes PKA key tokens (RSA, DSS, and ECC) or trusted block key tokens, from encipherment under the cryptographic coprocessor's old RSA master key or ECC master key to encipherment under the current cryptographic coprocessor's RSA master key or ECC master key. This callable service only changes private internal tokens.
CSNDPKT CSNFPKT	PKA key translate	Translates a CCA RSA key token to a smart card format.
CSNDPKX	PKA public key extract	Extracts a PKA public key token from a supplied PKA internal or external private key token. Performs no cryptographic verification of the PKA private token.
CSNDRKD CSNFRKD	Retained key delete	Deletes a key that has been retained within the PCICC, PCIXCC, or CCA Crypto Express coprocessor.
CSNDRKL CSNFRKL	Retained key list	Lists key labels of keys that have been retained within all currently active PCICCs, PCIXCCs, or CCA Crypto Express coprocessors.
<b>Chapter 11, "Key Data Set Management"</b>		
CSNDKRC CSNFKRC	PKDS key record create	Writes a new record to the PKDS.
CSNDKRD CSNFKRD	PKDS key record delete	Delete a record from the PKDS.
CSNDKRR CSNFKRR	PKDS key record read	Read a record from the PKDS and return the contents of that record.
CSNDKRW CSNFKRW	PKDS key record write	Write over an existing record in the PKDS.

---

## Chapter 4. Introducing PKCS #11 and using PKCS #11 callable services

The Integrated Cryptographic Service Facility has implemented callable service in support of PKCS #11. A callable service is a routine that receives control using a CALL statement in an application language. Each callable service performs one or more functions, including:

- initializing and deleting PKCS11 tokens
- creating, reading, updating and deleting PKCS11 objects
- performing cryptographic operations

Many services have hardware requirements. See each service for details. All new callable services will be invocable in AMODE(24), AMODE(31), or AMODE(64).

For more information about PKCS #11 see *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

---

### PKCS #11 Management Services

ICSF provides callable services that support PKCS #11 token and object management. The following table summarizes these callable services. For complete syntax and reference information, refer to Part 3, “PKCS #11 Callable Services,” on page 707.

*Table 13. Summary of PKCS #11 callable services*

Verb	Service Name	Function
CSFPDVK	PKCS #11 Derive key	Generate a new secret key object from an existing key object
CSFPDMK	PKCS #11 Derive multiple keys	Generate multiple secret key objects and protocol dependent keying material from an existing secret key object
CSFPHMG	PKCS #11 Generate HMAC	Generate a hashed message authentication code (MAC)
CSFPGKP	PKCS #11 Generate key pair	Generate an RSA, DSA, Elliptic Curve, or Diffie-Hellman key pair
CSFPGSK	PKCS #11 Generate secret key	Generate a secret key or set of domain parameters
CSFPGAV	PKCS #11 Get attribute value	List the attributes of a PKCS11 object
CSFPOWH	PKCS #11 One-way hash, sign, or verify	Generate a one-way hash on specified text, sign specified text, or verify a signature on specified text
CSFPPKS	PKCS #11 Private key sign	<ul style="list-style-type: none"><li>• Decrypt or sign data using an RSA private key using zero-pad or PKCS #1 v1.5 formatting</li><li>• Sign data using a DSA private key</li><li>• Sign data using an Elliptic Curve private key in combination with DSA</li></ul>

Table 13. Summary of PKCS #11 callable services (continued)

Verb	Service Name	Function
CSFPPRF	PKCS #11 Pseudo-random function	Generate pseudo-random output of arbitrary length.
CSFPPKV	PKCS #11 Public key verify	<ul style="list-style-type: none"> <li>• Encrypt or verify data using an RSA public key using zero-pad or PKCS #1 v1.5 formatting. For encryption, the encrypted data is returned</li> <li>• Verify a signature using a DSA public key. No data is returned</li> <li>• Verify a signature using an Elliptic Curve public key in combination with DSA. No data is returned</li> </ul>
CSFPSKD	PKCS #11 Secret key decrypt	Decipher data using a clear symmetric key
CSFPSKE	PKCS #11 Secret key encrypt	Encipher data using a clear symmetric key
CSFPSAV	PKCS #11 Set attribute value	Update the attributes of a PKCS11 object
CSFPTRC	PKCS #11 Token record create	Initialize or re-initialize a z/OS PKCS #11 token, creates or copies a token object in the token data set and creates or copies a session object for the current PKCS #11 session
CSFPTRD	PKCS #11 Token record delete	Delete a z/OS PKCS #11 token, token object, or session object
CSFPTRL	PKCS #11 Token record list	Obtain a list of z/OS PKCS #11 tokens. The caller must have SAF authority to the token. Also obtains a list of token and session objects for a token. Use a search template to restrict the search for specific attributes.
CSFPUWK	PKCS #11 Unwrap key	Unwrap and create a key object using another key
CSFPHMV	PKCS #11 Verify HMAC	Verify a hash message authentication code (MAC)
CSFPWPK	PKCS #11 Wrap key	Wrap a key with another key

## Attribute List

The attributes of an object can be the input and the output of a service. The format of the attributes is shown here and applies to all PKCS #11 callable services. For the token record list service, the search\_template has the same format as an attribute list. The lengths in the attribute list and attribute structures are unsigned integers.

An *attribute\_list* is a structure in this format:

Number of attributes	Attribute	Attribute	...
2 bytes	4 + 2 + length of <i>value</i> bytes	4 + 2 + length of <i>value</i> bytes	...

Each attribute is a structure in this format:

Attribute name	Length of value ( <i>n</i> )	Value
4 bytes	2 bytes	<i>n</i> bytes

---

## Handles

A handle is a 44-byte identifier for a token or an object. The format of the handle is as follows:

Name of token or object	Sequence number	ID
32 bytes	8 bytes	4 bytes

The token name in the first 32 bytes of the handle is provided by the PKCS #11 application when the token or object is created. The first character of the name must be alphabetic or a national character ("#", "\$", or "@"). Each of the remaining characters can be alphanumeric, a national character ("#", "\$", or "@"), or a period(".").

The sequence number is a hexadecimal number stored as the EBCDIC representation of 8 hexadecimal numbers. The sequence number field in a token is EBCDIC blanks. The token record contains a last-used sequence number field, which is incremented each time an object associated with the token is created. This sequence number value is placed in the handle of the newly-created object.

The ID field is 4 characters. The first character (EBCDIC) identifies the object's category:

- S** the handle belongs to a clear session object
- T** the handle belongs to a clear token object
- U** the handle belongs to a clear state object
- X** the handle belongs to a secure session object
- Y** the handle belongs to a secure token object
- Z** the handle belongs to a secure state object

If the first character is blank, the handle belongs to a token.

The last three characters must be EBCDIC blanks.



---

## Part 2. CCA Callable Services

This publication introduces DES, AES and PKA callable services.

**Note:** References to the IBM @server zSeries 800 (z800) do not appear in this information. Be aware that the documented notes and restrictions for the IBM @server zSeries 900 (z900) also apply to the z800.





---

## Chapter 5. Managing Symmetric Cryptographic Keys

This topic describes the callable services that generate and maintain cryptographic keys.

Using ICSF, you can generate keys using either the key generator utility program or the key generate callable service. ICSF provides a number of callable services to assist you in managing and distributing keys and maintaining the cryptographic key data set (CKDS).

This topic describes these callable services:

- “Clear Key Import (CSNBCKI and CSNECKI)” on page 104
- “Control Vector Generate (CSNBCVG and CSNECVG)” on page 106
- “Control Vector Translate (CSNBCVT and CSNECVT)” on page 110
- “Cryptographic Variable Encipher (CSNBCVE and CSNECVE)” on page 114
- “Data Key Export (CSNBDKX and CSNEDKX)” on page 116
- “Data Key Import (CSNBDKM and CSNEDKM)” on page 119
- “Diversified Key Generate (CSNBDKG and CSNEDKG)” on page 122
- “ECC Diffie-Hellman (CSNDEDH and CSNFEDH)” on page 129
- “Key Export (CSNBKEX and CSNEKEX)” on page 137
- “Key Generate (CSNBKGN and CSNEKGN)” on page 142
- “Key Generate2 (CSNBKGN2 and CSNEKGN2)” on page 156
- “Key Import (CSNBKIM and CSNEKIM)” on page 165
- “Key Part Import (CSNBKPI and CSNEKPI)” on page 171
- “Key Part Import2 (CSNBKPI2 and CSNEKPI2)” on page 176
- “Key Test (CSNBKYT and CSNEKYT)” on page 179
- “Key Test2 (CSNBKYT2 and CSNEKYT2)” on page 184
- “Key Test Extended (CSNBKYTX and CSNEKTX)” on page 188
- “Key Token Build (CSNBKTB and CSNEKTB)” on page 192
- “Key Token Build2 (CSNBKTB2 and CSNEKTB2)” on page 202
- “Key Translate (CSNBKTR and CSNEKTR)” on page 209
- “Key Translate2 (CSNBKTR2 and CSNEKTR2)” on page 211
- “Multiple Clear Key Import (CSNBCKM and CSNECKM)” on page 218
- “Multiple Secure Key Import (CSNBCKM and CSNECKM)” on page 222
- “PKA Decrypt (CSNDPKD and CSNFPKD)” on page 228
- “PKA Encrypt (CSNDPKE and CSNFPKE)” on page 234
- “Prohibit Export (CSNBPEX and CSNEPEX)” on page 239
- “Prohibit Export Extended (CSNBPEXX and CSNEPEXX)” on page 241
- “Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)” on page 243
- “Remote Key Export (CSNDRKX and CSNFRKX)” on page 247
- “Restrict Key Attribute (CSNBRKA and CSNERKA)” on page 255
- “Secure Key Import (CSNBSKI and CSNESKI)” on page 260
- “Secure Key Import2 (CSNBSKI2 and CSNESKI2)” on page 264
- “Symmetric Key Export (CSNDSYX and CSNFSYX)” on page 269
- “Symmetric Key Generate (CSNDSYG and CSNFSYG)” on page 275
- “Symmetric Key Import (CSNDSYI and CSNFSYI)” on page 283
- “Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2)” on page 291
- “Transform CDMF Key (CSNBTCK and CSNETCK)” on page 295
- “Trusted Block Create (CSNDTBC and CSNETBC)” on page 298
- “TR-31 Export (CSNBT31X and CSNET31X)” on page 301
- “TR-31 Import (CSNBT31I and CSNET31I)” on page 315
- “TR-31 Optional Data Build (CSNBT31O and CSNET31O)” on page 329

- “TR-31 Optional Data Read (CSNBT31R and CSNET31R)” on page 332
- “TR-31 Parse (CSNBT31P and CSNET31P)” on page 336
- “Unique Key Derive (CSNBUKD and CSNEUKD)” on page 339
- “User Derived Key (CSFUDK and CSFUDK6)” on page 347

## Clear Key Import (CSNBCKI and CSNECKI)

Use the clear key import callable service to import a clear DATA key that is to be used to encipher or decipher data. This callable service can import only DATA keys. Clear key import accepts an 8-byte clear DATA key, enciphers it under the master key, and returns the encrypted DATA key in operational form in an internal key token.

If the clear key value does not have odd parity in the low-order bit of each byte, the service returns a warning value in the *reason\_code* parameter. The callable service does not adjust the parity of the key.

**Note:** To import 16-byte or 24-byte DATA keys, use the multiple clear key import callable service that is described in “Multiple Clear Key Import (CSNBCKM and CSNECKM)” on page 218. The multiple clear key import service supports AES DATA keys.

The callable service name for AMODE(64) invocation is CSNECKI.

### Format

```
CALL CSNBCKI(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    clear_key,
    key_identifier )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

#### **clear\_key**

Direction: Input

Type: String

The *clear\_key* specifies the 8-byte clear key value to import.

#### **key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string that is to receive the internal key token. “Key Identifier for Key Token” on page 8 describes the internal key token.

## Usage Notes

These usage notes only apply to CCF systems.

This service produces an internal DATA token with a control vector which is usable on the Cryptographic Coprocessor Feature. If a valid internal token is supplied as input to the service in the *key\_identifier* field, that token's control vector will not be used in the encryption of the clear key value.

This service marks this internal key token CDMF or DES, according to the system's default encryption algorithm, unless token copying overrides this. The service marks this internal key token CDMF or DES, according to the system's default encryption algorithm, unless token copying overrides this. See “System Encryption Algorithm” on page 52 for details.

## Access Control Points

The **Clear Key Import/Multiple Clear Key Import - DES** access control point controls the function of this service.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 14. Clear key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	

## Clear Key Import

Table 14. Clear key import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 890 IBM @server zSeries 990	PCI X Cryptographic Coprocessor/Crypto Express2 Coprocessor	There are no internal token markings for CDMF or DES. There is no token copying.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	There are no internal token markings for CDMF or DES. There is no token copying.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	There are no internal token markings for CDMF or DES. There is no token copying.
z196 z114	Crypto Express3 Coprocessor	There are no internal token markings for CDMF or DES. There is no token copying.
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	There are no internal token markings for CDMF or DES. There is no token copying.

---

## Control Vector Generate (CSNBCVG and CSNECVG)

The Control Vector Generate callable service builds a control vector from keywords specified by the *key\_type* and *rule\_array* parameters.

The callable service name for AMODE(64) is CSNECVG.

### Format

```
CALL CSNBCVG(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    key_type,  
    rule_array_count,  
    rule_array,  
    reserved,  
    control_vector )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

**exit\_data**

Direction: Ignored

Type: String

This field is ignored.

**key\_type**

Direction: Input

Type: String

A string variable containing a keyword for the key type. The keyword is 8 bytes in length, left justified, and padded on the right with space characters. It is taken from this list:

- CIPHER
- CIPHERXI
- CIPHERXL
- CIPHERXO
- CVARDEC
- CVARENC
- CVARPINE
- CVARXCVL
- CVARXCVR
- DATA
- DATAM
- DATAMV
- DECIPHER
- DKYGENKY
- ENCIPHER
- EXPORTER
- IKEYXLAT
- IMPORTER
- IPINENC
- KEYGENKY
- MAC
- MACVER
- OKEYXLAT
- OPINENC

## Control Vector Generate

- PINGEN
- PINVER
- SECMSG

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter.

### **rule\_array**

Direction: Input

Type: Character String

Keywords that provide control information to the callable service. Each keyword is left justified in 8-byte fields, and padded on the right with blanks. All keywords must be in contiguous storage. “Key Token Build (CSNBKTB and CSNEKTB)” on page 192 illustrates the key type and key usage keywords that can be combined in the Control Vector Generate and Key Token Build callable services to create a control vector. The rule array keywords are:

- AMEX-CSC
- ANSI9.9
- ANY
- ANY-MAC
- CLR8-ENC
- CPINENC
- CPINGEN
- CPINGENA
- CVVKEY-A
- CVVKEY-B
- DALL
- DATA
- DDATA
- DEXP
- DIMP
- DKYL0
- DKYL1
- DKYL2
- DKYL3
- DKYL4
- DKYL5
- DKYL6
- DKYL7
- DMAC
- DMKEY
- DMPIN
- DMV
- DOUBLE
- DOUBLE-O



- DPVR
- ENH-ONLY
- EPINGEN
- EPINGENA
- EPINVER
- EXEX
- EXPORT
- GBP-PIN
- GBP-PINO
- IBM-PIN
- IBM-PINO
- IMEX
- IMIM
- IMPORT
- INBK-PIN
- KEY-PART
- KEYLN8
- KEYLN16
- LMTD-KEK
- MIXED
- NO-SPEC
- NO-XPORT
- NON-KEK
- NOOFFSET
- NOT31XPT
- OPEX
- OPIM
- REFORMAT
- SINGLE
- SMKEY
- SMPIN
- T31XPTOK
- TRANSLAT
- UKPT
- VISA-PVV
- XLATE
- XPORT-OK

**Note:** CLR8-ENC or UKPT must be coded in *rule\_array* when the KEYGENKY key type is coded. When the SECMSG *key\_type* is coded, either SMKEY or SMPIN must be specified in the *rule\_array*. ENH-ONLY is not supported with key type DATA.

#### reserved

Direction: Input

Type: String

## Control Vector Generate

The *reserved* parameter must be a variable of 8 bytes of X'00'.

### **control\_vector**

Direction: Output

Type: String

A 16-byte string variable in application storage where the service returns the generated control vector.

## Usage Notes

See Table 66 on page 200 for an illustration of key type and key usage keywords that can be combined in the Control Vector Generate and Key Token Build callable services to create a control vector.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 15. Control vector generate required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None.	
z114		
IBM zEnterprise EC12	None.	

---

## Control Vector Translate (CSNBCVT and CSNECVT)

The Control Vector Translate callable service changes the control vector used to encipher an external key.

See “Changing Control Vectors with the Control Vector Translate Callable Service” on page 904 for additional information about this service.

The callable service name for AMODE(64) invocation is CSNECVT.

## Format

```
CALL CSNBCVT(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    KEK_key_identifier,
    source_key_token,
    array_key_left,
    mask_array_left,
    array_key_right,
    mask_array_right,
    rule_array_count,
    rule_array,
    target_key_token )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'FFFFFFFF' (2 gigabytes). The data is defined in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **KEK\_key\_identifier**

Direction: Input/Output

Type: String

The 64-byte string variable containing an internal key token or the key label of an internal key token record containing the key-encrypting key. The control vector in the internal key token must specify the key type of IMPORTER, EXPORTER, IKEYXLAT, or OKEYXLAT.

### **source\_key\_token**

## Control Vector Translate

Direction: Input

Type: String

A 64-byte string variable containing the external key token with the key and control vector to be processed.

### **array\_key\_left**

Direction: Input/Output

Type: String

A 64-byte string variable containing an internal key token or a key label of an internal key token record that deciphers the left mask array. The internal key token must contain a control vector specifying a CVARXCVL key type.

### **mask\_array\_left**

Direction: Input

Type: String

A string of seven 8-byte elements containing the mask array enciphered under the left array key.

### **array\_key\_right**

Direction: Input/Output

Type: String

A 64-byte string variable containing an internal key token or a key label of an internal key token record that deciphers the right mask array. The internal key token must contain a control vector specifying a CVARXCVR key type.

### **mask\_array\_right**

Direction: Input

Type: String

A string of seven 8-byte elements containing the mask array enciphered under the right array key.

### **rule\_array\_count**

Direction: Input

Type: Integer

An integer containing the number of elements in the rule array. The value of the *rule\_array\_count* must be 0, 1, or 2 for this service. If the *rule\_array\_count* is 0, the default keywords are used.

### **rule\_array**

Direction: Input

Type: Character String

The *rule\_array* parameter is an array of keywords. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The *rule\_array* keywords are:

*Table 16. Keywords for Control Vector Translate*

Keyword	Meaning
<i>Parity Adjustment Rule (optional)</i>	
ADJUST	Ensures that all target key bytes have odd parity. This is the default.
NOADJUST	Prevents the parity of the target being altered.

Table 16. Keywords for Control Vector Translate (continued)

Keyword	Meaning
<i>Key-portion Rule (optional)</i>	
BOTH	Causes both halves of a 16-byte source key to be processed with the result placed into corresponding halves of the target key. When you use the BOTH keyword, the mask array must be able to validate the translation of both halves.
LEFT	Causes an 8-byte source key, or the left half of a 16-byte source key, to be processed with the result placed into both halves of the target key. This is the default.
RIGHT	Causes the right half of a 16-byte source key to be processed with the result placed into the right half of the target key. The left half is copied unchanged (still enciphered) from the source key.
SINGLE	Causes the left half of the source key to be processed with the result placed into the left half of the target key token. The right half of the target key is unchanged.

**target\_key\_token**

Direction: Input/Output

Type: String

A 64-byte string variable containing an external key token with the new control vector. This key token contains the key halves with the new control vector.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *target\_key\_token* will be wrapped in the same manner as the input *source\_key\_token*.

**Restrictions**

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

If *KEK\_key\_identifier* is a label of an IMPORTER or EXPORTER key, the label must be unique in the CKDS.

**Access Control Point**

The **Control Vector Translate** access control point controls the function of this service.

**Required Hardware**

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

## Control Vector Translate

Table 17. Control vector translate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	ENH-ONLY, USECONFIG, WRAP-ENC and WRAP-ECB not supported.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	ENH-ONLY, USECONFIG, WRAP-ENC and WRAP-ECB not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ENH-ONLY, USECONFIG, WRAP-ENC and WRAP-ECB not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	ENH-ONLY, USECONFIG, WRAP-ENC and WRAP-ECB not supported.
	Crypto Express3 Coprocessor	Enhanced key token wrapping not supported.
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## Cryptographic Variable Encipher (CSNBCVE and CSNECVE)

The Cryptographic Variable Encipher callable service uses a CVARENC key to encrypt plaintext by using the Cipher Block Chaining (CBC) method. You can use this service to prepare a mask array for the Control Vector Translate service. The plaintext must be a multiple of eight bytes in length.

The callable service name for AMODE(64) invocation is CSNECVE.

### Format

```
CALL CSNBCVE(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    c-variable_encrypting_key_identifier,  
    text_length,  
    plaintext,  
    initialization_vector,  
    ciphertext )
```

### Parameters

**return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFFFF' (2 gigabytes). The data is defined in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **c-variable\_encrypting\_key\_identifier**

Direction: Input/Output

Type: String

The 64-byte string variable containing an internal key or a key label of an internal key token record in the CKDS. The internal key must contain a control vector that specifies a CVARENC key type.

### **text\_length**

Direction: Input

Type: Integer

An integer variable containing the length of the plaintext and the returned ciphertext.

### **plaintext**

Direction: Input

Type: String

A string of length 8 to 256 bytes which contains the plaintext. The data must be a multiple of 8 bytes.

### **initialization\_vector**

Direction: Input

Type: String

A string variable containing the 8-byte initialization vector that the service uses in encrypting the plaintext.

### **ciphertext**

Direction: Output

Type: String



## Cryptographic Variable Encipher

The field which receives the ciphertext. The length of this field is the same as the length of the plaintext.

### Restrictions

- The text length must be a multiple of 8 bytes.
- The maximum length of text that the security server can process is 256 bytes.

### Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

### Access Control Point

The **Cryptographic Variable Encipher** access control point controls the function of this service.

### Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 18. Cryptographic variable encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	

---

## Data Key Export (CSNBDKX and CSNEDKX)

Use the data key export callable service to reencipher a data-encrypting key (key type of DATA only) from encryption under the master key to encryption under an exporter key-encrypting key. The reenciphered key is in a form suitable for export to another system.

The data key export service generates a key token with the same key length as the input token's key.

The callable service name for AMODE(64) invocation is CSNEDKX.

## Format

```
CALL CSNBDKX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    source_key_identifier,
    exporter_key_identifier,
    target_key_identifier )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'FFFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **source\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string for an internal key token or label that contains a data-encrypting key to be reencrypted. The data-encrypting key is encrypted under the master key.

### **exporter\_key\_identifier**

## Data Key Export

Direction: Input/Output

Type: String

A 64-byte string for an internal key token or key label that contains the exporter *key\_encrypting* key. The data-encrypting key previously discussed will be encrypted under this exporter *key\_encrypting* key.

### **target\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte field that is to receive the external key token, which contains the reenciphered key that has been exported. The reenciphered key can now be exchanged with another cryptographic system.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *target\_key\_identifier* will be wrapped in the same manner as the *source\_key\_identifier*.

## Restrictions

For existing TKE V3.1 (or higher) users, you may have to explicitly enable new access control points. Current applications will fail if they use an equal key halves exporter to export a key with unequal key halves. You must have access control point 'Data Key Export - Unrestricted' explicitly enabled.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

When the service is processed on the CCF, ICSF examines the data encryption algorithm bits on the exporter key-encrypting key and DATA key for consistency. It does not export a CDMF key under a DES-marked key-encrypting key or a DES key under a CDMF-marked key-encrypting key. ICSF does not propagate the data encryption marking on the operational key to the external token.

Token marking for DES/CDMF on DATA and key-encrypting keys is ignored on a PCICC, PCIXCC, or CCA Crypto Express coprocessor.

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

*Table 19. Required access control points for Data key export*

Access Control Point	Restrictions
Data Key Export	None
Data Key Export - Unrestricted	Key-encrypting key may have equal key halves

To use a NOCV key-encrypting key with the data key export service, the **NOCV KEK usage for export-related functions** access control point must be enabled in addition to one or both of the access control points listed.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 20. Data key export required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
	PCI Cryptographic Coprocessor	ICSF routes the request to a PCI Cryptographic Coprocessor if the control vector of the <i>exporter_key_identifier</i> cannot be processed on the Cryptographic Coprocessor Feature.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	
	Crypto Express3 Coprocessor	
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor	
	Crypto Express4 Coprocessor	

## Data Key Import (CSNBDKM and CSNEDKM)

Use the data key import callable service to import an encrypted source DES single-length, double-length or triple-length DATA key and create or update a target internal key token with the master key enciphered source key.

The callable service name for AMODE(64) invocation is CSNEDKM.

### Format

```
CALL CSNBDM(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    source_key_token,  
    importer_key_identifier,  
    target_key_identifier)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

#### **source\_key\_token**

Direction: Input

Type: String

64-byte string variable containing the source key to be imported. The source key must be an external token or null token. The external key token must indicate that a control vector is present; however, the control vector is usually valued at zero. A double-length key that should result in a default DATA control vector must be specified in a version X'01' external key token. Otherwise, both single and double-length keys are presented in a version X'00' key token. For the null token, the service will process this token format as a DATA key encrypted by the importer key and a null (all zero) control vector.

#### **importer\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string variable containing the (IMPORTER) transport key or key label of the transport key used to decipher the source key.

**target\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string variable containing a null key token or an internal key token. The key token receives the imported key.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. If a skeleton key token is provided as input to this parameter, the wrapping method in the skeleton token will be used. Otherwise, the system default key wrapping method will be used to wrap the token.

**Restrictions**

For existing TKE V3.1 (or higher) users, you may have to explicitly enable new access control points. Current applications will fail if they use an equal key halves importer to import a key with unequal key halves. You must have access control point 'Data Key Import - Unrestricted' explicitly enabled.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This service does not adjust the key parity of the source key.

CDMF/DES token markings will be ignored.

**Access Control Points**

The following table shows the access control points in the ICSF role that control the function of this service.

*Table 21. Required access control points for Data key import*

Access Control Point	Restrictions
Data Key Import	None
Data Key Import - Unrestricted	Key-encrypting key may have equal key halves

To use a NOCV key-encrypting key with the data key import service, the **NOCV KEK usage for import-related functions** access control point must be enabled in addition to one or both of the access control points listed.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 22. Data key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	Does not support triple length DATA keys.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	

## Diversified Key Generate (CSNBDKG and CSNEDKG)

Use the diversified key generate service to generate a key based on the key-generating key, the processing method, and the parameter supplied. The control vector of the key-generating key also determines the type of target key that can be generated.

To use this service, specify:

- The rule array keyword to select the diversification process.
- The operational key-generating key from which the diversified keys are generated. The control vector associated with this key restricts the use of this key to the key generation process. This control vector also restricts the type of key that can be generated.
- The data and length of data used in the diversification process.
- The generated-key may be an internal token or a skeleton token containing the desired CV of the generated-key. The generated key CV must be one that is permitted by the processing method and the key-generating key. The generated-key will be returned in this parameter.
- A key generation method keyword. Some keywords require Requires May 2004 or later version of Licensed Internal Code (LIC) or a z890.

This service generates diversified keys as follows:



- Determines if it can support the process specified in rule array.
- Recovers the key-generating key and checks the key-generating key class and the specified usage of the key-generating key.
- Determines that the control vector in the generated-key token is permissible for the specified processing method.
- Determines that the control vector in the generated-key token is permissible by the control vector of the key-generating key.
- Determines the required data length from the processing method and the generated-key CV. Validates the *data\_length*.
- Generates the key appropriate to the specific processing method. Adjusts parity of the key to odd. Creates the internal token and returns the generated diversified key.

The callable service name for AMODE(64) invocation is CSNEDKG.

## Format

```
CALL CSNBDKG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    generating_key_identifier,
    data_length,
    data,
    key_identifier,
    generated_key_identifier)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

## Diversified Key Generate

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The only valid value is 1, 2, or 3.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. The processing method is the algorithm used to create the generated key. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

*Table 23. Rule Array Keywords for Diversified Key Generate*

Keyword	Meaning
<i>Processing Method for generating or updating diversified keys (required)</i>	
CLR8-ENC	Specifies that 8-bytes of clear data shall be multiply encrypted with the generating key. The <i>generating_key_identifier</i> must be a KEYGENKY key type with bit 19 of the control vector set to 1. The control vector in <i>generated_key_identifier</i> must specify a single-length key. The key type may be DATA, MAC, or MACVER. <b>Note:</b> CIPHER class keys are not supported.
SESS-XOR	Modifies an existing DATA, DATAC, MAC, DATAM, or MACVER, DATAMV single- or double-length key. Specifies the VISA method for session key generation. Data supplied may be 8 or 16 bytes of data depending on whether the <i>generating_key_identifier</i> is a single or double length key. The 8 or 16 bytes of data is XORed with the clear value of the <i>generating_key_identifier</i> . The <i>generated_key_identifier</i> has the same control vector as the <i>generating_key_identifier</i> . The <i>generating_key_identifier</i> may be DATA/DATAC, MAC/DATAM or MACVER/DATAMV key types.
TDES-DEC	Data supplied may be 8 or 16 bytes of clear data. If the <i>generated_key_identifier</i> specifies a single length key, then 8-bytes of data is TDES decrypted under the <i>generating_key_identifier</i> . If the <i>generated_key_identifier</i> specifies a double length key, then 16-bytes of data is TDES ECB mode decrypted under the <i>generating_key_identifier</i> . No formatting of data is done prior to encryption. The <i>generating_key_identifier</i> must be a DKYGENKY key type, with appropriate usage bits for the desired generated key.

Table 23. Rule Array Keywords for Diversified Key Generate (continued)

Keyword	Meaning
TDES-ENC	Data supplied may be 8 or 16 bytes of clear data. If the <i>generated_key_identifier</i> specifies a single length key, then 8-bytes of data is TDES encrypted under the <i>generating_key_identifier</i> . If the <i>generated_key_identifier</i> specifies a double length key, then 16-bytes of data is TDES ECB mode encrypted under the <i>generating_key_identifier</i> . No formatting of data is done prior to encryption. The <i>generating_key_identifier</i> must be a DKYGENKY key type, with appropriate usage bits for the desired generated key. The <i>generated_key_identifier</i> may be a single or double length key with a CV that is permitted by the <i>generating_key_identifier</i> .
TDES-XOR	Requires May 2004 or later version of Licensed Internal Code (LIC). It combines the function of the existing TDES-ENC and SESS-XOR into one step.  The generating key must be a level 0 DKYGENKY and cannot have replicated halves. The session key generated must be double length and the allowed key types are DATA, DATAC, MAC, MACVER, DATAM, DATAMV, SMPIN and SMKEY. Key type must be allowed by the generating key control vector.
TDESEMV2	Requires May 2004 or later version of Licensed Internal Code (LIC): supports generation of a session key by the EMV 2000 algorithm (This EMV2000 algorithm uses a branch factor of 2). The generating key must be a level 0 DKYGENKY and cannot have replicated halves. The session key generated must be double length and the allowed key types are DATA, DATAC, MAC, MACVER, DATAM, DATAMV, SMPIN and SMKEY. Key type must be allowed by the generating key control vector.
TDESEMV4	Requires May 2004 or later version of Licensed Internal Code (LIC): supports generation of a session key by the EMV 2000 algorithm (This EMV2000 algorithm uses a branch factor of 4). The generating key must be a level 0 DKYGENKY and cannot have replicated halves. The session key generated must be double length and the allowed key types are DATA, DATAC, MAC, MACVER, DATAM, DATAMV, SMPIN and SMKEY. Key type must be allowed by the generating key control vector.
<b>Key Wrapping Method (optional)</b>	
USECONFIG	Specifies that the system default configuration should be used to determine the wrapping method. This is the default keyword.  The system default key wrapping method can be specified using the DEFAULTWRAP parameter in the installation options data set. See the <i>z/OS Cryptographic Services ICSF System Programmer's Guide</i> .
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.

## Diversified Key Generate

Table 23. Rule Array Keywords for Diversified Key Generate (continued)

Keyword	Meaning
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens.
<i>Translation Control (optional)</i>	
ENH-ONLY	Restrict rewrapping of the <i>generated_key_identifier</i> token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method.

### **generating\_key\_identifier**

Direction: Input/Output

Type: String

The label or internal 64 byte token of a key that is a DKYLO, DKYGENKY or a subtype appropriate to the session key to be derived. The type of key-generating key depends on the processing method.

### **data\_length**

Direction: Input

Type: Integer

The length of the *data* parameter that follows. Length depends on the processing method and the generated key. The data length for TDESEMV4 or TDESEMV2 is either 18 or 34.

### **data**

Direction: Input

Type: String

Data input to the diversified key or session key generation process. Data depends on the processing method and the *generated\_key\_identifier*.

For TDESEMV4 or TDESEMV2 the data is either 18 bytes (36 digits) or 34 bytes 68 digits) or data comprised of:

- 16 bytes (32 digits) of card specific data used to create the card specific intermediate key (UDK) as per the TDES-ENC method. This will typically be the PAN and PAN Sequence number as per the EMV specifications
- 2 bytes (4 digits) of ATC (Application Transaction Count)
- (optional) 16 bytes (32 digits) of IV (Initial Value) used in the EMV

### **key\_identifier**

Direction: Input/Output

Type: String

This parameter is currently not used. It must be a 64-byte null token.

### **generated\_key\_identifier**

Direction: Input/Output

Type: String

The internal token of an operational key, a skeleton token containing the control vector of the key to be generated, or a null token. A null token can be supplied if the *generated\_key\_identifier* will be a DKYGENKY with a CV derived from the *generating\_key\_identifier*. A skeleton token or internal token is required

when *generated\_key\_identifier* will not be a DKEYGENKY key type or the processing method is not SESS-XOR. For SESS-XOR, this must be a null token. On output, this parameter contains the generated key.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *generated\_key\_token* will use the default method unless a rule array keyword overriding the default is specified.

## Restrictions

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Refer to Appendix C, "Control Vectors and Changing Control Vectors with the CVT Callable Service," on page 893 for information on the control vector bits for the DKG key generating key.

For Session key algorithm (EMV Smartcard specific), an MDK can be used in two ways:

- To calculate the Card Specific Key (or UDK) in the personalization process, call this service with the TDES-ENC method using an output token that has been primed with the CV of the final session key, for instance, if the MDK is a DMPIN, the token should have the CV of an SMPIN key; DMAC; a double length MAC; DDATA, a double length DATA key, etc.

The result would then be exported in the personalization file. This key is not usable in this form for any other calculations.

- To use the session key, call this service with the TDESEMV4 method. Provide, for input, the same card data that was used to create the UDK as well as the ATC and optionally the IV value. This is the key that will be used in EMV related Smartcard processing.

This same processing applies to those API's the generate the session key on your behalf, like CSNBPCU.

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

Table 24. Required access control points for Diversified Key Generate

Rule array keyword	Access control point
CLR8-ENC	Diversified Key Generate - CLR8-ENC
SESS-XOR	Diversified Key Generate - SESS-XOR
TDES-DEC	Diversified Key Generate - TDES-DEC
TDES-ENC	Diversified Key Generate - TDES-ENC
TDES-XOR	Diversified Key Generate - TDES-XOR
TDESEMV2 or TDESEMV4	Diversified Key Generate - TDESEMV2/TDESEMV4

## Diversified Key Generate

Table 24. Required access control points for Diversified Key Generate (continued)

Rule array keyword	Access control point
WRAP-ECB or WRAP-ENH and default key-wrapping method setting does not match the keyword	Diversified Key Generate - Allow wrapping override keywords

When a key-generating key of key type DKYGENKY is specified with control vector bits (19 – 22) of B'1111', the **Diversified Key Generate - DKYGENKY – DALL** access control point must also be enabled in the ICSF role.

When using the TDES-ENC or TDES-DEC modes, you can specifically enable generation of a single-length key or a double-length key with equal key-halves by enabling the **Diversified Key Generate - Single length or same halves** access control point.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 25. Diversified key generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	Keywords TDES-XOR, TDESEMV2 and TDESEMV4 are not supported.  ENH-ONLY, USECONFIG, WRAP-ENH and WRAP-ECB not supported.  Enhanced key token wrapping not supported.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	ENH-ONLY, USECONFIG, WRAP-ENH and WRAP-ECB not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	Enhanced key token wrapping not supported.
IBM System z9 EC IBM System z9 BC	Cryptographic Express2 Coprocessor	ENH-ONLY, USECONFIG, WRAP-ENH and WRAP-ECB not supported.  Enhanced key token wrapping not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	ENH-ONLY, USECONFIG, WRAP-ENH and WRAP-ECB not supported.  Enhanced key token wrapping not supported.
	Crypto Express3 Coprocessor	Enhanced key token wrapping not supported.

Table 25. Diversified key generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## ECC Diffie-Hellman (CSNDEDH and CSNFEDH)

Use the ECC Diffie-Hellman callable service to create:

- Symmetric key material from a pair of ECC keys using the Elliptic Curve Diffie-Hellman protocol and the static unified model key agreement scheme.
- “Z” – The “secret” material output from D-H process.

Output may be one of the following forms:

- Internal CCA Token (DES or AES): AES keys are in the "Variable-length Symmetric Key Token" format. DES keys are in the "DES Internal Key Token" format.
- External CCA Token (DES or AES): AES keys are in the "Variable-length Symmetric Key Token" format. DES keys are in the "DES External Key Token" format.
- “Z” – The “secret” material output from D-H process.



## Format

```
CALL CSNDEDH(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    private_key_identifier_length,
    private_key_identifier,
    private_KEK_key_identifier_length,
    private_KEK_key_identifier,
    public_key_identifier_length,
    public_key_identifier,
    chaining_vector_length,
    chaining_vector,
    party_identifier_length,
    party_identifier,
    key_bit_length,
    reserved_length,
    reserved,
    reserved2_length,
    reserved2,
    reserved3_length,
    reserved3,
    reserved4_length,
    reserved4,
    reserved5_length,
    reserved5,
    output_KEK_key_identifier_length,
    output_KEK_key_identifier,
    output_key_identifier_length,
    output_key_identifier)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'FFFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the rule\_array parameter. Valid values are between 1 and 5.

**rule\_array**

Direction: Input

Type: String

The *rule\_array* parameter is an array of keywords. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The rule\_array keywords are:

*Table 26. Keywords for ECC Diffie-Hellman*

Keyword	Meaning
<i>Key agreement (one required)</i>	
DERIV01	Use the static unified model key agreement scheme.
PASSTHRU	Skip Key derivation step and return raw "Z" material.
<i>Transport Key Type (one optional if output KEK key identifier is present)</i>	
OKEK-DES	The output KEK key identifier is a "DES" KEK token.
OKEK-AES	The output KEK key identifier is a "AES" KEK token.
<i>Output Key Type (one optional if output key identifier is present)</i>	
KEY-DES	The output key identifier is a "DES" skeleton token.
KEY-AES	The output key identifier is an "AES" skeleton token.
<i>Key Wrapping Method (one optional, only supported when the output type is DES)</i>	
USECONFIG	Specifies that the configuration setting for the default wrapping method is to be used to wrap the key. This is the default.
WRAP-ENH	Specifies that the new enhanced wrapping method is to be used to wrap the key.
WRAP-ECB	Specifies that the original wrapping method is to be used.
<i>Translation Control (one optional, only supported when the output type is DES)</i>	
ENH-ONLY	Specify this keyword to indicate that the key once wrapped with the enhanced method cannot be wrapped with the original method. This restricts translation to the original method. If the keyword is not specified translation to the original method will be allowed. This turns on bit 56 (ENH ONLY) in the control vector. This keyword is not valid if processing a zero CV data key.

**private\_key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *private\_key\_identifier* parameter.

**private\_key\_identifier**

## ECC Diffie-Hellman

Type: String

The *private\_key\_identifier* must contain an internal or an external token or a label of an internal or external ECC key. The ECC key token must contain a public-private key pair. Clear keys will be accepted.

private KEK key identifier length

Type: Integer

The length of the *private\_KEK\_key\_identifier* in bytes. The maximum value is 900. If the *private\_key\_identifier* contains an internal ECC token this value must be a zero.

private KEK key identifier

Type: String

The *private\_KEK\_key\_identifier* must contain a KEK key token, the label of a KEK key token, or a null token. The KEK key token must be present if the *private key identifier* contains an external ECC token.

public key identifier length

Type: Integer

The length of the *public\_key\_identifier*.

public key identifier

Type: String

The *public\_key\_identifier* parameter must contain an ECC public token or the label of an ECC Public token. The *public\_key\_identifier* specifies the other party's ECC public key which is enabled for key management functions. If the *public\_key\_identifier* identifies a token containing a public-private key pair, no attempt to decrypt the private part will be made.

chaining vector length

Type: Integer

The *chaining\_vector\_length* parameter must be zero.

## chaining\_vector

Type: String

The *chaining\_vector* parameter is ignored.

party identifier length

Type: Integer

The length of the `party_identifier` parameter. Valid values are 0, or between 8 and 64. The `party_identifier_length` must be 0 when the PASSTHRU rule array keyword is specified.

**party\_identifier**

Type: String

The *party\_identifier* parameter contains the entity identifier information. This information should contain the both entities data according to NIST SP800-56A Section 5.8 when the DERIV01 rule array keyword is specified.

**key\_bit\_length**

Direction: Input/Output

Type: Integer

The key bit length parameter contains the number of bits of key material to derive and place in the provided key token. The value must be 0 if the PASSTHRU rule array keyword was specified. Otherwise it must be 64 - 2048.

**reserved\_length**

Direction: Input/Output

Type: Integer

The *reserved\_length* parameter must be zero.

**reserved**

Direction: Input/Output

Type: String

This parameter is ignored.

**reserved2\_length**

Direction: Input/Output

Type: Integer

The *reserved2\_length* parameter must be zero.

**reserved2**

Direction: Input/Output

Type: String

This parameter is ignored.

**reserved3\_length**

Direction: Input/Output

Type: Integer

The *reserved3\_length* parameter must be zero.

**reserved3**

Direction: Input/Output

Type: String

This parameter is ignored.

**reserved4\_length**

Direction: Input/Output

Type: Integer

The *reserved4\_length* parameter must be zero.

**reserved4**

Direction: Input/Output

Type: String

This parameter is ignored.

**reserved5\_length**

Direction: Input/Output

Type: Integer

The *reserved5\_length* parameter must be zero.

**reserved5**

Direction: Input/Output

Type: String

This parameter is ignored.

### **output\_KEK\_key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *output\_KEK\_key\_identifier*. The maximum value is 900. The *output\_KEK\_key\_identifier\_length* must be zero if *output\_key\_identifier* will contain an internal token or if the PASSTHRU rule array keyword was specified.

### **output\_KEK\_key\_identifier**

Direction: Input

Type: String

The *output\_KEK\_key\_identifier* contains a KEK key token or the label of a KEK key if the *output\_key\_identifier* will contain an external ECC token. Otherwise this field is ignored.

If the output KEK key identifier identifies a DES KEK, then it must be an IMPORTER or an EXPORTER key type, and have the export bit set. The XLATE bit is not checked. If the output KEK key identifier identifies an AES KEK, then it must be either an IMPORTER or an EXPORTER key type and have the export/import bit set in key usage field 1 and the derivation bit set in key usage field 4.

### **output\_key\_identifier\_length**

Direction: Input/Output

Type: Integer

The length of the *output\_key\_identifier*. The service checks the field to ensure it is at least equal to the size of the token to return. On return from this service, this field is updated with the exact length of the key token created. The maximum allowed value is 900 bytes.

### **output\_key\_identifier**

Direction: Input/Output

Type: String

On input, the *output\_key\_identifier* must contain a skeleton token or a null token.

On output, the *output\_key\_identifier* will contain:

- An internal or an external key token containing the generated symmetric key material.
- "Z" data (in the clear) if the PASSTHRU rule array keyword was specified.

If this variable specifies an external DES key token then the output KEK key identifier must identify a DES KEK key token. If this specifies an external key token other than a DES key token then the output KEK key identifier must identify an AES KEK key token.

## **Restrictions**

The NIST security strength requirements will be enforced, with respect to ECC Curve type (input) and derived key length.

Only the following key types will be generated, skeleton key tokens of any other type will fail.

- DES: (Legacy DES token)
  - CIPHER
  - CIPHERXI
  - CIPHERXL

- CIPHERXO
- DECIPHER
- ENCIPHER
- IMPORTER
- EXPORTER
- IMP-PKA
- AES
  - DATA (Legacy AES token)
  - CIPHER (Variable-length symmetric key-token)
  - IMPORTER (Variable-length symmetric key-token)
  - EXPORTER (Variable-length symmetric key-token)

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This table lists the valid key bit lengths and the minimum curve size required for each of the supported output key types.

*Table 27. Valid key bit lengths and minimum curve size required for the supported output key types.*

Output Key ID type	Valid Key Bit Lengths	Minimum Curve Required
DES	64	P160
	128	P160
AES	128	P256
	192	P384
	256	P512

If the output key-encrypting key identifier is a weaker key than the key being generated, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key or the key-encrypting key is a double-length key.

## Access Control Points

The ECC Diffie-Hellman callable service requires the **ECC Diffie-Hellman Callable Service** access control point to be enabled in the active role.

Specifying the PASSTHRU rule array keyword requires that the **ECC Diffie-Hellman – Allow PASSTHRU** access control point be enabled in the active role.

## ECC Diffie-Hellman

If the *output\_key\_identifier* parameter references a DES key token and the wrapping method specified is not the default method, then the **ECC Diffie-Hellman – Allow key wrap override** access control point must be enabled in the active role.

Each Elliptic Curve type supported has its own access control point. The access control point must be enabled to use the curve type and strength.

- ECC Diffie-Hellman – Allow Prime Curve 192
- ECC Diffie-Hellman – Allow Prime Curve 224
- ECC Diffie-Hellman – Allow Prime Curve 256
- ECC Diffie-Hellman – Allow Prime Curve 384
- ECC Diffie-Hellman – Allow Prime Curve 521
- ECC Diffie-Hellman – Allow BP Curve 160
- ECC Diffie-Hellman – Allow BP Curve 192
- ECC Diffie-Hellman – Allow BP Curve 224
- ECC Diffie-Hellman – Allow BP Curve 256
- ECC Diffie-Hellman – Allow BP Curve 320
- ECC Diffie-Hellman – Allow BP Curve 384
- ECC Diffie-Hellman – Allow BP Curve 512

To prevent a weaker key from being used to generate a stronger key, enable the **ECC Diffie-Hellman – Prohibit weak key generate** access control point in the ICSF role.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 28. ECC Diffie-Hellman required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This callable service is not supported.
IBM @server zSeries 990		This callable service is not supported.
IBM @server zSeries 890		This callable service is not supported.
IBM System z9 EC IBM System z9 BC		This callable service is not supported.
IBM System z10 EC IBM System z10 BC		This callable service is not supported.
z196 z114	Crypto Express3 Coprocessor	ECC Clear Key and Internal tokens support requires the Sep. 2010 licensed internal code (LIC).  ECC External and Diffie-Hellman support requires Sep. 2011 licensed internal code (LIC).



Table 28. ECC Diffie-Hellman required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise EC12	Crypto Express3 Coprocessor	
	Crypto Express4 Coprocessor	

## Key Export (CSNBKEX and CSNEKEX)

Use the key export callable service to reencrypt any type of key (except an AKEK or an IMP-PKA) from encryption under a master key variant to encryption under the same variant of an exporter key-encrypting key. The reencrypted key can be exported to another system.

If the key to be exported is a DATA key, the key export service generates a key token with the same key length as the input token's key.

This service supports the no-export bit that the prohibit export service sets in the internal token.

The callable service name for AMODE(64) invocation is CSNEKEX.

## Format

```
CALL CSNBKEX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_type,
    source_key_identifier,
    exporter_key_identifier,
    target_key_identifier )
```

## Parameters

### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### reason\_code

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### exit\_data\_length

## Key Export

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **key\_type**

Direction: Input

Type: Character string

The parameter is an 8-byte field that contains either a key type value or the keyword TOKEN. The keyword is left-justified and padded on the right with blanks.

If the key type is TOKEN, ICSF determines the key type from the control vector (CV) field in the internal key token provided in the *source\_key\_identifier* parameter. If the control vector is invalid on the Cryptographic Coprocessor Feature, the key export request will be routed to the PCI Cryptographic Coprocessor.

Key type values for the Key Export callable service are: CIPHER, CIPHERXI, CIPHERXL, CIPHERXO, DATA, DATAC, DATAM, DATAMV, DATAXLAT, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, MAC, MACD, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER. For information on the meaning of the key types, see Table 3 on page 23.

### **source\_key\_identifier**

Direction: Input

Type: String

A 64-byte string of the internal key token that contains the key to be reenciphered. This parameter must identify an internal key token in application storage, or a label of an existing key in the cryptographic key data set.

If you supply TOKEN for the *key\_type* parameter, ICSF looks at the control vector in the internal key token and determines the key type from this information. If you supply TOKEN for the *key\_type* parameter and supply a label for this parameter, the label must be unique in the cryptographic key data set.

### **exporter\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string of the internal key token or key label that contains the exporter key-encrypting key. This parameter must identify an internal key token in application storage, or a label of an existing key in the cryptographic key data set.

If the NOCV bit is on in the internal key token containing the key-encrypting key, the key-encrypting key itself (not the key-encrypting key variant) is used to encipher the generated key. For example, the key has been installed in the cryptographic key data set through the key generator utility program or the

key entry hardware using the NOCV parameter; or you are passing the key-encrypting key in the internal key token with the NOCV bit on and your program is running in supervisor state or in key 0-7.

Control vectors are explained in “Control Vector for DES Keys” on page 19 and the NOCV bit is shown in Table 350 on page 837.

#### **target\_key\_identifier**

Direction: Output

Type: String

The 64-byte field external key token that contains the reenciphered key. The reenciphered key can be exchanged with another cryptographic system.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *target\_key\_identifier* will be wrapped in the same manner as the *source\_key\_identifier*.

## **Restrictions**

For existing TKE V3.1 (or higher) users, you may have to explicitly enable new access control points. Current applications will fail if they use an equal key halves exporter to export a key with unequal key halves. You must have access control point 'Key Export - Unrestricted' explicitly enabled.

This service cannot be used to export AKEKs. Refer to “ANSI X9.17 Key Export (CSNAKEX and CSNGKEX)” on page 687 for information on exporting AKEKs.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## **Usage Notes — General**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

For key export, you can use these combinations of parameters:

- A valid key type in the *key\_type* parameter and an internal key token in the *source\_key\_identifier* parameter. The key type must be equivalent to the control vector specified in the internal key token.
- A *key\_type* parameter of TOKEN and an internal key token in the *source\_key\_identifier* parameter. The *source\_key\_identifier* can be a label with TOKEN only if the labelname is unique on the CKDS. The key type is extracted from the control vector contained in the internal key token.
- A valid key type in the *key\_type* parameter, and a label in the *source\_key\_identifier* parameter.

If internal key tokens are supplied in the *source\_key\_identifier* or *exporter\_key\_identifier* parameters, the key in one or both tokens can be reenciphered. This occurs if the master key was changed since the internal key token was last used. The return and reason codes that indicate this do *not* indicate which key was reenciphered. Therefore, assume both keys have been reenciphered.

## **Usage Notes — CCF Systems**

ICSF examines the data encryption algorithm bits on the exporter key-encrypting key and the key being exported for consistency. It does not export a CDMF key

## Key Export

under a DES-marked key-encrypting key or a DES key under a CDMF-marked key-encrypting key. ICSF does not propagate the data encryption marking on the operational key to the external token.

If the key type is MACD, the control vectors of the input keys must be the standard control vectors supported by the Cryptographic Coprocessor Feature, since the key export service will be processed on the Cryptographic Coprocessor Feature in this case.

To use NOCV key-encrypting keys or to export double-length DATAM and DATAMV keys, the NOCV-enablement keys must be installed in the CKDS. NOCV-enablement keys are only needed with the Cryptographic Coprocessor Feature.

For a double-length MAC key with a key type of DATAM, the service uses the data compatibility control vector to create an external token. For a double-length MAC key with a key type of MACD, the service uses the single-length control vector for both the left and right half of the key to create an external token (MAC || MAC). For a table of control vectors, refer to Control Vector Table.

Key Export operations which specify a NOCV key-encrypting key as the exporter key and also specify a source or key-encrypting key which contains a control vector not supported by the Cryptographic Coprocessor Feature will fail.

To export a double-length MAC generation or MAC verification key, it is recommended that a key type of TOKEN be used.

## Usage Notes — Cryptographic Coprocessor Systems

If running with a PCIXCC, CEX2C, or CEX3C, existing internal tokens created with key type MACD must be exported with either a TOKEN or DATAM key type. The external CV will be DATAM CV. The MACD key type is not supported.

To export a double-length MAC generation or MAC verification key, it is recommended that a key type of TOKEN be used.

For key types CIPHERXI, CIPHERXL, and CIPHERXO, the key-encrypting key in the exporter\_key\_identifier parameter must have a control vector with the key halves guaranteed unique flag on in the key form bits. An existing key-encrypting key can have its control vector updated using the restrict key attribute callable service.

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

Table 29. Required access control points for Key Export

Access Control Point	Restrictions
Key Export	None
Key Export - Unrestricted	Key-encrypting key may have equal key halves

To use a NOCV key-encrypting key with the key export service, the **NOCV KEK usage for export-related functions** access control point must be enabled in addition to one or both of the access control points listed.

If the output key-encrypting key identifier is a weaker key than the key being exported, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys access** control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys access** control point is enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the key-encrypting key is a double-length key.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 30. Key export required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	Key_type MACD is processed on a Cryptographic Coprocessor Feature.  Key types CIPHERXI, CIPHERXL, CIPHERXO and DATAC are not supported.
	PCI Cryptographic Coprocessor	ICSF routes the request to a PCI Cryptographic Coprocessor if: <ul style="list-style-type: none"> <li>• The key_type specified is one of these: DECIPHER, ENCIPHER, IKEYXLAT, OKEYXLAT or CIPHER.</li> <li>• The control vector of either the <i>exporter_key_identifier</i> or the <i>source_key_identifier</i> cannot be processed on the Cryptographic Coprocessor Feature.</li> <li>• Token markings for DES/CDMF on DATA and KEKs are ignored.</li> <li>• Key types CIPHERXI, CIPHERXL, and CIPHERXO are not supported.</li> </ul>
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	Key types CIPHERXI, CIPHERXL, CIPHERXO, DATAXLAT and MACD are not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	DES/CDMF token markings are ignored for DATA and key-encrypting keys.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Key types CIPHERXI, CIPHERXL, CIPHERXO, DATAXLAT and MACD are not supported.  DES/CDMF token markings are ignored for DATA and key-encrypting keys.

## Key Export

Table 30. Key export required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor  Crypto Express3 Coprocessor	Key types CIPHERXI, CIPHERXL, CIPHERXO, DATAXLAT and MACD are not supported.  DES/CDMF token markings are ignored for DATA and key-encrypting keys.
z196  z114	Crypto Express3 Coprocessor	Key types CIPHERXI, CIPHERXL, CIPHERXO, DATAXLAT and MACD are not supported.  DES/CDMF token markings are ignored for DATA and key-encrypting keys.
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	Key types DATAXLAT and MACD are not supported.  DES/CDMF token markings are ignored for DATA and key-encrypting keys.

## Key Generate (CSNBKGN and CSNEKGN)

Use the key generate callable service to generate either one or two odd parity DES keys of *any* type. The keys can be single-length (8 bytes), double-length (16 bytes), or, in the case of DATA keys, triple-length (24 bytes). The callable service does not produce keys in clear form and all keys are returned in encrypted form. When two keys are generated, each key has the same clear value, although this clear value is not exposed outside the secure cryptographic feature.

Use the key generate callable service to generate an AES key of DATA type. The callable service does not produce AES keys in clear form and all AES keys are returned in encrypted form. Only one AES key is generated.

The callable service name for AMODE (64) invocation is CSNEKGN.

## Format

```
CALL CSNBKGN(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    key_form,  
    key_length,  
    key_type_1,  
    key_type_2,  
    KEK_key_identifier_1,  
    KEK_key_identifier_2,  
    generated_key_identifier_1,  
    generated_key_identifier_2 )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **key\_form**

Direction: Input

Type: Character string

A 4-byte keyword that defines the type of key(s) you want to generate. This parameter also specifies if each key should be returned for either operational, importable, or exportable use. The keyword must be in a 4-byte field, left-justified, and padded with blanks.

The first two characters refer to *key\_type\_1*. The next two characters refer to *key\_type\_2*.

These keywords are allowed: OP, IM, EX, OPIM, OPEX, IMEX, EXEX, OPOP, and IMIM. See Table 31 for their meanings.

If the *key\_form* is OP, EX or IM, the *KEK\_key\_identifier\_2*, *key\_type\_2*, and *generated\_key\_identifier\_2* should be set to NULL.

*Table 31. Key Form values for the Key Generate callable service*

Keyword	Meaning
EX	One key that can be sent to another system.
EXEX	A key pair; both keys to be sent elsewhere, possibly for exporting to two different systems. The key pair has the same clear value.



## Key Generate

Table 31. Key Form values for the Key Generate callable service (continued)

Keyword	Meaning
IM	One key that can be locally imported. The key can be imported onto this system to make it operational at another time.
IMEX	A key pair to be imported; one key to be imported locally and one key to be sent elsewhere. Both keys have the same clear value.
IMIM	A key pair to be imported; both keys to be imported locally at another time.
OP	One operational key. The key is returned to the caller in the key token format. Specify the OP key form when generating AKEKs and AES keys.
OPEX	A key pair; one key that is operational and one key to be sent from this system. Both keys have the same clear value.
OPIM	A key pair; one key that is operational and one key to be imported to the local system. Both keys have the same clear value. On the other system, the external key token can be imported to make it operational.
OPOP	A key pair; normally with different control vector values.

The key forms are defined as follows:

### Operational (OP)

The key value is enciphered under a master key. The result is placed into an internal key token. The key is then operational at the local system. For AKEKs, the result is placed in a skeleton token created by the key token build callable service. AES AKEKs are not supported.

### Importable (IM)

The key value is enciphered under an importer key-encrypting key. The result is placed into an external key token.

### Exportable (EX)

The key value is enciphered under an exporter key-encrypting key. The result is placed into an external key token. The key can then be transported or exported to another system and imported there for use. This key form cannot be used by any ICSF callable service.

The keys are placed into tokens that the *generated\_key\_identifier\_1* and *generated\_key\_identifier\_2* parameters identify.

Valid key type combinations depend on the key form. See Table 37 for valid key combinations.

### key\_length

Direction: Input

Type: Character string

An 8-byte value that defines the length of the key. The keyword must be left-justified and padded on the right with blanks. You must supply one of the key length values in the *key\_length* parameter.

Table 32. Key Length values for the Key Generate callable service

Value	Description	Algorithm
SINGLE or KEYLN8	The key should be a single length (8-byte) key.	DES
SINGLE-R	The key should be a double length (16-byte) key. The two key halves will be the same. This makes the key effectively a single length key.	DES
DOUBLE or KEYLN16	The key should be a double length (16-byte or 128-bit) key	AES or DES
DOUBLE-O	The key should be a double length (16-byte) key. Each of the two key halves will be unique (not the same value).	DES
KEYLN24	The key should be a 24-byte (192-bit) key.	AES or DES
KEYLN32	The key should be a 32-byte (256-bit) key.	AES

**DES Keys:** Double-length (16-byte) keys have an 8-byte left half and an 8-byte right half. Both halves can have identical clear values or not. If you want the same value to be used in both key halves (referred to as replicated key values), specify *key\_length* as SINGLE, SINGLE-R or KEYLN8. If you want different values to be the basis of each key half, specify *key\_length* as DOUBLE, DOUBLE-O or KEYLN16.

Triple-length (24-byte) keys have three 8-byte key parts. This key length is valid for DATA keys only. To generate a triple-length DATA key with three different values to be the basis of each key part, specify *key\_length* as KEYLN24.

Use SINGLE/SINGLE-R if you want to create a DES transport key that you would use to exchange DATA keys with a PCF system. Because PCF does not use double-length transport keys, specify SINGLE so that the effects of multiple encipherment are nullified. When generating an AKEK, the *key\_length* parameter is ignored. The AKEK key length (8-byte or 16-byte) is determined by the skeleton token created by the key token build callable service and provided in the *generated\_key\_identifier\_1* parameter.

**Note:** SINGLE-R and DOUBLE-O are not supported on IBM @server zSeries 900 servers.

**AES Keys:** AES only allows KEYLN16, KEYLN24, KEYLN32. To generate a 128-bit AES key, specify *key\_length* as KEYLN16. For 192-bit AES keys specify *key\_length* as KEYLN24. A 256-bit AES key requires a *key\_length* of KEYLN32. All AES keys are DATA keys.

**Systems with CCFs (with or without PCICCs):** This table shows the valid key lengths for each key type supported by DES keys. An **X** indicates that a key length is permitted for a key type. A **Y** indicates that the key generated will be a double-length key with replicated key values.

## Key Generate

**Note:** When generating a double-length key with replicated key values and the *key\_form* parameter as IMEX, the *KEK\_key\_identifier\_1* parameter must contain a NOCV IMPORTER key-encrypting key either as a key label or an internal key token. Also the CKDS must contain NOCV enablement keys.

Table 33. Key lengths for DES keys - CCF systems

Key Type	Single - KEYLN8	Double - KEYLN16	KEYLN24
MAC	X		
MACVER	X		
DATA	X	X	X
DATAM		X	
DATAMV		X	
EXPORTER	Y	X	
IMPORTER	Y	X	
IKEYXLAT	Y	X	
OKEYXLAT	Y	X	
CIPHER#	X		
DECIPHER#	X		
ENCIPHER#	X		
IPINENC	Y	X	
OPINENC	Y	X	
PINGEN	Y	X	
PINVER	Y	X	
CVARDEC*#	X	X	
CVARENC*#	X	X	
CVARPINE*#	X	X	
CVARXCVL*#	X	X	
CVARXCVR*#	X	X	
DKYGENKY*#	Y	X	
KEYGENKY*#	X	X	

### Notes:

1. \* — key types marked with an asterisk (\*) are requested through the use of the TOKEN keyword and specifying a proper control-vector in a key token
2. # — key types marked with a pound sign (#) require a PCICC

**Cryptographic Coprocessor Systems:** This table shows the valid key lengths for each key type supported by DES keys. An **X** indicates that a key length is permitted for a key type. A **Y** indicates that the key generated will be a double-length key with replicated key values. It is preferred that SINGLE-R be used for this result.

Table 34. Key lengths for DES keys - PCIXCC and CCA Crypto Express coprocessor systems

Key Type	Single - KEYLN8	Single-R	Double - KEYLN16	DOUBLE-O	KEYLN24
MAC	X	X	X	X	
MACVER	X	X	X	X	
DATA	X		X		X

Table 34. Key lengths for DES keys - PCIXCC and CCA Crypto Express coprocessor systems (continued)

DATA*		X	X	X	
DATAM		X	X	X	
DATAMV		X	X	X	
EXPORTER	Y	X	X	X	
IMPORTER	Y	X	X	X	
IKEYXLAT	Y	X	X	X	
OKEYXLAT	Y	X	X	X	
CIPHER	X	X	X	X	
DECIPHER	X	X	X	X	
ENCIPHER	X	X	X	X	
IPINENC	Y	X	X	X	
OPINENC	Y	X	X	X	
PINGEN	Y	X	X	X	
PINVER	Y	X	X	X	
CVARDEC*	X	X	X		
CVARENC*	X	X	X		
CVARPINE*	X	X	X		
CVARXCVL*	X	X	X		
CVARXCVR*	X	X	X		
DKYGENKY*		X	X	X	
KEYGENKY*		X	X	X	
CIPHERXI			X	X	
CIPHERXL			X	X	
CIPHERXO			X	X	

This table shows the valid key lengths for each key type supported by AES keys. An X indicates that a key length is permitted for that key type.

Table 35. Key lengths for AES keys - CCA Crypto Express coprocessor systems

Key Type	128-byte	192-byte	256-byte
AESTOKEN	X	X	X
AESDATA	X	X	X

### key\_type\_1

Direction: Input

Type: Character string

Use the *key\_type\_1* parameter for the first, or only key, that you want generated. The keyword must be left-justified and padded with blanks. Valid type combinations depend on the key form.

The 8-byte keyword for the *key\_type\_1* parameter can be one of the following:

- AESDATA, AESTOKEN, CIPHER, CIPHERXI, CIPHERXL, CIPHERXO, DATA, DATAC, DATAM, DATAMV, DATAXLAT, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER
- or the keyword TOKEN

For information on the meaning of the key types, see Table 3 on page 23.

If *key\_type\_1* is TOKEN, ICSF examines the control vector (CV) field in the *generated\_key\_identifier\_1* parameter to derive the key type. When *key\_type\_1* is

## Key Generate

TOKEN, ICSF does not check for the length of the key for DATA keys. Instead, ICSF uses the *key\_length* parameter to determine the length of the key.

To generate a DES AKEK, specify a *key\_type\_1* of TOKEN. The *generated\_key\_identifier\_1* parameter must be a skeleton token of an AKEK created by the Key Token Build callable service. The token cannot be a partially notarized AKEK or an AKEK key part.

If *key\_type\_1* is AESDATA or AESTOKEN, the key generated will be an AES key of type DATA. When *key\_type\_1* is AESTOKEN, ICSF uses the *key\_length* parameter to determine the length of the key.

See Table 36 and Table 37 for valid key type and key form combinations.

### key\_type\_2

Direction: Input

Type: Character string

Use the *key\_type\_2* parameter for a key pair, which is shown in Table 37 on page 151. The keyword must be left-justified and padded with blanks. Valid type combinations depend on the key form. *key\_type\_2* is only used when DES keys are generated.

The 8-byte keyword for the *key\_type\_2* parameter can be one of the following:

- CIPHER, CIPHERXI, CIPHERXL, CIPHERXO, DATA, DATAC, DATAM, DATAMV, DATAXLAT, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER
- or the keyword TOKEN

For information on the meaning of the key types, see Description of Key Types, Table 3 on page 23.

If *key\_type\_2* is TOKEN, ICSF examines the control vector (CV) field in the *generated\_key\_identifier\_2* parameter to derive the key type. When *key\_type\_2* is TOKEN, ICSF does not check for the length of the key for DATA keys. Instead, ICSF uses the *key\_length* parameter to determine the length of the key.

If only one key is to be generated, *key\_type\_2* and *KEK\_key\_identifier\_2* are ignored.

See Table 36 on page 151 and Table 37 on page 151 for valid key type and key form combinations.

### KEK\_key\_identifier\_1

Direction: Input/Output

Type: String

A 64-byte string of a DES internal key token containing the importer or exporter key-encrypting key, or a key label. If you supply a key label that is less than 64-bytes, it must be left-justified and padded with blanks. *KEK\_key\_identifier\_1* is required for a *key\_form* of IM, EX, IMEX, EXEX, or IMIM.

When *key\_form* OP is used, parameters *KEK\_key\_identifier\_1* and *KEK\_key\_identifier\_2* are ignored. In this case, it is recommended that the parameters are initialized to 64-bytes of X'00'.

If the NOCV bit is on in the internal key token containing the key-encrypting key, the key-encrypting key itself (not the key-encrypting key variant) is used to encipher the generated key. For example, the key has been installed in the

cryptographic key data set through the key generator utility program or the key entry hardware using the NOCV parameter; or you are passing the key-encrypting key in the internal key token with the NOCV bit on and your program is running in supervisor state or key 0-7.

Control vectors are explained in “Control Vector for DES Keys” on page 19 and the NOCV bit is shown in Table 350 on page 837.

KEK\_key\_identifier\_1 cannot be an AES key token or label.

### KEK\_key\_identifier\_2

Direction: Input/Output

Type: String

A 64-byte string of a DES internal key token containing the importer or exporter key-encrypting key, or a key label of an internal token. If you supply a key label that is less than 64-bytes, it must be left-justified and padded with blanks. *KEK\_key\_identifier\_2* is required for a *key\_form* of OPIM, OPEX, IMEX, IMIM, or EXEX. This field is ignored for *key\_form* keywords OP, IM and EX. When *key\_form* OP is used, parameter *KEK\_key\_identifier\_2* is ignored. In this case, it is recommended that the parameter is initialized to 64-bytes of X'00'.

If the NOCV bit is on in the internal key token containing the key-encrypting key, the key-encrypting key itself (not the key-encrypting key variant) is used to encipher the generated key. For example, the key has been installed in the cryptographic key data set through the key generator utility program or the key entry hardware using the NOCV parameter; or you are passing the key-encrypting key in the internal key token with the NOCV bit on and your program is running in supervisor state or in key 0-7.

Control vectors are explained in “Control Vector for DES Keys” on page 19 and the NOCV bit is shown in Table 350 on page 837.

KEK\_key\_identifier\_2 cannot be an AES key token or label.

### generated\_key\_identifier\_1

Direction: Input/Output

Type: String

This parameter specifies either a generated:

- Internal DES or AES key token for an operational key form, or
- External DES key tokens containing a key enciphered under the *KEK\_key\_identifier\_1* parameter.

If you specify a *key\_type\_1* of TOKEN, then this field contains a valid DES token of the key type you want to generate. Otherwise, on input, this parameter must be binary zeros. See *key\_type\_1* for a list of valid key types.

If you specify a *key\_type\_1* of IMPORTER or EXPORTER and a *key\_form* of OPEX, and if the *generated\_key\_identifier\_1* parameter contains a valid DES internal token of the SAME type, the NOCV bit, if on, is propagated to the generated key token.

When generating an AKEK, specify the skeleton key token created by the Key Token Build callable service as input for this parameter.

When *key\_type\_1* parameter is AESDATA, then *generated\_key\_identifier\_1* is ignored. In this case, it is recommended that the parameter be initialized to 64-bytes of X'00'. If you specify a *key\_type\_1* of AESTOKEN, the

## Key Generate

*generated\_key\_identifier\_1* parameter must be an internal AES key token or a clear AES key token. Information in this token can be used to determine the key type:

- The *key\_type\_1* parameter overrides the type in the token.
- The *key\_length* parameter overrides the length value in the generated key token.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *generated\_key\_identifier\_1* will use the default wrapping method unless a skeleton token is supplied as input. If a skeleton token is supplied as input, the wrapping method in the skeleton token will be used.

### **generated\_key\_identifier\_2**

Direction: Input/Output

Type: String

This parameter specifies either a generated:

- internal DES key token or
- external DES key token enciphered under *KEK\_key\_identifier\_2*.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *generated\_key\_identifier\_2* will use the default wrapping method unless a skeleton token is supplied as input. If a skeleton token is supplied as input, the wrapping method in the skeleton token will be used.

## Restrictions

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes — General

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

## Usage Notes — CCF systems

This applies to requests processed on a system with CCFs and only if the request is processed by the CCF. Processing on a PCICC does not cause tokens to be marked.

Internal DATA, IMPORTER and EXPORTER tokens are marked with the system encryption algorithm. No external tokens generated by this service are marked.

When the key form is OP, the token is marked with the system default algorithm. This marking can be overridden by specifying a valid token in the *generated\_key\_identifier\_1* parameter with the marking required.

When the key form is OPEX or OPIM, the operational token is marked with the markings of the key-encrypting key (*KEK\_key\_identifier\_2*). This marking can be overridden by specifying a valid token in the *generated\_key\_identifier\_1* parameter with the marking required.



It is possible to generate an operational DES-marked DATA key on a CDMF-only system or a CDMF-marked DATA key on a DES-only system. However, the Encipher and Decipher callable services fail when you use these keys on the systems where they were generated unless overridden by keyword.

## Usage Notes - Cryptographic Coprocessor Systems

For key types CIPHERXI, CIPHERXL, and CIPHERXO, the key-encrypting keys in the KEK\_key\_identifier\_1 and KEK\_key\_identifier\_2 parameters must have a control vector with the key halves guaranteed unique flag on in the key form bits. An existing key-encrypting key can have its control vector updated using the restrict key attribute callable service.

## Usage Notes — Key type and key form combinations

Table 36 shows the valid key type and key form combinations for a single DES or AES key. Key types marked with an "\*" must be requested through the specification of a proper control vector in a key token and through the use of the TOKEN keyword.

**Note:** Not all keytypes are valid on all hardware. See Table 3 on page 23.

*Table 36. Key Generate Valid Key Types and Key Forms for a Single Key*

Key Type 1	Key Type 2	OP	IM	EX
AESDATA	Not applicable	X		
AESTOKEN	Not applicable	X		
DATA	Not applicable	X	X	X
DATA*	Not applicable	X	X	X
DATAM	Not applicable	X	X	X
DKYGENKY*	Not applicable	X	X	X
KEYGENKY*	Not applicable	X	X	X
MAC	Not applicable	X	X	X
PINGEN	Not applicable	X	X	X

Table 37 shows the valid key type and key form combinations for a DES key pair. Key types marked with an "\*" must be requested through the specification of a proper control vector in a key token and through the use of the TOKEN keyword.

*Table 37. Key Generate Valid Key Types and Key Forms for a Key Pair*

Key Type 1	Key Type 2	OPEX	EXEX	OPIM, OPOP, IMIM	IMEX
CIPHER	CIPHER CIPHERXI CIPHERXL CIPHERXO DECIPHER ENCIPHER	X	X	X	X
CIPHERXI	CIPHER ENCIPHER	E	X	X	E
CIPHERXI	CIPHERXO	E	X		E
CIPHERXL	CIPHER	E	X	X	E

## Key Generate

Table 37. Key Generate Valid Key Types and Key Forms for a Key Pair (continued)

Key Type 1	Key Type 2	OPEX	EXEX	OPIM, OPOP, IMIM	IMEX
CIPHERXL	CIPHERXL	E	X		E
CIPHERXO	CIPHER DECIPHER	E	X	X	E
CIPHERXO	CIPHERXI	E	X		E
CVARDEC*	CVARENC* CVARPINE*	E			E
CVARENC*	CVARDEC* CVARXCVL* CVARXCVR*	E			E
CVARXCVL*	CVARENC*	E			E
CVARXCVR*	CVARENC*	E			E
CVARPINE*	CVARDEC*	E			E
DATA	DATA DATAXLAT	X	X	X	X
DATAAC*	DATAAC*	X	X	X	X
DATAM	DATAM DATAMV	X	X	X	X
DATAXLAT	DATAXLAT	X	X		X
DECIPHER	CIPHER CIPHERXO ENCIPHER	X	X	X	X
DKYGENKY*	DKYGENKY*	X	X	X	X
ENCIPHER	CIPHER CIPHERXI DECIPHER	X	X	X	X
EXPORTER	IKEYXLAT IMPORTER	X	X		X
IKEYXLAT	EXPORTER OKEYXLAT	X	X		X
IMPORTER	EXPORTER OKEYXLAT	X	X		X
IPINENC	OPINENC	X	X	E	X
KEYGENKY*	KEYGENKY*	X	X	X	X
MAC	MAC MACVER	X	X	X	X
OKEYXLAT	IKEYXLAT IMPORTER	X	X		X
OPINENC	IPINENC	X	X	E	X
OPINENC	OPINENC			X	
PINVER	PINGEN	X	X		X
PINGEN	PINVER	X	X		X

If you are running with the Cryptographic Coprocessor Feature and the *key\_form* is IMEX, the *key\_length* is SINGLE, and *key\_type\_1* is IPINENC, OPINENC, PINGEN, IMPORTER, or EXPORTER, you must specify the *KEK\_key\_identifier\_1* parameter as NOCV IMPORTER

If you are running with the Cryptographic Coprocessor Feature and need to use NOCV key-encrypting keys, NOCV-enablement keys must be installed in the CKDS. If you running with a PCIXCC or CCA Crypto Express coprocessor and need to use NOCV key-encrypting keys, you need to enable NOCV IMPORTER and NOCV EXPORTER access control points

If you are running with the Cryptographic Coprocessor Feature and need to generate DATAM and DATAMV keys in the importable form, the ANSI system keys must be installed in the CKDS.

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

*Table 38. Required access control points for Key Generate*

Usage	Access Control Point
The key-form and key-type combinations shown with an 'X' in the Key_Form OP column in Table 36 on page 151.	Key Generate – OP
The key-form and key-type combinations shown with an 'X' in the Key_Form IM column in Table 36 on page 151.	Key Generate – Key set
The key-form and key-type combinations shown with an 'X' in the Key_Form EX column in Table 36 on page 151.	Key Generate - Key set
The key-form and key-type combinations shown with an 'X' in Table 37 on page 151	Key Generate - Key set
The key-form and key-type combinations shown with an 'E' in Table 37 on page 151	Key Generate - Key set extended
The SINGLE-R key-length keyword is specified	Key Generate - SINGLE-R

To use a NOCV IMPORTER key-encrypting key with the key generate service, the **NOCV KEK usage for import-related functions** access control point must be enabled in addition to one or both of the access control points listed.

To use a NOCV EXPORTER key-encrypting key with the key generate service, the **NOCV KEK usage for export-related functions** access control point must be enabled in addition to one or both of the access control points listed.

To use the SINGLE-R rule array keyword, the **Key Generate – SINGLE-R** access control point must be enable.

## Key Generate

If a key-encrypting key identifier is a weaker key than the key being generated, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key or the key-encrypting key is a double-length key.

## Required Hardware

Table 39 on page 155 lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 39. Key generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	<p>OPIM is valid on the Cryptographic Coprocessor Feature for key forms DATA/DATA, DATAM/DATAM and MAC/MAC. All other OPIM key forms are routed to the PCI Cryptographic Coprocessor. In <i>key_form</i> and <i>generated_key_identifier_1</i>, marking of data encryption algorithm bits and token copying are only performed if this service is processed on a Cryptographic Coprocessor Feature. In <i>KEK_key_identifier_2</i> propagation of token markings is only relevant when this service is processed on the Cryptographic Coprocessor Feature. In <i>generated_key_identifier_1</i>, propagation of the NOCV bit is performed only if the service is processed on the Cryptographic Coprocessor Feature.</p> <p>AKEKs are processed on CCFs</p> <p>Key types CIPHERXI, CIPHERXL, CIPHERXO and DATAC is not supported.</p> <p>Secure AES keys are not supported.</p> <p>Key lengths SINGLE-R and DOUBLE-O are not supported</p>
	PCI Cryptographic Coprocessor	<p>ICSF routes the request to a PCI Cryptographic Coprocessor if:</p> <ul style="list-style-type: none"> <li>• OPIM key forms are not DATA/DATA, DATAM/DATAM or MAC/MAC.</li> <li>• The key type specified in <i>key_type_1</i> or <i>key_type_2</i> is not valid for the Cryptographic Coprocessor Feature or if the control vector in a supplied token cannot be processed on the Cryptographic Coprocessor Feature.</li> <li>• A key length of SINGLE-R is specified, or if a key form of OPIM, OPOP or IMIM is specified.</li> <li>• Tokens are not marked with the system encryption algorithm. The NOCV flag is not propagated to key-encrypting keys.</li> </ul> <p>Secure AES keys are not supported.</p>
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	Key types AKEK, CIPHERXI, CIPHERXL, CIPHERXO, and DATAXLAT are not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	<p>Key length DOUBLE-O is not supported</p> <p>Secure AES keys are not supported.</p>

## Key Generate

Table 39. Key generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Key types AKEK, CIPHERXI, CIPHERXL, CIPHERXO, and DATAXLAT are not supported.  Key length DOUBLE-O is not supported  Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor  Crypto Express3 Coprocessor	Key types AKEK, CIPHERXI, CIPHERXL, CIPHERXO, and DATAXLAT are not supported.  Key length DOUBLE-O is not supported  Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).
z196  z114	Crypto Express3 Coprocessor	Key types AKEK, CIPHERXI, CIPHERXL, CIPHERXO, and DATAXLAT are not supported.  Key length DOUBLE-O is not supported
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	Key types AKEK and DATAXLAT are not supported.

---

## Key Generate2 (CSNBKGN2 and CSNEKGN2)

Use the Key Generate2 callable service to generate either one or two keys of any type. This callable service does not produce keys in clear form and all keys are returned in encrypted form. When two keys are generated, each key has the same clear value, although this clear value is not exposed outside the secure cryptographic feature.

This service returns variable-length CCA key tokens and uses the AESKW wrapping method.

This service supports HMAC and AES keys. Operational keys will be encrypted under the AES master key.

The callable service name for AMODE(64) is CSNEKGN2.

## Format

```
CALL CSNBKGN2(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    clear_key_bit_length,
    key_type_1,
    key_type_2,
    key_name_1_length,
    key_name_1,
    key_name_2_length,
    key_name_2,
    user_associated_data_1_length,
    user_associated_data_1,
    user_associated_data_2_length,
    user_associated_data_2,
    key_encrypting_key_identifier_1_length,
    key_encrypting_key_identifier_1,
    key_encrypting_key_identifier_2_length,
    key_encrypting_key_identifier_2,
    generated_key_identifier_1_length,
    generated_key_identifier_1,
    generated_key_identifier_2_length,
    generated_key_identifier_2 )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'FFFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.



## Key Generate2

### rule\_array\_count

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value must be 2.

### rule\_array

Direction: Input

Type: String

The *rule\_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 40. Keywords for Key Generate2 Control Information

Keyword	Meaning
<i>Token algorithm (required)</i>	
HMAC	Specifies to generate an HMAC key token.
AES	Specifies to generate an AES key token.
<i>Key Form (required)</i>	
The first two characters refer to <b>key_type_1</b> . The next two characters refer to <b>key_type_2</b> . See the Usage Notes section for further details.	
EX	One key that can be sent to another system.
EXEX	A key pair; both keys to be sent elsewhere, possibly for exporting to two different systems. Both keys have the same clear value.
IM	One key that can be locally imported. The key can be imported onto this system to make it operational at another time.
IMEX	A key pair to be imported; one key to be imported locally and one key to be sent elsewhere. Both keys have the same clear value.
IMIM	A key pair to be imported; both keys to be imported locally at another time. Both keys have the same clear value.
OP	One operational key. The key is returned to the caller in operational form to be used locally.
OPEX	A key pair; one key that is operational and one key to be sent elsewhere. Both keys have the same clear value.
OPIM	A key pair; one key that is operational and one key to be imported locally at another time. Both keys have the same clear value.
OPOP	A key pair; either with the same key type with different associated data or complementary key types. Both keys have the same clear value.

### clear\_key\_bit\_length

Direction: Input

Type: Integer

The size (in bits) of the key to be generated.

- For the HMAC algorithm, this is a value between 80 and 2048, inclusive.
- For the AES algorithm, this is a value of 128, 192, or 256.

When *key\_type\_1* or *key\_type\_2* is TOKEN, this value overrides the key length contained in *generated\_key\_identifier\_1* or *generated\_key\_identifier\_2*, respectively.

### key\_type\_1

Direction: Input

Type: String

Use the *key\_type\_1* parameter for the first, or only, key that you want generated. The keyword must be left-justified and padded with blanks. Valid type combinations depend on the key form, and are documented in Table 43 on page 163 and Table 44 on page 163.

The 8-byte keyword for the *key\_type\_1* parameter can be one of the following:

*Table 41. Keywords and associated algorithms for key\_type\_1 parameter*

Keyword	Algorithm
CIPHER	AES
EXPORTER	AES
IMPORTER	AES
MAC	HMAC
MACVER	HMAC
Specify the keyword TOKEN when supplying a key token in the <i>generated_key_identifier_1</i> parameter.	

If *key\_type\_1* is TOKEN, the associated data in the *generated\_key\_identifier\_1* parameter is examined to derive the key type.

### key\_type\_2

Direction: Input

Type: String

Use the *key\_type\_2* parameter for a key pair, which is shown in Table 44 on page 163. The keyword must be left-justified and padded with blanks. Valid type combinations depend on the key form.

The 8-byte keyword for the *key\_type\_2* parameter can be one of the following:

*Table 42. Keywords and associated algorithms for key\_type\_2 parameter*

Keyword	Algorithm
CIPHER	AES
EXPORTER	AES
IMPORTER	AES
MAC	HMAC
MACVER	HMAC
Specify the keyword TOKEN when supplying a key token in the <i>generated_key_identifier_2</i> parameter.	

If *key\_type\_2* is TOKEN, the associated data in the *generated\_key\_identifier\_2* parameter is examined to derive the key type.

## Key Generate2

When only one key is being generated, this parameter is ignored.

### **key\_name\_1\_length**

Direction: Input

Type: Integer

The length of the *key\_name* parameter for *generated\_key\_identifier\_1*. Valid values are 0 and 64 bytes.

### **key\_name\_1**

Direction: Input

Type: String

A 64-byte key store label to be stored in the associated data structure of *generated\_key\_identifier\_1*.

### **key\_name\_2\_length**

Direction: Input

Type: Integer

The length of the *key\_name* parameter for *generated\_key\_identifier\_2*. Valid values are 0 and 64 bytes.

When only one key is being generated, this parameter is ignored.

### **key\_name\_2**

Direction: Input

Type: String

A 64-byte key store label to be stored in the associated data structure of *generated\_key\_identifier\_2*.

When only one key is being generated, this parameter is ignored.

### **user\_associated\_data\_1\_length**

Direction: Input

Type: Integer

The length of the user-associated data parameter for *generated\_key\_identifier\_1*. The valid values are 0 to 255 bytes.

### **user\_associated\_data\_1**

Direction: Input

Type: String

User-associated data to be stored in the associated data structure for *generated\_key\_identifier\_1*.

### **user\_associated\_data\_2\_length**

Direction: Input

Type: Integer

The length of the user-associated data parameter for *generated\_key\_identifier\_2*. The valid values are 0 to 255 bytes.

When only one key is being generated, this parameter is ignored.

### **user\_associated\_data\_2**

Direction: Input

Type: String

User associated data to be stored in the associated data structure for *generated\_key\_identifier\_2*.

When only one key is being generated, this parameter is ignored.

### **key\_encrypting\_key\_identifier\_1\_length**

Direction: Input

Type: Integer

The length of the buffer for *key\_encrypting\_key\_identifier\_1* in bytes. When the Key Form rule is OP, OPOP, OPIM, or OPEX, this length must be zero. When the Key Form rule is EX, EXEX, IM, IMEX, or IMIM, the value must be between the actual length of the token and 725 bytes when *key\_encrypting\_key\_identifier\_1* is a token.

The value must be 64 bytes when *key\_encrypting\_key\_identifier\_1* is a label.

### **key\_encrypting\_key\_identifier\_1**

Direction: Input/Output

Type: String

When *key\_encrypting\_key\_identifier\_1\_length* is zero, this parameter is ignored. Otherwise, *key\_encrypting\_key\_identifier\_1* contains an internal key token containing the AES importer or exporter key-encrypting key, or a key label.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

### **key\_encrypting\_key\_identifier\_2\_length**

Direction: Input

Type: Integer

The length of the buffer for *key\_encrypting\_key\_identifier\_2* in bytes. When the Key Form rule is OPOP, this length must be zero. When the Key Form rule is EXEX, IMEX, IMIM, OPIM, or OPEX, the value must be between the actual length of the token and 725 when *key\_encrypting\_key\_identifier\_2* is a token. The value must be 64 when *key\_encrypting\_key\_identifier\_2* is a label.

When only one key is being generated, this parameter is ignored.

### **key\_encrypting\_key\_identifier\_2**

Direction: Input/Output

Type: String

When *key\_encrypting\_key\_identifier\_2\_length* is zero, this parameter is ignored. Otherwise, *key\_encrypting\_key\_identifier\_2* contains an internal key token containing the AES importer or exporter key-encrypting key, or a key label.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

When only one key is being generated, this parameter is ignored.

### **generated\_key\_identifier\_1\_length**

Direction: Input/Output

Type: Integer

On input, the length of the buffer for the *generated\_key\_identifier\_1* parameter in bytes. The maximum value is 900 bytes.

On output, the parameter will hold the actual length of the *generated\_key\_identifier\_1*.

## Key Generate2

### **generated\_key\_identifier\_1**

Direction: Input/Output

Type: String

The buffer for the first generated key token.

On input, if you specify a *key\_type\_1* of TOKEN, then the buffer contains a valid key token of the key type you want to generate. The key token must be left justified in the buffer. See *key\_type\_1* for a list of valid key types.

On output, the buffer contains the generated key token.

### **generated\_key\_identifier\_2\_length**

Direction: Input/Output

Type: Integer

On input, the length of the buffer for the *generated\_key\_identifier\_2* in bytes. The maximum value is 900 bytes.

On output, the parameter will hold the actual length of the *generated\_key\_identifier\_2*.

When only one key is being generated, this parameter is ignored.

### **generated\_key\_identifier\_2**

Direction: Input/Output

Type: String

The buffer for the second generated key token.

On input, if you specify a *key\_type\_2* of TOKEN, then the buffer contains a valid key token of the key type you want to generate. The key token must be left justified in the buffer. See *key\_type\_2* for a list of valid key types.

On output, the buffer contains the generated key token.

When only one key is being generated, this parameter is ignored.

## Usage Notes

The key forms are defined as follows:

### **Operational (OP)**

The key value is enciphered under a master key. The result is placed into an internal key token. The key is then operational at the local system.

### **Importable (IM)**

The key value is enciphered under an importer key-encrypting key. The result is placed into an external key token. The corresponding *key\_encrypting\_key\_identifier\_x* parameter must contain an AES IMPORTER key token or label.

### **Exportable (EX)**

The key value is enciphered under an exporter key-encrypting key. The result is placed into an external key token. The corresponding *key\_encrypting\_key\_identifier\_x* parameter must contain an AES EXPORTER key token or label.

These tables list the valid key type and key form combinations.

Table 43. Key Generate2 valid key type and key form for one key

key_type_1	Key Form OP, IM, EX
CIPHER	X
MAC	X

Table 44. Key Generate2 Valid key type and key forms for two keys

key_type_1	key_type_2	Key Form OPOP, OPIM, IMIM	Key Form OPEX, EXEX, IMEX
CIPHER	CIPHER	X	X
MAC	MAC	X	X
MAC	MACVER	X	X
MACVER	MAC	X	X
IMPORTER	EXPORTER		X
EXPORTER	IMPORTER		X

For AES CIPHER keys, there further restrictions for key usage bits when TOKEN is used in place of the key type. This table lists the allowable combinations of key-usage bits in the skeleton token supplied in generated key identifier parameters. CIPHER is the default which has the ENCRYPT and DECRYPT bits on in the usage field. CIPHERe has only the ENCRYPT bit on in the usage field. CIPHERd has only the DECRYPT bit on in the usage field. Adding x to either name means the TRANSLAT bit is on in the usage field for that key. (For example, CIPHERex means a variable length token with the ENCRYPT and TRANSLAT bits turned on.)

Table 45. Key Generate2 Valid key forms for CIPHER keys

generated_key_identifier_1 token	generated_key_identifier_2 token	Key Form OPOP, OPIM, IMIM	Key Form OPEX	Key Form EXEX	Key Form IMEX
CIPHER	CIPHER CIPHERd CIPHERe	X	X	X	X
CIPHERd	CIPHER CIPHERe	X	X	X	X
CIPHERe	CIPHER CIPHERd	X	X	X	X
CIPHER	CIPHERdx CIPHERedx CIPHERex	X	X	X	X
CIPHERd	CIPHERex	X	X	X	X
CIPHERe	CIPHERdx	X	X	X	X
CIPHERdx	CIPHER	X	E	X	E
CIPHERedx	CIPHER	X	E	X	E
CIPHERex	CIPHER CIPHERd	X	E	X	E
CIPHERdx	CIPHERe	X	E	X	E
CIPHERedx	CIPHERedx		E	X	E

## Key Generate2

Table 45. Key Generate2 Valid key forms for CIPHER keys (continued)

generated_key_identifier_1 token	generated_key_identifier_2 token	Key Form OPOP, OPIM, IMIM	Key Form OPEX	Key Form EXEX	Key Form IMEX
CIPHERdx	CIPHERex		E	X	E
CIPHERex	CIPHERdx		E	X	E

The strength of the key-encrypting key used to wrap a generated key will affect the results of the service. The resulting return code and reason code when using a key-encrypting key that is weaker than the key being generated depends on the **Prohibit weak wrapping - Transport keys** and **Warn when weak wrap - Transport keys** access control points:

- If the **Prohibit weak wrapping - Transport keys** access control point is disabled, the key strength requirement will not be enforced. Using a weaker key will result in return code 0 with a non-zero reason code if the **Warn when weak wrap - Transport keys** access control point is enabled. Otherwise, a reason code of zero will be returned.
- If the **Prohibit weak wrapping - Transport keys** access control point is enabled, the key strength requirement will be enforced, and attempting to use a weaker key will result in return code 8.

For AES keys, the AES KEK must be at least as strong as the key being generated to be considered sufficient strength.

For HMAC keys, the AES KEK must be sufficient strength as described in the following table.

Table 46. AES KEK strength required for generating an HMAC key under an AES KEK

Key-usage field 2 in the HMAC key contains	Minimum strength of AES KEK to adequately protect the HMAC key
SHA-256, SHA-384, SHA-512	256 bits
SHA-224	192 bits
SHA-1	128 bits

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

Table 47. Required access control points for Key Generate2

Access Control Point	Function control
Key Generate2 – OP	Key Form OP, EX, IM
Key Generate2 – Key set	The key-form and key-type combinations shown with an X in Table 44 on page 163 and Table 45 on page 163
Key Generate2 — key set extended	The key-form and key-type combinations shown with an E in Table 45 on page 163
Prohibit weak key wrapping — Transport keys	Prohibit wrapping a key with a weaker key

Table 47. Required access control points for Key Generate2 (continued)

Access Control Point	Function control
Warn when weak wrap — Transport keys	Issue a non-zero reason code when using a weak wrapping key

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 48. Key Generate2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This service is not supported.
IBM @server zSeries 990		This service is not supported.
IBM @server zSeries 890		
IBM System z9 EC		This service is not supported.
IBM System z9 BC		
IBM System z10 EC	Crypto Express2 Coprocessor	This service is not supported.
IBM System z10 BC	Crypto Express3 Coprocessor	This service is not supported.
z196	Crypto Express3 Coprocessor	AES key support require the Sep. 2011 or later licensed internal code (LIC).
z114		HMAC key support requires the Nov. 2010 or later licensed internal code (LIC).
IBM zEnterprise EC12	Crypto Express3 Coprocessor	
	Crypto Express4 Coprocessor	

## Key Import (CSNBKIM and CSNEKIM)

Use the key import callable service to reencipher a key (except an AKEK) from encryption under an importer key-encrypting key to encryption under the master key. The reenciphered key is in operational form.

Choose one of these options:

- Specify the *key\_type* parameter as TOKEN and specify the external key token in the *source\_key\_identifier* parameter. The key type information is determined from the control vector in the external key token.
- Specify a key type in the *key\_type* parameter and specify an external key token in the *source\_key\_identifier* parameter. The specified key type must be compatible with the control vector in the external key token.



## Key Import

- Specify a valid key type in the *key\_type* parameter and a null key token in the *source\_key\_identifier* parameter. The default control vector for the *key\_type* specified will be used to process the key.

For DATA keys, this service generates a key of the same length as that contained in the input token.

The callable service name for AMODE(64) invocation is CSNEKIM.

## Format

```
CALL CSNBKIM(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    key_type,  
    source_key_identifier,  
    importer_key_identifier,  
    target_key_identifier )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **key\_type**

Direction: Input

Type: Character string

The type of key you want to reencipher under the master key. Specify an 8-byte keyword or the keyword TOKEN. The keyword must be left-justified and padded on the right with blanks.

If the key type is TOKEN, ICSF determines the key type from the control vector (CV) field in the external key token provided in the *source\_key\_identifier* parameter.

TOKEN is never allowed when the *importer\_key\_identifier* is NOCV.

Supported *key\_type* values are CIPHER, CIPHERXI, CIPHERXL, CIPHERXO, DATA, DATAM, DATAMV, DATAXLAT, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, MAC, MACVER, OKEXLAT, OPINENC, PINGEN and PINVER. Use *key\_type* TOKEN for all other key types.

For information on the meaning of the key types, see Table 3 on page 23.

We recommend using key type of TOKEN when importing double-length MAC and MACVER keys.

### **source\_key\_identifier**

Direction: Input

Type: String

The key you want to reencipher under the master key. The parameter is a 64-byte field for the enciphered key to be imported containing either an external key token or a null key token. If you specify a null token, the token is all binary zeros, except for a key in bytes 16-23 or 16-31, or in bytes 16-31 and 48-55 for triple-length DATA keys. Refer to Table 353 on page 841.

If key type is TOKEN, this field may not specify a null token.

This service supports the no-export function in the CV.

### **importer\_key\_identifier**

Direction: Input/Output

Type: String

The importer key-encrypting key that the key is currently encrypted under. The parameter is a 64-byte area containing either the key label of the key in the cryptographic key data set or the internal key token for the key. If you supply a key label that is less than 64-bytes, it must be left-justified and padded with blanks.

**Note:** If you specify a NOCV importer in the *importer\_key\_identifier* parameter, the key to be imported must be enciphered under the importer key itself.

### **target\_key\_identifier**

Direction: Input/Output

Type: String

This parameter is the generated reenciphered key. The parameter is a 64-byte area that receives the internal key token for the imported key.

If the imported key TYPE is IMPORTER or EXPORTER and the token key TYPE is the same, the *target\_key\_identifier* parameter changes direction to both input and output. If the application passes a valid internal key token for an IMPORTER or EXPORTER key in this parameter, the NOCV bit is propagated to the imported key token.

## Key Import

**Note:** Propagation of the NOCV bit is performed only if the service is processed on a Cryptographic Coprocessor Feature or on a PCIXCC or CCA Crypto Express coprocessor.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *target\_key\_identifier* will use the default wrapping method unless a skeleton token is supplied as input. If a skeleton token is supplied as input, the wrapping method in the skeleton token will be used.

## Restrictions

For existing TKE V3.1 (or higher) users, you may have to explicitly enable new access control points. Current applications will fail if they use an equal key halves importer to import a key with unequal key halves. You must have access control point 'Key Import - Unrestricted' explicitly enabled.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes — General

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Use of NOCV keys are controlled by an access control point in the PCIXCC. Creation of NOCV key-encrypting keys is only available for standard IMPORTERS and EXPORTERS.

## Usage Notes — CCF Systems

The key import callable service cannot be used to import ANSI key-encrypting keys. For information on importing these types of keys, refer to “ANSI X9.17 Key Import (CSNAKIM and CSNGKIM)” on page 692. To use NOCV key-encrypting keys or to import DATAM or DATAMV keys, NOCV-enablement keys must be installed in the CKDS.

This service will mark an imported KEK as a NOCV-KEK by supplying a valid IMPORTER or EXPORTER token in the *target\_key\_identifier* field with the NOCV-KEK flag enabled. The type of the token must match the key type of the imported key.

This service will mark DATA and key-encrypting key tokens with the system encryption algorithm if the request is processed on the CCF. The service propagates the data encryption algorithm mark on the operational KEK unless token copying overrides this:

- The imported token is marked with the DES or CDMF encryption algorithm marks of the KEK token
- The imported token is marked with the system's default encryption algorithm when the KEK is marked SYS-ENC
- To override the encryption algorithm marks of the KEK, supply a valid token in the *target\_key\_identifier* field of the same key type being imported. The mark of the *target\_key\_identifier* token are used to mark the imported key token.

Key Import operations which specify a NOCV key-encrypting key as either the importer key or the target and also specify a source or key-encrypting key which contains a control vector not supported by the Cryptographic Coprocessor Feature will fail.

## Usage Notes — Cryptographic Coprocessor Systems

Use of NOCV keys are controlled by an access control point in the PCIXCC or CCA Crypto Express coprocessor.

This service will mark an imported KEK as a NOCV-KEK:

- If a token is supplied in the target token field, it must be a valid importer or exporter token. If the token fails token validation, processing continues, but the NOCV flag will not be copied
- The source token (key to be imported) must be a importer or exporter with the default control vector.
- If the target token is valid and the NOCV flag is on and the source token is valid and the control vector of the target token is exactly the same as the source token, the imported token will have the NOCV flag set on.
- If the target token is valid and the NOCV flag is on and the source token is valid and the control vector of the target token is NOT exactly the same as the source token, a return code will be given.
- All other scenarios will complete successfully, but the NOCV flag will not be copied

The software bit used to mark the imported token with export prohibited is not supported on a PCIXCC or CCA Crypto Express coprocessor. The internal token for an export prohibited key will have the appropriate control vector that prohibits export.

For key types CIPHERXI, CIPHERXL, and CIPHERXO, the key-encrypting key in the importer\_key\_identifier parameter must have a control vector with the key halves guaranteed unique flag on in the key form bits. An existing key-encrypting key can have its control vector updated using the restrict key attribute callable service.

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

*Table 49. Required access control points for Key Import*

Access Control Point	Restrictions
Key Import	None
Key Import - Unrestricted	Key-encrypting key may have equal key halves

To use a NOCV key-encrypting key with the key import service, the **NOCV KEK usage for import-related functions** access control point must be enabled in addition to one or both of the access control points listed.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 50. Key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	<p>Propagation of token markings is only relevant when this service is processed on the Cryptographic Coprocessor Feature.</p> <p>If the <i>key_type</i> is MACD or IMP-PKA, the control vectors of supplied internal tokens must all be supported by the Cryptographic Coprocessor Feature, since processing for these key types will not be routed to a PCI Cryptographic Coprocessor.</p> <p>Key types CIPHERXI, CIPHERXL, CIPHERXO, DATAC are not supported.</p> <p>Key type CIPHER, DECIPHER and ENCIPHER require a PCICC.</p>
	PCI Cryptographic Coprocessor	<p>ICSF routes the request to a PCI Cryptographic Coprocessor if:</p> <ul style="list-style-type: none"> <li>The <i>key_type</i> cannot be processed on the Cryptographic Coprocessor Feature.</li> <li>The control vector of the <i>source_key_identifier</i> or the <i>importer_key_identifier</i> cannot be processed on the Cryptographic Coprocessor Feature.</li> </ul> <p>Key types CIPHERXI, CIPHERXL, and CIPHERXO are not supported.</p>
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	Key types CIPHERXI, CIPHERXL, CIPHERXO and DATAXLAT are not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	DES and CDMF markings are not made on DATA and key-encrypting keys and are ignored on the IMPORTER key-encrypting key. IMP-PKA keys are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	<p>Key types CIPHERXI, CIPHERXL, CIPHERXO and DATAXLAT are not supported.</p> <p>DES and CDMF markings are not made on DATA and key-encrypting keys and are ignored on the IMPORTER key-encrypting key. IMP-PKA keys are not supported.</p>

Table 50. Key import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor  Crypto Express3 Coprocessor	Key types CIPHERXI, CIPHERXL, CIPHERXO and DATAXLAT are not supported.  DES and CDMF markings are not made on DATA and key-encrypting keys and are ignored on the IMPORTER key-encrypting key. IMP-PKA keys are not supported.
z196  z114	Crypto Express3 Coprocessor	Key types CIPHERXI, CIPHERXL, CIPHERXO and DATAXLAT are not supported.  DES and CDMF markings are not made on DATA and key-encrypting keys and are ignored on the IMPORTER key-encrypting key. IMP-PKA keys are not supported.
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	Key type DATAXLAT is not supported.  DES and CDMF markings are not made on DATA and key-encrypting keys and are ignored on the IMPORTER key-encrypting key. IMP-PKA keys are not supported.

## Key Part Import (CSNBKPI and CSNEKPI)

Use the key part import callable service to combine, by exclusive ORing, the clear key parts of any key type and return the combined key value either in an internal token or as an update to the CKDS.

Prior to using the key part import service for the first key part, you must use the key token build service to create the internal key token into which the key will be imported. Subsequent key parts are combined with the first part in internal token form or as a label from the CKDS.

The preferred way to specify key parts is FIRST, ADD-PART, and COMPLETE in the *rule\_array*. Only when the combined key parts have been marked as COMPLETE can the key token be used in any other service. Key parts can also be specified as FIRST, MIDDLE, or LAST in the *rule\_array*. ADD-PART or MIDDLE can be executed multiple times for as many key parts as necessary. Only when the LAST part has been combined can the key token be used in any other service.

New applications should employ the ADD-PART and COMPLETE keywords in lieu of the MIDDLE and LAST keywords in order to ensure a separation of responsibilities between someone who can add key-part information and someone who can declare that appropriate information has been accumulated in a key.

The key part import callable service can also be used to import a key without using key parts. Call the key part import service FIRST with key part value X'0000...' then call the key part import service LAST with the complete value.

## Key Part Import

Keys created via this service have odd parity. The FIRST key part is adjusted to odd parity. All subsequent key parts are adjusted to even parity prior to being combined.

The callable service name for AMODE(64) invocation is CSNEKPI.

## Format

```
CALL CSNBKPI(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_part,  
    key_identifier)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value must be 1 or 2.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

*Table 51. Keywords for Key Part Import Control Information*

Keyword	Meaning
<b>Key Part (Required)</b>	
FIRST	This keyword specifies that an initial key part is being entered. The callable service returns this key-part encrypted by the master key in the key token that you supplied.
ADD-PART	This keyword specifies that additional key-part information is provided.
COMPLETE	This keyword specifies that the key-part bit shall be turned off in the control vector of the key rendering the key fully operational. Note that no key-part information is added to the key with this keyword.
MIDDLE	This keyword specifies that an intermediate key part, which is neither the first key part nor the last key part, is being entered. Note that the command control point for this keyword is the same as that for the LAST keyword and different from that for the ADD-PART keyword.
LAST	This keyword specifies that the last key part is being entered. The key-part bit is turned off in the control vector.
<b>Key Wrapping Method (optional)</b>	
USECONFIG	Specifies that the system default configuration should be used to determine the wrapping method. This is the default keyword.  The system default key wrapping method can be specified using the DEFAULTWRAP parameter in the installation options data set. See the <i>z/OS Cryptographic Services ICSF System Programmer's Guide</i> .
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens.

### key\_part

Direction: Input

Type: String

A 16-byte field containing the clear key part to be entered. If the key is a single-length key, the key part must be left-justified and padded on the right with zeros. This field is ignored if COMPLETE is specified.

### key\_identifier

Direction: Input/Output

Type: String



## Key Part Import

A 64-byte field containing an internal token or a label of an existing CKDS record. If *rule\_array* is FIRST, this field is the skeleton of an internal token of a single- or double-length key with the KEY-PART marking. If *rule\_array* is MIDDLE or LAST, this is an internal token or the label of a CKDS record of a partially combined key. Depending on the input format, the accumulated partial or complete key is returned as an internal token or as an updated CKDS record. The returned *key\_identifier* will be encrypted under the current master key.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *key\_identifier* will use the default method unless a rule array keyword overriding the default for the FIRST key part is specified. When the *key\_identifier* is an existing token, the same wrapping method as the existing token will be used.

## Restrictions

If a label is specified on *key\_identifier*, the label must be unique. If more than one record is found, the service fails.

For existing TKE V3.1 (or higher) users, you may have to explicitly enable new access control points. You must have access control point 'Key Part Import - Unrestricted' explicitly enabled. Otherwise, current applications will fail with either of these conditions:

- the first 8 bytes of key identifier is different than the second 8 bytes AND the first 8 bytes of the combined key are the same as the last second 8 bytes
- the first 8 bytes of key identifier is the same as the second 8 bytes AND the first 8 bytes of the combined key are different than the second 8 bytes.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes

If you are running with the Cryptographic Coprocessor Feature, this service requires that the ANSI system keys be installed on the CKDS.

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

Table 52. Required access control points for Key Part Import

Rule array keyword	Access control point
FIRST	Key Part Import - first key part
MIDDLE or LAST	Key Part Import - middle and last
ADD-PART	Key Part Import - ADD-PART
COMPLETE	Key Part Import - COMPLETE
WRAP-ECB or WRAP-ENH and default key-wrapping method setting does not match keyword	Key Part Import - Allow wrapping override keywords

A “replicated key-halves” key (both cleartext halves of a double-length key are equal) is not as secure as a double-length key with key halves that are not the same. The key part import service verb enforces the key-halves restriction documented above when the **Key Part Import - Unrestricted** access control point is disabled in the ICSF role.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 53. Key part import required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	Only key type AKEK is supported  ENH-ONLY, USECONFIG, WRAP-ENC and WRAP-ECB not supported.
	PCI Cryptographic Coprocessor	ICSF routes all requests to the PCI Cryptographic Coprocessor except for key type of AKEK. AKEK is always processed on the Cryptographic Coprocessor Feature.  Key type AKEK is not supported.  ENH-ONLY, USECONFIG, WRAP-ENC and WRAP-ECB not supported.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	AKEK key types are not supported.  ENH-ONLY, USECONFIG, WRAP-ENC and WRAP-ECB not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	AKEK key types are not supported.  ENH-ONLY, USECONFIG, WRAP-ENC and WRAP-ECB not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	AKEK key types are not supported.  ENH-ONLY, USECONFIG, WRAP-ENC and WRAP-ECB not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	AKEK key types are not supported.  ENH-ONLY, USECONFIG, WRAP-ENC and WRAP-ECB not supported.
	Crypto Express3 Coprocessor	Enhanced key token wrapping not supported.
z196 z114	Crypto Express3 Coprocessor	AKEK key types are not supported.
IBM zEnterprise EC12	Crypto Express3 Coprocessor	AKEK key types are not supported.
	Crypto Express4 Coprocessor	

## Related Information

This service is consistent with the Transaction Security System key part import verb.

---

### Key Part Import2 (CSNBKPI2 and CSNEKPI2)

Use the Key Part Import2 callable service to combine, by exclusive ORing, the clear key parts of any key type and return the combined key value either in a variable-length internal token or as an update to the CKDS.

Prior to using the key part import2 service for the first key part, you must use the Key Token Build2 service to create the internal key token into which the key will be imported. Subsequent key parts are combined with the first part in internal token form or as a label from the CKDS.

On each call to Key Part Import2 (except with the COMPLETE keyword), specify the number of bits to use for the clear key part. Place the clear key part in the *key\_part* parameter, and specify the number of bits using the *key\_part\_length* variable. Any extraneous bits of *key\_part* data will be ignored.

Consider using the Key Test2 callable service to ensure a correct key value has been accumulated prior to using the COMPLETE option to mark the key as fully operational.

The callable service name for AMODE(64) is CSNEKPI2.

### Format

```
CALL CSNBKPI2(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_part_bit_length,  
    key_part,  
    key_identifier_length,  
    key_identifier)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

**exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value may be 2 or 3.

**rule\_array**

Direction: Input

Type: Integer

The *rule\_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 54. Keywords for Key Part Import2 Control Information

Keyword	Meaning
<i>Token Algorithm (Required)</i>	
HMAC	Specifies to import an HMAC key token.
AES	Specifies to import an AES key token.
<i>Key Part (One required)</i>	
FIRST	This keyword specifies that an initial key part is being entered. The callable service returns this key-part encrypted by the master key in the key token that you supplied.
ADD-PART	This keyword specifies that additional key-part information is provided.
COMPLETE	This keyword specifies that the key-part bit shall be turned off in the control vector of the key rendering the key fully operational. Note that no key-part information is added to the key with this keyword.
<i>Split Knowledge (One required). Use only with FIRST keyword.</i>	
MIN3PART	Specifies that the key must be entered in at least three parts.
MIN2PART	Specifies that the key must be entered in at least two parts.
MIN1PART	Specifies that the key must be entered in at least one part.

**key\_part\_bit\_length**

Direction: Input

Type: Integer

## Key Part Import2

The length of the clear key in bits. This indicates the bit length of the key supplied in the *key\_part* field. For FIRST and ADD-PART keywords, valid values are 80 to 2048 for HMAC keys or 128, 192, or 256 for AES keys. The value must be 0 for the COMPLETE keyword.

### **key\_part**

Direction: Input

Type: String

This parameter is the clear key value to be applied. The key part must be left-justified. This parameter is ignored if COMPLETE is specified.

### **key\_identifier\_length**

Direction: Input/Output

Type: Integer

On input, the length of the buffer for the *key\_identifier* parameter. For labels, the value is 64 bytes. The *key\_identifier* must be left justified in the buffer. The buffer must be large enough to receive the updated token. The maximum value is 725 bytes. The output token will be longer when the first key part is imported.

On output, the actual length of the token returned to the caller. For labels, the value will be 64.

### **key\_identifier**

Direction: Input/Output

Type: String

The parameter containing an internal token or a 64-byte label of an existing CKDS record. If the Key Part rule is FIRST, the key is a skeleton token. If the Key Part rule is ADD-PART, this is an internal token or the label of a CKDS record of a partially combined key. Depending on the input format, the accumulated partial or complete key is returned as an internal token or as an updated CKDS record. The returned *key\_identifier* will be encrypted under the current master key.

## Usage Notes

On each call to Key Part Import2, also specify a rule-array keyword to define the service action: FIRST, ADD-PART, or COMPLETE.

- With the FIRST keyword, the input key-token must be a skeleton token (no key material).
- With the ADD-PART keyword, the service exclusive-ORs the clear key-part with the key value in the input key-token. The key remains incomplete in the updated key token returned from the service.
- With the COMPLETE keyword, the KEY-PART bit is set off in the updated key token that is returned from the service. The *key\_part\_bit\_length* parameter must be set to zero.

## Access Control Points

The following table shows the access control points in the default role that control the function of this service.

Table 55. Required access control points for Key Part Import2

Rule array keywords	Access control point
ADD-PART	Key Part Import2 - Add second of three or more key parts
ADD-PART	Key Part Import2 - Add last required key part
ADD-PART	Key Part Import2 - Add optional key part
COMPLETE	Key Part Import2 - Complete key
FIRST MIN3PART	Key Part Import2 - Load first key part, require 3 key parts
FIRST MIN2PART	Key Part Import2 - Load first key part, require 2 key parts
FIRST MIN1PART	Key Part Import2 - Load first key part, require 1 key parts

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 56. Key Part Import2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This service is not supported.
IBM @server zSeries 990		This service is not supported.
IBM @server zSeries 890		
IBM System z9 EC		This service is not supported.
IBM System z9 BC		
IBM System z10 EC	Crypto Express2 Coprocessor	This service is not supported.
IBM System z10 BC	Crypto Express3 Coprocessor	This service is not supported.
z196 z114	Crypto Express3 Coprocessor	AES key support requires the Sep. 2011 or later licensed internal code (LIC).  HMAC key support requires the Nov. 2010 or later licensed internal code (LIC).
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## Key Test (CSNBKYT and CSNEKYT)

Use the key test callable service to generate or verify a secure, cryptographic verification pattern for keys. The key to test can be in the clear or encrypted under the master key. Keywords in the *rule\_array* specify whether the callable service generates or verifies a verification pattern.

## Key Test

DES keys use the algorithm defined in “DES Algorithm (single- and double-length keys)” on page 956 as the default algorithm (except for triple-length DATA keys). When generating a verification pattern, the service generates a random number and calculates the verification pattern. The random number and verification pattern are returned to the caller. When verifying a key, the random number and key are used to verify the verification pattern.

AES keys use the SHA-256 algorithm as the default algorithm. An 8-byte verification pattern is generated for the key specified. The random number parameter is not used.

The optional ENC-ZERO algorithm can be used with any key. A 4-byte verification pattern is generated. The random number parameter is not used.

CSNBKYT is consistent with the Transaction Security System verb of the same name. If you generate a key on the Transaction Security System, you can verify it on ICSF and vice versa.

See “Key Test Extended (CSNBKYTX and CSNEKTX)” on page 188 to verify the value of a DES key encrypted using a KEK.

The callable service name for AMODE(64) invocation is CSNEKYT.

## Format

```
CALL CSNBKYT(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_identifier,  
    random_number,  
    verification_pattern)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

#### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value can be 2, 3 or 4.

#### **rule\_array**

Direction: Input

Type: String

Keywords provide control information to the callable service. Table 57 lists the keywords. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

*Table 57. Keywords for Key Test Control Information*

Keyword	Meaning
<b>Key or key part rule (one keyword required)</b>	
CLR-A128	Process a 128-bit AES clear key.
CLR-A192	Process a 192-bit AES clear key.
CLR-A256	Process a 256-bit AES clear key.
KEY-CLR	Specifies the key supplied in <i>key_identifier</i> is a single-length clear key.
KEY-CLRD	Specifies the key supplied in <i>key_identifier</i> is a double-length clear key.
KEY-ENC	Specifies the key supplied in <i>key_identifier</i> is a single-length encrypted key.
KEY-ENCD	Specifies the key supplied in <i>key_identifier</i> is a double-length encrypted key.
TOKEN	Process an AES clear or encrypted key token.
<b>Process Rule (one keyword required)</b>	
GENERATE	Generate a verification pattern for the key supplied in <i>key_identifier</i> .
VERIFY	Verify a verification pattern for the key supplied in <i>key_identifier</i> .
<b>Parity Adjustment - can not be specified with any of the AES keywords (optional)</b>	
ADJUST	Adjust the parity of test key to odd prior to generating or verifying the verification pattern. The <i>key_identifier</i> field itself is not adjusted.
NOADJUST	Do not adjust the parity of test key to odd prior to generating or verifying the verification pattern. This is the default.



## Key Test

Table 57. Keywords for Key Test Control Information (continued)

Keyword	Meaning
<i>Verification Process Rule (optional)</i>	
ENC-ZERO	ENC-ZERO can be used with any of the rules. It is not supported on systems with CCFs.
SHA-256	Use the 'SHA-256' method. Use with CLR-A128, CLR-A192, CLR-A256, and TOKEN. SHA-256 is also the default for the AES rules.

### key\_identifier

Direction: Input/Output

Type: String

The key for which to generate or verify the verification pattern. The parameter is a 64-byte string of an internal token, key label, or a clear key value left-justified.

**Note:** If you supply a key label for this parameter, it must be unique on the CKDS.

### random\_number

Direction: Input/Output

Type: String

This is an 8-byte field that contains a random number supplied as input for the test pattern verification process and returned as output with the test pattern generation process. *random\_number* is only used with the default algorithm for DES operational keys.

### verification\_pattern

Direction: Input/Output

Type: String

This is an 8-byte field that contains a verification pattern supplied as input for the test pattern verification process and returned as output with the test pattern generation process.

## Restrictions

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You can generate the verification pattern for a key when you generate the key. You can distribute the pattern with the key and it can be verified at the receiving node. In this way, users can ensure using the same key at the sending and receiving locations. You can generate and verify keys of any combination of key forms, that is, clear, operational or external.

The parity of the key is not tested.

With a PCIxCC or CCA Crypto Express coprocessor, there is support for the generation and verification of single, double and triple-length keys for the ENC-ZERO verification process. For triple-length keys, use KEY-ENC or KEY-ENCD with ENC-ZERO. Clear triple-length keys are not supported.

In the Transaction Security System, KEY-ENC and KEY-ENCD both support enciphered single-length and double-length keys. They use the key-form bits in byte 5 of CV to determine the length of the key. To be consistent, in ICSF, both KEY-ENC and KEY-ENCD handle single- and double-length keys. Both products effectively ignore the keywords, which are supplied only for compatibility reasons.

## Access Control Point

The access control point in the ICSF role that controls the function of this service is **Key Test** and **Key Test 2**. This access control point cannot be disabled. It is required for ICSF master key validation.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 58. Key test required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	Triple-length DATA keys are not supported. AES keys are not supported.
	PCI Cryptographic Coprocessor	Triple-length DATA keys are not supported. ICSF routes the request to a PCI Cryptographic Coprocessor if: <ul style="list-style-type: none"> <li>• ANSI enablement keys are not installed in the CKDS.</li> <li>• Verification process rule ENC-ZERO is specified.</li> </ul> AES keys are not supported.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.
IBM @server zSeries 890	Crypto Express2 Coprocessor	AES keys are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.  Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor  Crypto Express3 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.  Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).

## Key Test

Table 58. Key test required hardware (continued)

Server	Required cryptographic hardware	Restrictions
z196 z114	Crypto Express3 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.

## Key Test2 (CSNBKYT2 and CSNEKYT2)

Use this callable service to generate or verify a secure, cryptographic verification pattern for keys. The key to test can be in the clear, encrypted under the master key, or encrypted under a key-encrypting key. Keywords in the *rule\_array* specify whether the callable service generates or verifies a verification pattern.

The callable service name for AMODE(64) invocation is CSNEKYT2.

### Format

```
CALL CSNBKYT2(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_identifier_length,  
    key_identifier,  
    key_encrypting_key_identifier_length,  
    key_encrypting_key_identifier,  
    reserved_length,  
    reserved,  
    verification_pattern_length,  
    verification_pattern)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

**exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value must be 2, 3, 4, or 5.

**rule\_array**

Direction: Input

Type: String

The *rule\_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 59. Keywords for Key Test2 Control Information

Keyword	Meaning
<i>Token algorithm (Required)</i>	
AES	Specifies the key token is an AES key token.
DES	Specifies the key token is a DES token. CCA internal, CCA external, and TR-31 token types are supported. Clear keys are not supported for this rule.
HMAC	Specifies the key token is an HMAC key token.
<i>Process rule (One required)</i>	
GENERATE	Generate a verification pattern for the specified key.
VERIFY	Verify that a verification pattern matches the specified key.
<i>Verification pattern calculation algorithm (Optional)</i>	
ENC-ZERO	Verification pattern for AES and DES keys calculated by encrypting a data block filled with 0x00 bytes.  This is the default and only method available for DES. This method is only available for AES if Access Control Point "Key Test2 - AES, ENC-ZERO" is enabled.

## Key Test2

Table 59. Keywords for Key Test2 Control Information (continued)

Keyword	Meaning
SHA-256	Verification pattern will be calculated for an AES token using the same method as the Key Test service with the SHA-256 rule.  This rule can be used to verify that the same key value is present in a version 4 DATA token and version 5 AES CIPHER token or to verify that the same key value is present in a version 5 AES IMPORTER/EXPORTER pair.
SHA2VP1	Specifies to use the SHA-256 based verification pattern calculation algorithm. For more information, see “SHAVP1 Algorithm” on page 956. This is the default and only method available for HMAC.
<i>Token type rule (Required if TR-31 token passed and Token algorithm DES is specified; not valid otherwise)</i>	
TR-31	Specifies that <i>key_identifier</i> contains a TR-31 key block.
<i>KEK identifier rules (Optional - see defaults)</i>	
IKEK-AES	The wrapping KEK for the key to test is an AES KEK. This is the default for AES and HMAC Token algorithms, and is not allowed with DES.
IKEK-DES	The wrapping KEK for the key to test is a DES KEK. This is the default for DES Token algorithm, and is only allowed with DES Token algorithm.
IKEK-PKA	The wrapping KEK for the key to test is an RSA or (other key stored in PKA key storage.) This is not the default for any Token algorithm and must be specified if an RSA KEK is used. This rule is not allowed with DES Token algorithm.

### key\_identifier\_length

Direction: Input

Type: Integer

The length of the *key\_identifier* in bytes. The maximum value is 9992.

### key\_identifier

Direction: Input

Type: String

The key for which to generate or verify the verification pattern. This is an internal or external token or the 64-byte label of a key in the CKDS. This token may be a DES internal or external token, AES internal version '04'X token, internal or external variable-length symmetric token, or a TR-31 key block.

Clear DES tokens are not supported.

If an internal token was supplied and was encrypted under the old master key, the token will be returned encrypted under the current master key.

### key\_encrypting\_key\_identifier\_length

Direction: Input

Type: Integer

The length of the *key\_encrypting\_key\_identifier* parameter. When *key\_identifier* is an internal token, the value must be zero.

If *key\_encrypting\_key\_identifier* is a label for either the CKDS (IKEK-AES or IKEK-DES rules) or PKDS (IKEK-PKA rule), the value must be 64. If *key\_encrypting\_key\_identifier* is an AES KEK, the value must be between the actual length of the token and 725. If *key\_encrypting\_key\_identifier* is a DES KEK, the value must be 64. If *key\_encrypting\_key\_identifier* is an RSA KEK, the maximum length is 3500.

#### **key\_encrypting\_key\_identifier**

Direction: Input/Output

Type: String

When *key\_encrypting\_key\_identifier\_length* is non-zero, *key\_encrypting\_key\_identifier* contains an internal key token containing the key-encrypting key, or a key label.

If the key identifier supplied was an AES or DES token encrypted under the old master key, the token will be returned encrypted under the current master key.

#### **reserved\_length**

Direction: Input

Type: Integer

The length of the reserved parameter. The value must be zero.

#### **reserved**

Direction: Input/Output

Type: String

This parameter is ignored.

#### **verification\_pattern\_length**

Direction: Input/Output

Type: Integer

The length of the *verification\_pattern* parameter.

On input: For GENERATE, the length must be at least 8 bytes; For VERIFY, the length must be 8 bytes.

On output for GENERATE, the length of the verification pattern returned.

#### **verification\_pattern**

Direction: Input/Output

Type: String

For GENERATE, the verification pattern generated for the key.

For VERIFY, the supplied verification pattern to be verified.

## **Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

You can generate the verification pattern for a key when you generate the key. You can distribute the pattern with the key and it can be verified at the receiving node. In this way, users can ensure using the same key at the sending and receiving locations. You can generate and verify keys of any combination of key forms: clear, operational or external.

## Access Control Point

The access control point in the ICSF role that controls the function of this service is **Key Test and Key Test 2**. This access control point cannot be disabled. It is required for ICSF master key validation.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 60. Key Test2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This service is not supported.
IBM @server zSeries 990		This service is not supported.
IBM @server zSeries 890		
IBM System z9 EC		This service is not supported.
IBM System z9 BC		
IBM System z10 EC	Crypto Express2 Coprocessor	This service is not supported.
IBM System z10 BC	Crypto Express3 Coprocessor	This service is not supported.
z196 z114	Crypto Express3 Coprocessor	DES/AES key support requires the Sep. 2011 or later licensed internal code (LIC).  HMAC key support requires the Nov. 2010 or later licensed internal code (LIC).
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## Key Test Extended (CSNBKYTX and CSNEKTX)

Use the key test extended service to generate or verify a secure, cryptographic verification pattern for keys. The key to test can be in the clear or encrypted under the master key. The callable service also supports keys encrypted under a key-encrypting key (KEK). AES keys are not supported by this service. Keywords in the rule array specify whether the callable service generates or verifies a verification pattern.

This algorithm is supported for encrypted single and double length keys. Single, double and triple length keys are also supported with the ENC-ZERO algorithm.

When the service generates a verification pattern, it creates and cryptographically processes a random number. The service returns the random number with the verification pattern.

When the service tests a verification pattern against a key, you must supply the random number and the verification pattern from a previous call to key test extended. The service returns the verification result in the return and reason codes.

The callable service name for AMODE(64) invocation is CSNEKTX.

## Format

```
CALL CSNBKYTX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier,
    random_number,
    verification_pattern,
    KEK_key_identifier)
```

## Parameters

### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### reason\_code

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### exit\_data\_length

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'FFFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### exit\_data

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### rule\_array\_count

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value can be 2, 3 or 4.



## Key Test Extended

### rule\_array

Direction: Input

Type: String

Two or three keywords that provide control information to the callable service. Table 61 lists the keywords. The keywords must be in 16 or 24 bytes of contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 61. Keywords for Key Test Extended Control Information

Keyword	Meaning
<b>Key Rule (required)</b>	
KEY-ENC	Specifies the key supplied in <i>key_identifier</i> is a single-length encrypted DES key.
KEY-ENCD	Specifies the key supplied in <i>key_identifier</i> is a double-length encrypted DES key.
<b>Process Rule (required)</b>	
GENERATE	Generate a verification pattern for the key supplied in <i>key_identifier</i> .
VERIFY	Verify a verification pattern for the key supplied in <i>key_identifier</i> .
<b>Parity Adjustment (optional)</b>	
ADJUST	Adjust the parity of test key to odd prior to generating or verifying the verification pattern. The <i>key_identifier</i> field itself is not adjusted.
NOADJUST	Do not adjust the parity of test key to odd prior to generating or verifying the verification pattern. This is the default.
<b>Verification Process Rule (optional)</b>	
ENC-ZERO	Specifies use of the "encrypted zeros" method.

### key\_identifier

Direction: Input/Output

Type: String

The key for which to generate or verify the verification pattern. The parameter is a 64-byte string of an internal token or key label that is left-justified.

**Note:** If you supply a key label for this parameter, it must be unique on the CKDS.

### random\_number

Direction: Input/Output

Type: String

This is an 8-byte field that contains a random number supplied as input for the test pattern verification process and returned as output with the test pattern generation process.

### verification\_pattern

Direction: Input/Output

Type: String

This is an 8-byte field that contains a verification pattern supplied as input for the test pattern verification process and returned as output with the test pattern generation process.

#### KEK\_key\_identifier

Direction: Input/Output

Type: String

If *key\_identifier* is an external token, then this is a 64-byte string of an internal token or a key label of an IMPORTER or EXPORTER used to encrypt the test key. If *key\_identifier* is an internal token, then the parameter is ignored.

**Note:** If you supply a key label for this parameter, it must be unique on the CKDS.

## Restrictions

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You can generate the verification pattern for a key when you generate the key. You can distribute the pattern with the key and it can be verified at the receiving node. In this way, users can ensure using the same key at the sending and receiving locations. You can generate and verify keys of any combination of key forms, that is, clear, operational or external.

The parity of the key is not tested.

With a PCIXCC or CCA Crypto Express coprocessor and using the ENC-ZERO verification rule, there is support for enciphered single and double-length DES keys. There is no support for systems with CCF's.

## Access Control Point

The access control point in the ICSF role that controls the function of this service is **Key Test and Key Test 2**. This access control point cannot be disabled. It is required for ICSF master key validation.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

## Key Test Extended

Table 62. Key test extended required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	Triple-length DATA keys are not supported.  The key test extended callable service is processed on the Cryptographic Coprocessor Feature. <i>Rule_array</i> keyword ENC-ZERO is not valid.  AES keys are not supported.
	PCI Cryptographic Coprocessor	Triple-length DATA keys are not supported.  ICSF routes the request to a PCI Cryptographic Coprocessor if: <ul style="list-style-type: none"> <li>• ANSI enablement keys are not installed in the CKDS.</li> <li>• Verification process rule ENC-ZERO is specified.</li> </ul> AES keys are not supported.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.
IBM @server zSeries 890	Crypto Express2 Coprocessor	AES keys are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.  AES keys are not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor  Crypto Express3 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.  AES keys are not supported.
z196  z114	Crypto Express3 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	Clear triple-length keys are not supported. Encrypted triple-length keys are supported with the ENC-ZERO keyword only.

## Key Token Build (CSNBKTB and CSNEKTB)

Use the key token build callable service to build an external or internal key token from information which you supply. The token can be used as input for the key generate and key part import callable services. You can specify a control vector or the service can build a control vector based upon the key type you specify and the control vector-related keywords in the rule array. ICSF supports the building of an internal key token with the key encrypted under a master key other than the current master key and building internal clear AES and DES tokens.

The callable service name for AMODE(64) invocation is CSNEKTb.

## Format

```
CALL CSNBKTB(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_token,
    key_type,
    rule_array_count,
    rule_array,
    key_value,
    master_key_version_number,
    key_register_number,
    token_data_1,
    control_vector,
    initialization_vector,
    pad_character,
    cryptographic_period_start,
    master_key_verification_pattern)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **key\_token**

Direction: Input/Output

Type: String

## Key Token Build

If the *key\_type* parameter is TOKEN, then this is a 64-byte internal token that is updated as specified in the *rule\_array*. The internal token must be a DATA, IMPORTER or EXPORTER key type. Otherwise this field is an output-only field.

### key\_type

Direction: Input

Type: String

An 8-byte field that specifies the type of key you want to build or the keyword TOKEN for updating a supplied token. The key types are:

Table 63. Key type keywords for key token build

Key type	Description	Algorithm
AKEK	See Table 3 on page 23.	DES
CIPHER CIPHERXI CIPHERXL CIPHERXO	See Table 3 on page 23.	DES
CLRAES	The <i>key_token</i> parameter is a clear AES token. The <i>rule_array</i> must contain the keyword INTERNAL and one of the optional keywords: KEYLN16, KEYLN24 or KEYLN32. A <i>key value</i> parameter must also be provided.	AES
CLRDES	The <i>key_token</i> parameter is a clear DES token. The <i>rule_array</i> must contain the keyword INTERNAL and one of the optional keywords: KEYLN8, KEYLN16 or KEYLN24. A <i>key value</i> parameter must also be provided.	DES
CVARDEC CVARENC CVARPINE CVARXCVL CVARXCVR	See Table 3 on page 23.	DES
DATA	Valid for AES and DES keys and must be specified with the <i>rule_array</i> keyword AES to build an encrypted AES key token.	AES and DES
DATAAC DATAM DATAMV DATAXLAT DECIPHER DKYGENKY ENCIPHER	See Table 3 on page 23.	DES
EXPORTER	If the <i>key_type</i> parameter is TOKEN, then this is a 64-byte internal token that is updated as specified in the <i>rule_array</i> .	DES
IKEYXLAT	See Table 3 on page 23.	DES
IMPORTER	If the <i>key_type</i> parameter is TOKEN, then this is a 64-byte internal token that is updated as specified in the <i>rule_array</i> .	DES
KEYGENKY	CLR8-ENC or UKPT must be coded in <i>rule_array</i> parameter	DES

Table 63. Key type keywords for key token build (continued)

Key type	Description	Algorithm
IPINENC MAC MACVER OKEYXLAT OPINENC PINGEN PINVER	See Table 3 on page 23.	DES
SECMSG	SMKEY or SMPIN must be specified in the <i>rule_array</i> parameter.	DES

If *key\_type* is TOKEN, then the *key\_token* field must contain a single-length DATA key or an IMPORTER or EXPORTER key with the standard control vector. The valid keywords for TOKEN are EXTERNAL, INTERNAL, DES and SYS-ENC. The service will set the system encryption bits in the token (byte 59, bits 0 and 1) to zero and return the token.

Key type USE-CV is used when a user-supplied control vector is specified. The USE-CV key type specifies that the key type should be obtained from the control vector specified in the *control\_vector* parameter. The CV rule array keyword should be specified if USE-CV is specified.

For information on the meaning of the key types, see Table 3 on page 23.

#### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter.

#### **rule\_array**

Direction: Input

Type: String

The *rule\_array* contains keywords that provide control information to the callable service. See Table 64 for a list. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. For any key type, there are no more than four valid *rule\_array* values.

Table 64. Keywords for Key Token Build Control Information

Keyword	Meaning	Algorithm
<i>Token Algorithm (optional - zero or one keyword)</i>		
AES	Specifies that an AES key token will be built. This keyword is required when building an encrypted AES token. It is optional when using the CLRAES key type to build a clear AES token.	AES
DES	Specifies a DES token will be built.	DES
SYS-ENC	Tolerated for compatibility reasons.	DES
<i>Token Type (one keyword required)</i>		
EXTERNAL	Specifies that an external key token will be built.	DES

Table 64. Keywords for Key Token Build Control Information (continued)

Keyword	Meaning	Algorithm
INTERNAL	Specifies that an internal key token will be built.	AES or DES
<b>Key Status (optional - zero or one keyword)</b>		
KEY	This keyword indicates that the key token to build will contain an encrypted key. The <i>key_value</i> parameter identifies the field that contains the key.	AES or DES
NO-KEY	This keyword indicates that the key token to build will not contain a key. This is the default key status.	AES or DES
<b>Key Length (one keyword required for AES keys)</b>		
KEYLN8	Single-length or 8-byte key. Default for CLRDES.	DES
KEYLN16	Specifies that the key is 16-bytes long.	AES or DES
KEYLN24	Specifies that the key is 24-bytes long.	AES or DES
KEYLN32	Specifies that the key is 32-bytes long.	AES
DOUBLE	Double-length or 16-byte key. Synonymous with KEYLN16. Not valid for CLRDES. <b>Note:</b> See Table 66 on page 200 for valid key types for these key length values.	DES
DOUBLE-O	Double-length key with guaranteed unique key values. The key is 16 bytes long. This key length can be used with any key type that supports DOUBLE.	DES
MIXED	Double-length key. Indicates that the key can either be a replicated single-length key or a double-length key with two different 8-byte values. Not valid for CLRDES.	DES
SINGLE	Single-length or 8-byte key. Synonymous with KEYLN8. Not valid for CLRDES.	DES
<b>Key Part Indicator (optional) — not valid for CLRDES</b>		
KEY-PART	This token is to be used as input to the key part import service.	DES
<b>Control vector (CV) source (optional - zero or one of these keywords is permitted)</b>		
CV	This specifies that the key token should be built using the control_vector supplied in the <i>control_vector</i> parameter.	DES
NO-CV	This specifies that the key token should be built using a control vector that is based on the supplied key type control vector related rule array keywords. It is the default.	DES
<b>Control vector on the link specification (optional) — valid only for IMPORTER and EXPORTER.</b>		

Table 64. Keywords for Key Token Build Control Information (continued)

Keyword	Meaning	Algorithm
CV-KEK	This keyword indicates marking the KEK as a CV KEK. The control vector is applied to the KEK prior to using it in encrypting other keys. This is the default.	DES
NOCV-KEK	This keyword indicates marking the KEK as a NOCV KEK. The control vector is not applied to the KEK prior to its use in encrypting other keys.	DES
<b>Control vector keywords (optional - zero or more of these keywords are permitted)</b>		
See Table 66 on page 200 for the key-usage keywords that can be specified for a given key type.		DES
<b>Master Key Verification Pattern (optional) — not valid for CLRDES or CLRAES keywords</b>		
MKVP	This keyword indicates that the <i>key_value</i> is enciphered under the master key which corresponds to the master key verification pattern specified in the <i>master_key_verification_pattern</i> parameter. If this keyword is not specified, the key contained in the <i>key_value</i> field must be enciphered under the current master key.	AES and DES
<b>Key Wrapping Method (optional)</b>		
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.	DES
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens. This is the default.	DES
<b>Translation Control (optional)</b>		
ENH-ONLY	Restrict rewrapping of the token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method. Can only be specified with WRAP-ENH.	DES

**key\_value**

Direction: Input

Type: String

If you use the KEY keyword, this parameter is a 16-byte string that contains the encrypted key value. Single-length keys must be left-justified in the field and padded on the right with X'00'. If you are building a triple-length DATA key, this parameter is a 24-byte string containing the encrypted key value. If you supply an encrypted key value and also specify INTERNAL, the service will check for the presence of the MKVP keyword. If MKVP is present, the service will assume the *key\_value* is enciphered under the master key which corresponds to the master key verification pattern specified in the *master\_key\_verification\_pattern* parameter, and will place the key into the internal token along with the verification pattern from the *master\_key\_verification\_pattern* parameter. If MKVP is not specified, ICSF



## Key Token Build

assumes the key is enciphered under the current host master key and places the key into an internal token along with the verification pattern for the current master key. In this case, the application must ensure that the master key has not changed since the key was generated or imported to this system. Otherwise, use of this parameter is not recommended.

For *key\_type* CLRDES and CLRAES, this field is required to contain the clear key value. For KEYLN8, this is an 8-byte field. For KEYLN16, this is a 16-byte field. For KEYLN24, this a 24-byte field. For KEYLN32, this is a 32-byte field.

Table 65. Key types and field lengths for AES keys

Key type	Field length
AES-128 clear text key	16-bytes
AES-192 clear text key	24-bytes
AES-256 clear text key	32-bytes
AES-128, AES-192, AES-256 encrypted key	32-bytes

### **master\_key\_version\_number**

Direction: Input

Type: Integer

This field is examined only if the KEY keyword is specified, in which case, this field must be zero.

### **key\_register\_number**

Direction: Input

Type: Integer

This field is ignored.

### **token\_data\_1**

Direction: Input

Type: String

This parameter is ignored for DES keys.

This parameter is the LRC value for AES keys. For clear AES keys it is 8-bytes of X'00' indicating to the service that it must compute the LRC field value. For encrypted AES keys, you provide a 1-byte area containing the LRC value for the key passed in the *key\_value* parameter. The service copies it into the LRC field of the key token.

### **control\_vector**

Direction: Input

Type: String

A pointer to a 16 byte string variable. When the CV rule array keyword is used, this parameter must point to a control vector which is copied into the key token. This parameter is ignored for AES keys.

### **initialization\_vector**

Direction: Input

Type: String

This field is ignored.

### **pad\_character**

Direction: Input

Type: Integer

The only allowed value for key types MAC and MACVER is 0. This field is ignored for all other key types.

**cryptographic\_period\_start**

Direction: Input

Type: String

This field is ignored.

**master\_key\_verification\_pattern**

Direction: Input

Type: String

8-byte verification pattern of the master key used to encrypt the key value. It is used when the KEY and INTERNAL *rule\_array* keywords are specified. The value is inserted into the master key verification pattern field of the key token. If the KEY and INTERNAL keywords are specified in *rule\_array*, the service will check for the existence of the MKVP rule array keyword. This parameter is ignored for any other *rule\_array* keyword combinations.

## Restrictions

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes

No pre- or post-processing or security exits are enabled for this service. No RACF checking is done, and no calls to RACF are issued when this service is used.

You can use this service to create skeleton key tokens with the desired data encryption algorithm bits for use in some key management services to override the default system specifications.

- If you are running with the Cryptographic Coprocessor Feature and need to generate operational AKEKs, use *key\_type* of TOKEN and provide a skeleton AKEK key token as the *generated\_key\_identifier\_1* into the key generate service.
- If you are running with the Cryptographic Coprocessor Feature, the KEY-PART AKEK key token can also be used as input to key part import service.
- To create an internal token with a specified KEY value, ICSF needs to supply a valid master key verification pattern (MKVP).

NOCV keyword is only supported for the standard IMPORTERs and EXPORTERs with the default CVs.

This illustrates the key type and key usage keywords that can be combined in the Control Vector Generate and Key Token Build callable services to create a control vector.

## Key Token Build

Table 66. Control Vector Generate and Key Token Build Control Vector Keyword Combinations

Key Type		Key Usage				
DATA			SINGLE KEYLN8 MIXED DOUBLE DOUBLE-0 KEYLN16 KEYLN24	XPORT-OK NO-XPORT	KEY-PART	
CIPHER ENCIPHER DECIPHER			SINGLE KEYLN8 MIXED DOUBLE DOUBLE-0 KEYLN16	XPORT-OK NO-XPORT	KEY-PART ENH-ONLY	T31XPTOK NOT31XPT
CIPHERXI CIPHERXL CIPHERXO			DOUBLE DOUBLE-0 KEYLN16	XPORT-OK NO-XPORT	KEY-PART ENH-ONLY	T31XPTOK NOT31XPT
MAC MACVER	ANY-MAC ANSIX9.9 CVVKEY-A CVVKEY-B AMEX-CSC		SINGLE KEYLN8 MIXED DOUBLE DOUBLE-0 KEYLN16	XPORT-OK NO-XPORT	KEY-PART ENH-ONLY	T31XPTOK NOT31XPT
DATAXLAT CVARPINE CVARENC CVARDEC CVARXCVL CVARXCVR			SINGLE KEYLN8	XPORT-OK NO-XPORT	KEY-PART ENH-ONLY	T31XPTOK NOT31XPT
DATAC DATAM DATAMV			DOUBLE DOUBLE-0 KEYLN16 MIXED	XPORT-OK NO-XPORT	KEY-PART ENH-ONLY	T31XPTOK NOT31XPT
KEYGENKY	CLR8-ENC UKPT		DOUBLE DOUBLE-0 KEYLN16 MIXED	XPORT-OK NO-XPORT	KEY-PART ENH-ONLY	T31XPTOK NOT31XPT
DKYGENKY	DDATA DMAC DMV DIMP DEXP DPVR DMKEY DMPIN DALL	DKYL0 DKYL1 DKYL2 DKYL3 DKYL4 DKYL5 DKYL6 DKYL7	DOUBLE DOUBLE-0 KEYLN16 MIXED	XPORT-OK NO-XPORT	KEY-PART ENH-ONLY	T31XPTOK NOT31XPT
SECMSG		SMKEY SMPIN	DOUBLE DOUBLE-0 KEYLN16 MIXED	XPORT-OK NO-XPORT	KEY-PART ENH-ONLY	T31XPTOK NOT31XPT
IKEYXLAT OKEYXLAT		ANY NOT-KEK DATA PIN LMTD-KEK	DOUBLE DOUBLE-0 KEYLN16 MIXED	XPORT-OK NO-XPORT	KEY-PART ENH-ONLY	T31XPTOK NOT31XPT

Table 66. Control Vector Generate and Key Token Build Control Vector Keyword Combinations (continued)

Key Type	Key Usage						
IMPORTER	OPIM* IMEX* IMIM* IMPORT*	XLATE	<b>ANY</b> NOT-KEK DATA PIN LMTD-KEK	<b>DOUBLE</b> DOUBLE-0 KEYLN16 MIXED	<b>XPORT-OK</b> NO-XPORT	KEY-PART ENH-ONLY	<b>T31XPTOK</b> NOT31XPT
EXPORTER	OPEX* IMEX* EXEX* EXPORT*	XLATE	<b>ANY</b> NOT-KEK DATA PIN LMTD-KEK	<b>DOUBLE</b> DOUBLE-0 KEYLN16 MIXED	<b>XPORT-OK</b> NO-XPORT	KEY-PART ENH-ONLY	<b>T31XPTOK</b> NOT31XPT
PINVER		<b>NO-SPEC**</b> IBM-PIN** GBP-PIN** IBM-PINO GBP-PINO VISA-PVV INBK-PIN	NOOFFSET	<b>DOUBLE</b> DOUBLE-0 KEYLN16 MIXED	<b>XPORT-OK</b> NO-XPORT	KEY-PART ENH-ONLY	<b>T31XPTOK</b> NOT31XPT
PINGEN	CPINGEN* CPINGENA* EPINGENA* EPINGEN* EPINVER*	<b>NO-SPEC**</b> IBM-PIN** GBP-PIN** IBM-PINO GBP-PINO VISA-PVV INBK-PIN	NOOFFSET	<b>DOUBLE</b> DOUBLE-0 KEYLN16 MIXED	<b>XPORT-OK</b> NO-XPORT	KEY-PART ENH-ONLY	<b>T31XPTOK</b> NOT31XPT
IPINENC	CPINGENA* EPINVER* REFORMAT* TRANSLAT*			<b>DOUBLE</b> DOUBLE-0 KEYLN16 MIXED	<b>XPORT-OK</b> NO-XPORT	KEY-PART ENH-ONLY	<b>T31XPTOK</b> NOT31XPT
OPINENC	CPINENC* EPINGEN* REFORMAT* TRANSLAT*			<b>DOUBLE</b> DOUBLE-0 KEYLN16 MIXED	<b>XPORT-OK</b> NO-XPORT	KEY-PART ENH-ONLY	<b>T31XPTOK</b> NOT31XPT
Notes:	<p>Default keys are indicated in bold.</p> <p>* All keywords in the list are defaults unless one or more keywords in the list are specified</p> <p>** The NOOFFSET keyword is only valid if NO-SPEC, IBM-PIN, GBP-PIN, or the default (NO-SPEC) is specified.</p> <p>A key usage keyword is required for the KEYGENKY and SECMSG key types.</p> <ul style="list-style-type: none"> <li>• CLR8-ENC and/or UKPT must be specified for the KEYGENKY key type</li> <li>• SMKEY or SMPIN must be specified for the SECMSG key type</li> </ul>						

**Attention:** CDMF is no longer supported.

The ICSF key token build callable service provides a subset of the parameters and keywords available with the Transaction Security System key token build verb.

These key types are not supported: ADATA, AMAC, UKPTBASE.

These rule array keywords are not supported: ACTIVE, ADAPTER, CARD, CBC, CLEAR-IV, CUSP, INACTIVE, IPS, KEY-REF, MACLEN4, MACLEN6, MACLEN8, NO-IV, READER, X9.2, X9.9-1.

### Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 67. Key token build required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None.	
z114		
IBM zEnterprise EC12	None.	

---

### Key Token Build2 (CSNBKTB2 and CSNEKTB2)

Use the Key Token Build2 callable service to build a variable-length CCA symmetric key token in application storage from information that you supply. A clear key token built by this service can be used as input for the Key Test2 callable service. A skeleton token built by this service can be used as input for the Key Generate2 and Key Part Import2 callable services.

This service will build internal or external HMAC and AES tokens, both as clear key tokens and as skeleton tokens containing no key.

The callable service name for AMODE(64) is CSNEKTB2.

## Format

```
CALL CSNBKTB2(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    clear_key_bit_length,
    clear_key_value,
    key_name_length,
    key_name,
    user_associated_data_length,
    user_associated_data,
    token_data_length,
    token_data,
    reserved_length,
    reserved,
    target_key_token_length,
    target_key_token )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The minimum value is 3, and the maximum value is 33.

### **rule\_array**

## Key Token Build2

Direction: Input

Type: Integer

The *rule\_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 68. Keywords for Key Token Build2 Control Information

Keyword	Meaning
<i>Token type (one required)</i>	
EXTERNAL	Specifies to build an external key token.
INTERNAL	Specifies to build an internal key token.
<i>Token algorithm (one required)</i>	
AES	Specifies to build an AES key token.
HMAC	Specifies to build an HMAC key token.
<i>Key status (one, optional)</i>	
KEY-CLR	Specifies to build the key token with a clear key value. This creates a key token that can be used with the Key Test2 service to generate a verification pattern for the key value.
NO-KEY	Specifies to build the key token without a key value. This creates a skeleton key token that can later be supplied to the Key Generate2 service. This is the default.
<i>Key type (one required)</i>	
CIPHER	Specifies that this key is for an AES CIPHER key. Only valid for AES algorithm.
EXPORTER	Specifies that this key is for an AES KEK EXPORTER. Only valid for AES algorithm.
IMPORTER	Specifies that this key is for an AES KEK IMPORTER. Only valid for AES algorithm.
MAC	Specifies that this key is for message authentication code operations. Only valid for HMAC algorithm.
<b>Key-management related keywords</b>	
<i>Symmetric-key export control (one, optional)</i> <i>Key-management field 1 for all algorithms and key types.</i>	
NOEX-SYM	Prohibits the export of the key with a symmetric key.
XPRT-SYM	Permits the export of the key with a symmetric key. This is the default.
<i>Unauthenticated asymmetric-key export control (one, optional)</i> <i>Key-management field 1 for all algorithms and key types.</i>	
NOEXUASY	Prohibits the export of the key with an unauthenticated asymmetric key.
XPRTUASY	Permits the export of the key with an unauthenticated asymmetric key. This is the default.
<i>Authenticated asymmetric-key export control (one, optional)</i> <i>Key-management field 1 for all algorithms and key types.</i>	
NOEXAASY	Prohibits the export of the key with an authenticated asymmetric key.
XPRTAASY	Permits the export of the key with an authenticated asymmetric key. This is the default.
<i>RAW-format export control (one, optional)</i> <i>Key-management field 1 for all algorithms and key types.</i>	

Table 68. Keywords for Key Token Build2 Control Information (continued)

Keyword	Meaning
NOEX-RAW	Prohibits the export of the key in RAW format. This is the default.
XPRT-RAW	Permits the export of the key in RAW format.
<i>DES-key export control (one, optional)</i> <i>Key-management field 1 for all algorithms, all key types.</i>	
NOEX-DES	Prohibits the export of the key using DES key.
XPRT-DES	Permits the export of the key using DES key. This is the default.
<i>AES-key export control (one, optional)</i> <i>Key-management field 1 for all algorithms, all key types.</i>	
NOEX-AES	Prohibits the export of the key using AES key.
XPRT-AES	Permits the export of the key using AES key. This is the default.
<i>RSA-key export control (one, optional)</i> <i>Key-management field 1 for all algorithms, all key types.</i>	
NOEX-RSA	Prohibits the export of the key using RSA key.
XPRT-RSA	Permits the export of the key using RSA key. This is the default.
<b>Key-usage keywords (these are specific to the key type specified)</b>	
<i>Generate control (one required)</i> <i>Key-usage field 1 for HMAC algorithm, MAC key type.</i>	
GENERATE	Specifies that this key can be used to generate a MAC. A key that can generate a MAC can also verify a MAC.
VERIFY	Specifies that this key cannot be used to generate a MAC. It can only be used to verify a MAC.
<i>Encrypt control (optional, any combination)</i> <i>Key-usage field 1 for AES algorithm, CIPHER key type.</i> <b>Note:</b> All keywords in the list below are defaults unless one or more keywords in the list are specified.	
ENCRYPT	Specifies that this key can be used to encipher data using the AES algorithm.
DECRYPT	Specifies that this key can be used to decipher data using the AES algorithm.
<i>Ciphertext Translate Control (optional)</i> <i>Key-usage field 1 for AES algorithm, CIPHER key type.</i>	
C-XLATE	Specifies that this key can only be used for cipher text translation.
<i>Exporter control (any combination, optional)</i> <i>Key-usage field 1 for AES algorithm, EXPORTER key type.</i> <b>Note:</b> All keywords in the list below are defaults unless one or more keywords in the list are specified.	
EXPORT	Specifies that this key can be used for export.
TRANSLAT	Specifies that this key can be used for translate.
GEN-OPEX	Specifies that this key can be used for generate OPEX.
GEN-IMEX	Specifies that this key can be used for generate IMEX.
GEN-EXEX	Specifies that this key can be used for generate EXEX.
GEN-PUB	Specifies that this key can be used for generate PUB.
<i>Importer control (any combination, optional)</i> <i>Key-usage field 1 for AES algorithm, IMPORTER key type.</i> <b>Note:</b> All keywords in the list below are defaults unless one or more keywords in the list are specified.	
IMPORT	Specifies that this key can be used for import.
TRANSLAT	Specifies that this key can be used for translate.
GEN-OPIM	Specifies that this key can be used for generate OPIM.



## Key Token Build2

Table 68. Keywords for Key Token Build2 Control Information (continued)

Keyword	Meaning
GEN-IMEX	Specifies that this key can be used for generate IMEX.
GEN-IMIM	Specifies that this key can be used for generate IMIM.
GEN-PUB	Specifies that this key can be used for generate PUB.
<b>User-defined extension control (any combination, optional)</b> <b>Low-order byte of key-usage field 1 for all algorithms and key types.</b> <b>Note:</b> The default is such that the key can be used in both UDXs and CCA and none of the user-defined UDX bits are set.	
UDX-ONLY	Specifies that this key can only be used in UDXs.
UDX-001	Specifies that the rightmost user-defined UDX bit is set.
UDX-010	Specifies that the middle user-defined UDX bit is set.
UDX-100	Specifies that the leftmost user-defined UDX bit is set.
<b>Hash method control (any combination, optional)</b> <b>Key-usage field 2 for HMAC algorithm, MAC key type.</b> <b>Note:</b> All keywords in the list below are defaults unless one or more keywords in the list are specified.	
SHA-1	Specifies that the SHA-1 hash method is allowed for the key.
SHA-224	Specifies that the SHA-224 hash method is allowed for the key.
SHA-256	Specifies that the SHA-256 hash method is allowed for the key.
SHA-384	Specifies that the SHA-384 hash method is allowed for the key.
SHA-512	Specifies that the SHA-512 hash method is allowed for the key.
<b>Mode control (one, optional)</b> <b>Key-usage field 2 for AES algorithm, CIPHER key type.</b>	
CBC	Specifies that this key can be used for cipher block chaining. This is the default.
CFB	Specifies that this key can be used for cipher feedback.
ECB	Specifies that this key can be used for electronic code book.
GCM	Specifies that this key can be used for Galois/counter mode.
OFB	Specifies that this key can be used for output feedback.
XTS	Specifies that this key can be used for Xor-Encrypt-Xor-based Tweak Stealing.
<b>Key-encrypting key control (any combination, optional)</b> <b>Key-usage field 2 for AES algorithm, EXPORTER or IMPORTER key type.</b> <b>Note:</b> The default is such that the key cannot export a RAW key nor wrap or unwrap a TR-31 key block.	
KEK-RAW	Specifies that this key-encrypting key can export a RAW key. A RAW key is a key that is encrypted but does not have any associated data.
WR-TR31	Specifies that this key-encrypting key can wrap or unwrap a TR-31 key block
<b>Key-usage wrap algorithm control (any combination, optional)</b> <b>Key-usage field 3 for AES algorithm, EXPORTER or IMPORTER key type.</b> <b>Note:</b> Keywords WR-DES, WR-AES, and WR-HMAC are defaults unless one or more keywords are specified.	
WR-DES	Specifies that this key can be used to wrap DES keys.
WR-AES	Specifies that this key can be used to wrap AES keys.
WR-HMAC	Specifies that this key can be used to wrap HMAC keys.
WR-RSA	Specifies that this key can be used to wrap RSA keys.
WR-ECC	Specifies that this key can be used to wrap ECC keys.

Table 68. Keywords for Key Token Build2 Control Information (continued)

Keyword	Meaning
<i>Key-usage wrap class control (any combination, optional)</i> <i>Key-usage field 4 for AES algorithm, EXPORTER or IMPORTER key type.</i> <b>Note:</b> All keywords in the list below are defaults unless one or more keywords in the list are specified.	
WR-DATA	Specifies that this key can be used to wrap DATA class keys.
WR-KEK	Specifies that this key can be used to wrap KEK class keys.
WR-PIN	Specifies that this key can be used to wrap PIN class keys.
WRDERIVE	Specifies that this key can be used to wrap DERIVATION class keys.
WR-CARD	Specifies that this key can be used to wrap CARD class keys.

**clear\_key\_bit\_length**

Direction: Input

Type: Integer

The length of the clear key in bits. Specify 0 when no key value is supplied (Key status rule NO-KEY). Specify a valid key bit length when a key value is supplied (Key status rule KEY-CLR):

- For HMAC algorithm, MAC key type, this is a value between 80 and 2048.
- For AES algorithm, CIPHER/EXPORTER/IMPORTER key types, this is a value of 128, 192, or 256.

**clear\_key\_value**

Direction: Input

Type: String

This parameter is used when the KEY-CLR keyword is specified. This parameter is the clear key value to be put into the token being built.

**key\_name\_length**

Direction: Input

Type: Integer

The length of the *key\_name* parameter. Valid values are 0 and 64.

**key\_name**

Direction: Input

Type: String

A 64-byte key store label to be stored in the associated data structure of the token.

**user\_associated\_data\_length**

Direction: Input

Type: Integer

The length of the user-associated data. The valid values are 0 to 255 bytes.

**user\_associated\_data**

Direction: Input

Type: String

User-associated data to be stored in the associated data structure.

**token\_data\_length**

## Key Token Build2

Direction: Input

Type: Integer

This parameter is reserved. The value must be zero.

### **token\_data**

Direction: Ignored

Type: String

This parameter is ignored.

### **reserved\_length**

Direction: Input

Type: Integer

This parameter is reserved. The value must be zero.

### **reserved**

Direction: Ignored

Type: String

This parameter is ignored because *reserved\_length* must be zero.

### **target\_key\_token\_length**

Direction: Input/Output

Type: Integer

On input, the length of the *target\_key\_token* parameter supplied to receive the token. On output, the actual length of the token returned to the caller. Maximum length is 725 bytes.

### **target\_key\_token**

Direction: Output

Type: String

The key token built by this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 69. Key Token Build2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None	
IBM @server zSeries 990	None	
IBM @server zSeries 890		
IBM System z9 EC	None	
IBM System z9 BC		
IBM System z10 EC	None	
IBM System z10 BC		

Table 69. Key Token Build2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
z196	None	
z114		
IBM zEnterprise EC12	None	

## Key Translate (CSNBKTR and CSNEKTR)

The Key Translate callable service uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment.

**Note:** All key labels must be unique.

The callable service name for AMODE(64) invocation is CSNEKTR.

## Format

```
CALL CSNBKTR(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    input_key_token,
    input_KEK_key_identifier,
    output_KEK_key_identifier,
    output_key_token )
```

## Parameters

### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### reason\_code

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### exit\_data\_length

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

## Key Translate

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **input\_key\_token**

Direction: Input

Type: String

A 64-byte string variable containing an external key token. The external key token contains the key to be re-enciphered (translated).

### **input\_KEK\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string variable containing the internal key token or the key label of an internal key token record in the CKDS. The internal key token contains the key-encrypting key used to decipher the key. The internal key token must contain a control vector that specifies an importer or IKEYXLAT key type. The control vector for an importer key must have the XLATE bit set to 1.

### **output\_KEK\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string variable containing the internal key token or the key label of an internal key token record in the CKDS. The internal key token contains the key-encrypting key used to encipher the key. The internal key token must contain a control vector that specifies an exporter or OKEYXLAT key type. The control vector for an exporter key must have the XLATE bit set to 1.

### **output\_key\_token**

Direction: Output

Type: String

A 64-byte string variable containing an external key token. The external key token contains the re-enciphered key.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The *output\_key\_token* will be wrapped in the same manner as the *input\_key\_token*.

## Restrictions

Triple length DATA key tokens are not supported.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

## Access Control Points

The **Key Translate** access control point controls the function of this service.

If the output key-encrypting key identifier is a weaker key than the key being translated, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the output key-encrypting key identifier key is a 16-byte key.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 70. Key translate required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	

## Key Translate2 (CSNBKTR2 and CSNEKTR2)

The Key Translate2 callable service translates the *input\_key\_token* parameter in one of several ways:

- Changes an external DES or variable-length symmetric key token from encipherment under one key-encrypting key to another
- Changes the wrapping method of an external DES key token
- Converts an operational AES DATA token (version X'04') to an operational AES CIPHER token (version X'05') or converts an operational AES CIPHER token (version X'05') to an operational AES DATA token (version X'04')

## Key Translate2

To reencipher a key token, specify the TRANSLAT rule array keyword (the default), the external key token, and the input and output key-encrypting keys. If the *input\_key\_token* is a DES key token, you can also specify which key wrapping method to use. If no wrapping method is specified, the system default wrapping method will be used.

To change the wrapping method of an external DES key token, specify the REFORMAT rule array keyword, the Key Wrapping Method to use, the external key token and the input key-encrypting key. If no wrapping method is specified, the system default wrapping method will be used. Note that the *output\_KEK\_identifier* will be ignored.

To convert an operational AES DATA token (version X'04') to an operational AES CIPHER token (version X'05') or vice versa, specify the REFORMAT rule array keyword, the operational key token as *input\_key\_token*, and either a NULL token or skeleton token as *output\_key\_token*. Note that both the *input\_KEK\_identifier* and the *output\_KEK\_identifier* will be ignored as the corresponding lengths must be zero.

**Note:** All key labels must be unique.

## Format

```
CALL CSNBKTR2(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    input_key_length,  
    input_key_token,  
    input_KEK_length,  
    input_KEK_identifier,  
    output_KEK_length,  
    output_KEK_identifier,  
    output_key_length,  
    output_key_token )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is defined in the *exit\_data* parameter.

**exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The count must be between 0 and 4, inclusive.

**rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Keyword	Meaning
<i>Encipherment (optional)</i>	
REFORMAT	Reformat the <i>input_key_token</i> . <ul style="list-style-type: none"> <li>When <i>input_key_token</i> is a DES key token, reformat with the Key Wrapping Method specified.</li> <li>When <i>input_key_token</i> is an operational AES key token, either reformat an AES DATA key (version X'04') to an AES CIPHER key (version X'05') or the reverse (version X'05' to version X'04').</li> </ul>
TRANSLAT	Translate the <i>input_key_token</i> from encipherment under the <i>input_KEK_identifier</i> to encipherment under the <i>output_KEK_identifier</i> . This is the default.
<i>Key Wrapping Method (optional, valid only if input_key_token is an external DES key token)</i>	
USECONFIG	Specifies that the system default configuration should be used to determine the wrapping method. This is the default.  The system default key wrapping method can be specified using the DEFAULTWRAP parameter in the installation options data set. See the <i>z/OS Cryptographic Services ICSF System Programmer's Guide</i> .
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens.
<i>Translation Control (optional, valid only with WRAP-ENH)</i>	



Keyword	Meaning
ENH-ONLY	Restrict rewrapping of the <i>output_key_token</i> . Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method.
<i>Algorithm (optional)</i>	
AES	Specifies that the input key is an AES key. Where used, the key-encrypting keys will be AES transport keys.
DES	Specifies that the input key is a DES key. Where used, the key-encrypting keys will be DES transport keys. This is the default.
HMAC	Specifies that the input key is an HMAC key. Where used, the key-encrypting keys will be AES transport keys.

### input\_key\_length

Direction: Input

Type: Integer

The length of the *input\_key\_token* in bytes. The maximum value allowed is 900.

### input\_key\_token

Direction: Input/Output

Type: String

A variable length string variable containing the key token to be translated or reformatted.

If the REFORMAT keyword is specified and the *input\_key\_token* is an AES CIPHER key (version X'05'), the key must have the following characteristics:

- Key-usage field 1 allows the key to be used for encryption and decryption and has no UDX bits set (UDX bits are not supported in version '04'X AES tokens)
- Key-usage field 2 allows the key to be used for Cipher Block Chaining (CBC) mode or Electronic Code Book (ECB) mode
- Key-management field 1 allows export using symmetric, unauthenticated asymmetric, and authenticated asymmetric transport keys, and allows export using DES, AES, and RSA transport keys
- Key-management field 2 indicates that the key is complete

If the REFORMAT and AES keywords are specified and *input\_key\_token* was encrypted under the old master key, the token will be returned encrypted under the current master key.

### input\_KEK\_length

Direction: Input

Type: Integer

The length of the *input\_KEK\_identifier* in bytes. When the *input\_KEK\_identifier* is a token, the value must be between the actual length of the token and 725. When the *input\_KEK\_identifier* is a label, the value must be 64.

If the REFORMAT keyword is specified, and *input\_key\_token* is an AES key token, this parameter must be zero.

### input\_KEK\_identifier

Direction: Input/Output

Type: String

A variable length string variable containing the internal key token or the key label of an internal key token record in the CKDS. The internal key token contains the key-encrypting key used to decipher the key.

If *input\_KEK\_length* is zero, this parameter is ignored.

If the TRANSLAT keyword is specified and the *input\_key\_token* is an external DES key, the *input\_KEK\_identifier* must be an internal DES token that contains a control vector that specifies an IMPORTER or IKEYXLAT key type. The control vector for an IMPORTER key must have the XLATE bit set to 1.

If the TRANSLAT keyword is specified and the *input\_key\_token* is an external variable-length key token, the *input\_KEK\_identifier* must be an internal variable-length key token containing an IMPORTER key-encrypting key. The IMPORTER key must have the TRANSLAT bit on in key-usage field 1 of the token.

If the REFORMAT keyword is specified and *input\_key\_token* is an external DES key token, this parameter may be an IMPORTER, IKEYXLAT, EXPORTER, or OKEYXLAT key type.

If an internal token was supplied and was encrypted under the old master key, the token will be returned encrypted under the current master key.

#### **output\_KEK\_length**

Direction: Input

Type: Integer

The length of the *output\_KEK\_identifier* in bytes. When the *output\_KEK\_identifier* is a token, the value must be between the actual length of the token and 725. When the *output\_KEK\_identifier* is a label, the value must be 64.

If the REFORMAT keyword is specified, this value must be zero.

#### **output\_KEK\_identifier**

Direction: Input/Output

Type: String

A variable length string variable containing the internal key token or the key label of an internal key token record in the CKDS. The internal key token contains the key-encrypting key used to encipher the key.

If *output\_KEK\_length* is zero, this parameter is ignored.

If the *output\_key\_token* is an external DES key, the *output\_KEK\_identifier* must be an internal DES token that contains a control vector that specifies an EXPORTER or OKEYXLAT key type. The control vector for an EXPORTER key must have the XLATE bit set to 1.

If the *input\_key\_token* is an external variable-length key token, the *output\_KEK\_identifier* must be an internal variable-length key token containing an EXPORTER key-encrypting key. The EXPORTER key must have the TRANSLAT bit on in key-usage field 1 of the token.

If an internal token was supplied and was encrypted under the old master key, the token will be returned encrypted under the current master key.

#### **output\_key\_length**

Direction: Input/Output

Type: Integer

## Key Translate2

On input, the length of the output area provided for the *output\_key\_token*. This must be between 64 and 900 bytes and provide sufficient space for the output key. On output, the parameter is updated with the length of the token copied to the *output\_key\_token*.

### output\_key\_token

Direction: Input/Output

Type: String

If the REFORMAT keyword is specified and the *input\_key\_token* is an AES DATA key (version X'04'), *output\_key\_token* must contain an AES CIPHER key (version X'05') on input. This token must have the following characteristics:

- Algorithm is AES
- Key type CIPHER
- Key-usage field 2 either allows the key to be used for Cipher Block Chaining (CBC) mode or allows the key to be used for Electronic Code Book (ECB) mode

Otherwise, this field is ignored on input.

On output, a variable length string variable containing the key token that was translated or reformatted.

If the REFORMAT keyword is specified and the *input\_key\_token* is an AES DATA key (version X'04'), on output, *output\_key\_token* will be updated with the following characteristics:

- Key-usage field 1 allows the key to be used for encryption and decryption
- Key-management field 1 allows export using symmetric, unauthenticated asymmetric, and authenticated asymmetric transport keys, and allows export using DES, AES, and RSA transport keys
- Key-management field 2 indicates that the key is complete

## Restrictions

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

## Access Control Points

This table lists the access control points in the ICSF default role that control the function for this service.

Table 71. Key Translate2 Access Control Points

Access Control point	Function control
Key Translate2	Allows the Key Translate2 service to be functional.
Key Translate2 – Allow use of REFORMAT	Allows a key token to be rewrapped using one key-encrypting key.
Key Translate2 – Allow wrapping method override keywords	Allows the wrapping method keywords WRAP-ECB or WRAP-ENH to be used when the default key-wrapping method setting does not match the keyword.

When the **Key Translate2 - Disallow AES ver 5 to ver 4 conversion** access control point is enabled, a version 5 AES key token (variable-length token) can not be converted to a version 4 token.

If the output key-encrypting key identifier is a weaker key than the key being translated, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the input key is a triple-length DATA key and the output key-encrypting key identifier key is a 16-byte key.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 72. Key Translate2 required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	This service is not supported.
IBM @server zSeries 990	None.	This service is not supported.
IBM @server zSeries 890		
IBM System z9 EC	None.	This service is not supported.
IBM System z9 BC		
IBM System z10 EC	Crypto Express2 Coprocessor	This service is not supported.
IBM System z10 BC	Crypto Express3 Coprocessor	This service is not supported.
z196 z114	Crypto Express3 Coprocessor	Enhanced key token wrapping and HMAC key support requires the Nov. 2010 or later licensed internal code (LIC).  AES key support requires the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

---

### Multiple Clear Key Import (CSNBCKM and CSNECKM)

The multiple clear key import callable service imports a clear AES or DES key, enciphers the key under the corresponding master key, and returns the enciphered key in an internal key token. The enciphered key's type is DATA. The enciphered key is in operational form.

The callable service name for AMODE(64) invocation is CSNECKM.

#### Format

```
CALL CSNBCKM(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    clear_key_length,  
    clear_key,  
    key_identifier_length,  
    key_identifier )
```

#### Parameters

##### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

##### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

##### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

##### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

##### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The *rule\_array\_count* parameter must be 0, 1, 2, or 3. If the *rule\_array\_count* is 0, the default keywords are used.

**rule\_array**

Direction: Input

Type: String

Keywords that supply control information to the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. Refer to Table 73 for a list of keywords.

*Table 73. Keywords for Multiple Clear Key Import Rule Array Control Information*

Keyword	Meaning
<i>Algorithm (optional)</i>	
CDMF	The output key identifier is to be a CDMF token. For a DATA key of length 16 or 24, you may not specify CDMF.  CDMF is only supported on CCF systems.
AES	The output key identifier is to be an AES token.
DES	The output key identifier is to be a DES token. This is the default.
<i>Key Wrapping Method (optional)</i>	
USECONFIG	Specifies that the system default configuration should be used to determine the wrapping method. This is the default keyword.  The system default key wrapping method can be specified using the DEFAULTWRAP parameter in the installation options data set. See the <i>z/OS Cryptographic Services ICSF System Programmer's Guide</i> .
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens.
<i>Translation Control (optional)</i>	
ENH-ONLY	Restrict rewrapping of the <i>key_identifier</i> token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method.

**clear\_key\_length**

Direction: Input

Type: Integer

The *clear\_key\_length* specifies the length of the clear key value to import in bytes. For DES keys, this length must be 8-, 16-, or 24-bytes. For AES keys, this length must be 16-, 24-, or 32-bytes.

**clear\_key**

Direction: Input

Type: String

The *clear\_key* specifies the clear key value to import.

## Multiple Clear Key Import

### **key\_identifier\_length**

Direction: Input/Output

Type: Integer

The byte length of the *key\_identifier* parameter. This must be exactly 64 bytes.

### **key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string that is to receive an internal AES or DES key token.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *key\_identifier* will use the default method unless a rule array keyword overriding the default is specified.

## Usage Notes

This service produces an internal DES DATA token with a control vector which is usable on the Cryptographic Coprocessor Feature. If a valid internal DES token is supplied as input to the service in the *key\_identifier* field, that token's control vector will not be used in the encryption of the clear key value.

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

*Table 74. Required access control points for Multiple Clear Key Import*

Key algorithm	Access control point
DES	Clear Key Import/Multiple Clear Key Import – DES
AES	Multiple Clear Key Import/Multiple Secure Key Import – AES

When the WRAP-ECB or WRAP-ENH keywords are specified and default key-wrapping method setting does not match the keyword, the **Multiple Clear Key Import - Allow wrapping override keywords** access control point must be enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 75. Multiple clear key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	<p>Tokens are not marked with the system encryption algorithm.</p> <p>ENH-ONLY, USECONFIG, WRAP-ENH and WRAP-ECB not supported.</p> <p>Enhanced key token wrapping not supported.</p>
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	<p>CDMF keyword is not supported. Tokens are not marked with the system encryption algorithm.</p> <p>ENH-ONLY, USECONFIG, WRAP-ENH and WRAP-ECB not supported.</p> <p>Enhanced key token wrapping not supported.</p>
IBM @server zSeries 890	Crypto Express2 Coprocessor	<p>CDMF keyword is not supported. Tokens are not marked with the system encryption algorithm.</p> <p>ENH-ONLY, USECONFIG, WRAP-ENH and WRAP-ECB not supported.</p> <p>Enhanced key token wrapping not supported.</p>
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	<p>CDMF keyword is not supported. Tokens are not marked with the system encryption algorithm.</p> <p>Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).</p> <p>ENH-ONLY, USECONFIG, WRAP-ENH and WRAP-ECB not supported.</p> <p>Enhanced key token wrapping not supported.</p>
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	<p>CDMF keyword is not supported. Tokens are not marked with the system encryption algorithm.</p> <p>Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).</p> <p>ENH-ONLY, USECONFIG, WRAP-ENH and WRAP-ECB not supported.</p> <p>Enhanced key token wrapping not supported.</p>
	Crypto Express3 Coprocessor	Enhanced key token wrapping not supported.
z196 z114	Crypto Express3 Coprocessor	CDMF keyword is not supported. Tokens are not marked with the system encryption algorithm.
IBM zEnterprise EC12	Crypto Express3 Coprocessor	CDMF keyword is not supported. Tokens are not marked with the system encryption algorithm.
	Crypto Express4 Coprocessor	



### Multiple Secure Key Import (CSNBSKM and CSNESKM)

Use this service to encipher a single-length, double-length, or triple-length DES key under the system master key or an importer key-encrypting key. The clear DES key can then be imported as any of the possible key types.

In addition to DES keys, this service imports a clear AES key, enciphers the AES key under the AES master key, and returns the enciphered key in an internal token. The enciphered key's type is DATA. The enciphered key is in operational form.

The callable service can execute only when ICSF is in special secure mode, which is described in "Special Secure Mode" on page 10.

The callable service name for AMODE(64) invocation is CSNESKM.

### Format

```
CALL CSNBSKM(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    clear_key_length,  
    clear_key,  
    key_type,  
    key_form,  
    key_encrypting_key_identifier,  
    imported_key_identifier_length,  
    imported_key_identifier )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

**exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The *rule\_array\_count* parameter must be 0, 1, 2, 3, or 4. If the *rule\_array\_count* is 0, the default keywords are used.

**rule\_array**

Direction: Input

Type: String

Keywords that supply control information to the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The keywords are shown in Table 76.

The first keyword is the algorithm. If no algorithm is specified, the system default algorithm is used. If no algorithm is specified on a CDMF only system and either a double- or triple-length DATA key is specified, the token is marked DES. The algorithm keyword applies only when the desired output token is of key form OP and key type IMPORTER, EXPORTER, or DATA. For key form IM or any other key type, specifying DES or CDMF causes an error.

The second keyword is optional and specifies that the output key token be marked as an NOCV-KEK.

The third keyword is optional, and specifies whether the original key wrapping method or the enhanced key wrapping method (which is compliant with the ANSI X9.24 standard) should be used.

The fourth keyword enables an application to specify that the *imported\_key\_identifier* output token can not be rewrapped using the original wrapping method after it has been wrapped using the enhanced method.

Table 76. Keywords for Multiple Secure Key Import Rule Array Control Information

Keyword	Meaning
<i>Algorithm (optional)</i>	
CDMF	The output key identifier is to be a CDMF token. For a DATA key of length 16 or 24, you may not specify CDMF.  CDMF is only supported on CCF systems.
AES	The output key identifier is to be a AES token.
DES	The output key identifier is to be a DES token. This is the default.
<i>NOCV Choice (optional)</i>	
NOCV-KEK	The output token is to be marked as an NOCV-KEK. This keyword only applies if key form is OP and key type is IMPORTER, EXPORTER or IMP-PKA. For key form IM or any other key type, specifying NOCV-KEK causes an error.
<i>Key Wrapping Method (optional)</i>	

## Multiple Secure Key Import

Table 76. Keywords for Multiple Secure Key Import Rule Array Control Information (continued)

Keyword	Meaning
USECONFIG	Specifies that the system default configuration should be used to determine the wrapping method. This is the default keyword.  The system default key wrapping method can be specified using the DEFAULTWRAP parameter in the installation options data set. See the <i>z/OS Cryptographic Services ICSF System Programmer's Guide</i> .
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens.
<b>Translation Control (optional)</b>	
ENH-ONLY	Restrict rewrapping of the <i>imported_key_identifier</i> token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method.

### clear\_key\_length

Direction: Input

Type: Integer

The *clear\_key\_length* specifies the length of the clear key value to import in bytes. For AES keys, this length must be 16-, 24-, or 32-bytes. For DES keys, this length must be 8-, 16- or 24-bytes.

### clear\_key

Direction: Input

Type: String

The *clear\_key* specifies the AES or DES clear key value to import.

### key\_type

Direction: Input

Type: 8 Character String

The type of key you want to encipher under the master key or an importer key. Specify an 8-byte field that must contain a keyword from this list or the keyword TOKEN. For types with fewer than 8 characters, the type should be padded on the right with blanks. If the key type is TOKEN, ICSF determines the key type from the control vector (CV) field in the internal key token provided in the *imported\_key\_identifier* parameter. When *key\_type* is TOKEN, ICSF does not check for the length of the key but uses the *clear\_key\_length* parameter to determine the length of the key.

Key type values for the Multiple Secure Key Import callable service are: CIPHER, CIPHERXI, CIPHERXL, CIPHERXO, CVARDEC, CVARENC, CVARPINE, CVARXCVL, CVARXCVR, DATA, DATAM, DATAMV, DATAXLAT, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IMP-PKA, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER. For information on the meaning of the key types, see Table 3 on page 23.

**key\_form**

Direction: Input

Type: 4 Character String

The key form you want to generate. Enter a 4-byte keyword specifying whether the key should be enciphered under the master key (OP) or the importer key-encrypting key (IM). The keyword must be left-justified and padded with blanks. Valid DES keyword values are OP for encryption under the master key or IM for encryption under the importer key-encrypting key. If you specify IM, you must specify an importer key-encrypting key in the *key\_encrypting\_key\_identifier* parameter. For a *key\_type* of IMP-PKA, this service supports only the OP *key\_form*.

The only valid AES keyword value is OP.

**key\_encrypting\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string internal key token or key label of a DES importer key-encrypting key. This parameter is ignored for AES secure keys.

**imported\_key\_identifier\_length**

Direction: Input/Output

Type: Integer

The byte length of the *imported\_key\_identifier* parameter. This must be at least 64.

**imported\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string that is to receive the output key token. If OP is specified in the *key\_form* parameter, the service returns an internal key token. If IM is specified in the *key\_form* parameter, the service returns an external key token. On input, this parameter is ignored except when the *key\_type* is TOKEN. If you specify a *key\_type* of TOKEN, then this field contains a valid token of the key type you want to encipher. See *key\_type* for a list of valid key types. Appendix B, "Key Token Formats," on page 835 describes the key tokens.

Note that for a DATA key of length 16 or 24, no reference will be made to the data encryption algorithm bits or to the system's default algorithm; the token will be marked DES.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *imported\_key\_identifier* will use the default method unless a rule array keyword overriding the default is specified.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

On CCF systems, to generate double-length DATAM and DATAMV keys in the importable form, the ANSI system keys must be installed in the CKDS.

CDMF is only supported on CCF systems.

## Multiple Secure Key Import

With a PCIXCC or CCA Crypto Express coprocessor, creation of a DES NOCV key-encrypting key is only available for standard IMPORTERs and EXPORTERs.

On an IBM @server zSeries 990, if *key\_form* of the DES key is IM and the *key\_encrypting\_key\_identifier* is a NOCV KEK, then the NOCV IMPORTER access control point must be enabled in the PCIXCC to use the function.

For key types CIPHERXI, CIPHERXL, and CIPHERXO, the key-encrypting key in the *key\_encrypting\_key\_identifier* parameter must have a control vector with the key halves guaranteed unique flag on in the key form bits. An existing key-encrypting key can have its control vector updated using the restrict key attribute callable service.

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

Table 77. Required access control points for Multiple Secure Key Import

Key Algorithm and Key Form	Access control point
DES OP	Secure Key Import - DES, OP
DES IM	Secure Key Import - DES, IM
AES OP	Multiple Clear Key Import/Multiple Secure Key Import – AES

To use a NOCV key-encrypting key with the Multiple Secure Key Import service, the **NOCV KEK usage for import-related functions** access control point must be enabled in addition to one or both of the access control points listed.

When the WRAP-ECB or WRAP-ENH keywords are specified and default key-wrapping method setting does not match the keyword, the **Multiple Secure Key Import - Allow wrapping override** keywords access control point must be enabled.

If the key-encrypting key identifier is a weaker key than the key being imported, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key or the key-encrypting key is a double-length key.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 78. Multiple secure key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	<p>Only control vectors and key types supported by the Cryptographic Coprocessor Feature will be valid when importing a triple-length key.</p> <p>ICSF routes the request to a PCI Cryptographic Coprocessor if the control vector of a supplied internal token cannot be processed on the Cryptographic Coprocessor Feature, or if the key type is not valid for the Cryptographic Coprocessor Feature.</p> <p>Key types DATAC, CIPHERXI, CIPHERXL, and CIPHERXO are not supported.</p> <p>ENH-ONLY, USECONFIG, WRAP-ENH and WRAP-ECB not supported.</p> <p>Enhanced key token wrapping not supported.</p>
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	<p>Key types DATAXLAT, CIPHERXI, CIPHERXL, and CIPHERXO are not supported.</p>
IBM @server zSeries 890	Crypto Express2 Coprocessor	<p>CDMF keyword is not supported. DATA and KEK tokens are not marked with the system encryption algorithm.</p> <p>ENH-ONLY, USECONFIG, WRAP-ENH and WRAP-ECB not supported.</p> <p>Enhanced key token wrapping not supported.</p>
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	<p>Key types DATAXLAT, CIPHERXI, CIPHERXL, and CIPHERXO are not supported.</p> <p>CDMF keyword is not supported. DATA and KEK tokens are not marked with the system encryption algorithm.</p> <p>Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).</p> <p>ENH-ONLY, USECONFIG, WRAP-ENH and WRAP-ECB not supported.</p> <p>Enhanced key token wrapping not supported.</p>

## Multiple Secure Key Import

Table 78. Multiple secure key import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	Key types DATAXLAT, CIPHERXI, CIPHERXL, and CIPHERXO are not supported.  CDMF keyword is not supported. DATA and KEK tokens are not marked with the system encryption algorithm.  Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).  ENH-ONLY, USECONFIG, WRAP-ENH and WRAP-ECB not supported.  Enhanced key token wrapping not supported.
	Crypto Express3 Coprocessor	Key types DATAXLAT, CIPHERXI, CIPHERXL, and CIPHERXO are not supported.  CDMF keyword is not supported. DATA and KEK tokens are not marked with the system encryption algorithm.  Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).  Enhanced key token wrapping not supported.
z196 z114	Crypto Express3 Coprocessor	Key types DATAXLAT, CIPHERXI, CIPHERXL, and CIPHERXO are not supported.  CDMF keyword is not supported. DATA and KEK tokens are not marked with the system encryption algorithm.
IBM zEnterprise EC12	Crypto Express3 Coprocessor	Key type DATAXLAT is not supported.  CDMF keyword is not supported. DATA and KEK tokens are not marked with the system encryption algorithm.
	Crypto Express4 Coprocessor	

## PKA Decrypt (CSNDPKD and CSNFPKD)

Use this service to decrypt (unwrap) a formatted key value. The service unwraps the key, deformats it, and returns the deformatted value to the application in the clear. PKCS 1.2 and ZERO-PAD formatting is supported. For PKCS 1.2, the decrypted data is examined to ensure it meets RSA DSI PKCS #1 block type 2 format specifications.

For PKA private keys, this service allows the use of clear or encrypted RSA private keys. If an external clear key token is used, the master keys are not required to be installed in any cryptographic coprocessor and PKA callable services does not have

to be enabled. Requests are routed to a Cryptographic Accelerator if available when a clear key token is used. ZERO-PAD is only supported for external RSA clear private keys.

This service also supports the use of secure PKCS #11 private keys, which requires an active Enterprise PKCS #11 coprocessor. PKCS 1.2 formatting is supported.

The callable service name for AMODE(64) invocation is CSNFPKD.

## Format

```
CALL CSNDPKD(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    PKA_enciphered_keyvalue_length,
    PKA_enciphered_keyvalue,
    data_structure_length,
    data_structure,
    key_identifier_length,
    key_identifier,
    target_keyvalue_length,
    target_keyvalue)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.



## PKA Decrypt

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. This value must be 1.

### **rule\_array**

Direction: Input

Type: String

The keyword that provides control information to the callable service. The keyword is left-justified in an 8-byte field and padded on the right with blanks.

Table 79. Keywords for PKA Decrypt

Keyword	Meaning
<i>Recovery Method (required)</i> specifies the method to use to recover the key value.	
PKCS-1.2	RSA DSI PKCS #1 block type 02 will be used to recover the key value.
ZERO-PAD	<p>The input <i>PKA_enciphered_keyvalue</i> is decrypted using the RSA private key. The entire result (including leading zeros) will be returned in the <i>target_keyvalue</i> field. For PKA keys, the <i>key_identifier</i> must be an external RSA token or the labelname of a external token. This keyword requires May 2004 or later version of Licensed Internal Code (LIC) or a z890.</p> <p>This support on the PCICA does not require LIC code updates.</p> <p>This keyword is not valid when using a secure PKCS #11 private key.</p>

### **PKA\_enciphered\_keyvalue\_length**

Direction: Input

Type: integer

The length of the *PKA\_enciphered\_keyvalue* parameter in bytes. The maximum size that you can specify is 512 bytes. The length should be the same as the modulus length of the *key\_identifier*.

### **PKA\_enciphered\_keyvalue**

Direction: Input

Type: String

This field contains the key value protected under an RSA public key. This byte-length string is left-justified within the *PKA\_enciphered\_keyvalue* parameter.

### **data\_structure\_length**

Direction: Input

Type: Integer

The value must be 0.

### **data\_structure**

Direction: Input

Type: String

This field is currently ignored.

### **key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *key\_identifier* parameter. When the *key\_identifier* is a key label, this field specifies the length of the label. The maximum size that you can specify is 3500 bytes.

### **key\_identifier**

Direction: Input

Type: String

For PKA keys, an internal RSA private key token, the label of an internal RSA private key token, or an external RSA private key token containing a clear RSA private key in modulus-exponent or Chinese Remainder format.

For secure PKCS #11 keys, this is the 44-byte handle of the private key, prefixed with an EBCDIC equal sign character ('=' or x'7E'), and padded on the right with spaces for a total length of 64 bytes.

The corresponding public key was used to wrap the key value.

### **target\_keyvalue\_length**

Direction: Input/Output

Type: Integer

The length of the *target\_keyvalue* parameter. The maximum size that you can specify is 512 bytes. On return, this field is updated with the actual length of *target\_keyvalue*.

If ZERO-PAD is specified, this length will be the same as the RSA modulus byte length.

### **target\_keyvalue**

Direction: Output

Type: String

This field will contain the decrypted, deformatted key value. If ZERO-PAD is specified, the decrypted key value, including leading zeros, will be returned.

## **Restrictions**

The exponent of the RSA public key must be odd.

Access control checking will not be performed in the PCI Cryptographic Coprocessor when a clear external key token is supplied.

## **Authorization**

To use this service with a secure PKCS #11 private key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access) to the containing PKCS #11 token.

To use this service with a secure PKCS #11 private key that is a private object, the caller must have USER (READ) authority (user access) to the containing PKCS #11 token.

## PKA Decrypt

See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the SO and User PKCS #11 roles.

### Usage Notes — General

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS, PKDS, or TKDS.

PKA RSA private key must be enabled for key management functions. Secure PKCS #11 private keys must be enabled for decryption.

For PKA keys, the hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit.

### Usage Notes — CCF Systems

When routing requests to coprocessors for systems with CCFs, this service examines the RSA key specified in the *PKA\_key\_identifier* parameter to determine how to route the request.

- If the modulus bit length is less than 512 bits, or if the key is a X'02' form modulus-exponent private key, ICSF routes the request to the Cryptographic Coprocessor Feature.
- If the key is a X'08' form CRT private key or a retained private key, the service routes the request to a PCI Cryptographic Coprocessor.
- In the case of a retained key, the service routes the request to the specific PCI Cryptographic Coprocessor in which the key is retained.
- If the key is a modulus-exponent form private key with a private section ID of X'06', then the service routes the request as follows:
  - Since the key must be a key-management key, if the KMMK is equal to the SMK on the Cryptographic Coprocessor Feature, the PKA decrypt service uses load balancing to route the request to either a Cryptographic Coprocessor Feature or to an available PCI Cryptographic Coprocessor.
  - If the KMMK is not equal to the SMK on the Cryptographic Coprocessor Feature, the request must be processed on a PCI Cryptographic Coprocessor. If there is no PCI Cryptographic Coprocessor online, the request will fail.
- If the key is an external clear key, the request is routed in this order of preference:
  - PCICA
  - PCICC
  - CCF

### Access Control Points

For PKA keys, the **PKA Decrypt** access control point controls the function of this service.

For secure PKCS #11 private keys, see PKCS #11 Access Control Points in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the access control points of the Enterprise PKCS #11 coprocessor.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 80. PKA decrypt required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	ICSF routes the request to the Cryptographic Coprocessor Feature if the modulus bit length is less than 512 bits, or if the key is a X'02' form modulus-exponent private key.  The ZERO-PAD keyword is not supported.  RSA keys with moduli greater than 1024-bit length are not supported.
	PCI Cryptographic Coprocessor	This service routes the request to the PCI Cryptographic Coprocessor in which the key is retained if the key is a X'08' form CRT private key or a retained private key  The ZERO-PAD keyword is not supported.  RSA keys with moduli greater than 2048-bit length are not supported.
	PCI Cryptographic Accelerator	Only clear RSA private keys are supported.  The ZERO-PAD keyword is not supported.  RSA keys with moduli greater than 2048-bit length are not supported.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	Old RSA private tokens encrypted under the CCF KMMK are not usable on the PCIXCC/CEX2C if the KMMK was not same as the ASYM-MK.
IBM @server zSeries 890	Crypto Express2 Coprocessor	RSA keys with moduli greater than 2048-bit length are not supported.
	PCI Cryptographic Accelerator	Only clear RSA private keys are supported.  RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Old RSA private tokens encrypted under the CCF KMMK are not usable on the CEX2C if the KMMK was not same as the ASYM-MK.  RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express2 Accelerator	Only clear RSA private keys are supported.  RSA keys with moduli greater than 2048-bit length are not supported.

## PKA Decrypt

Table 80. PKA decrypt required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	Old RSA private tokens encrypted under the CCF KMMK are not usable on the CEX2C or CEX3C if the KMMK was not same as the ASYM-MK.  RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express3 Coprocessor	
	Crypto Express2 Accelerator Crypto Express3 Accelerator	Only clear RSA private keys are supported.  RSA keys with moduli greater than 2048-bit length are not supported.
z196 z114	Crypto Express3 Coprocessor	Old RSA private tokens encrypted under the CCF KMMK are not usable on the CEX3C if the KMMK was not same as the ASYM-MK.
	Crypto Express3 Accelerator	Only clear RSA private keys are supported.  RSA clear key support with moduli within the range 2048-bit to 4096-bit requires the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12	Crypto Express3 Coprocessor	Old RSA private tokens encrypted under the CCF KMMK are not usable on the CEX3C if the KMMK was not same as the ASYM-MK.
	Crypto Express4 CCA Coprocessor (CEX4C)	
	Crypto Express3 Accelerator Crypto Express4 Accelerator (CEX4A)	Only clear RSA private keys are supported. RSA clear key support with moduli within the range 2048-bit to 4096-bit requires the Sep. 2011 or later licensed internal code (LIC).
	Crypto Express4 Enterprise PKCS #11 coprocessor (CEX4P)	
		Required to use a secure PKCS #11 private key.

## PKA Encrypt (CSNDPKE and CSNFPKE)

This callable service encrypts a supplied clear key value under an RSA public key. The rule array keyword specifies the format of the key prior to encryption.

On the z900 and if the ZERO-PAD or MRP keyword is specified, this service is routed to a PCI Cryptographic Accelerator.

The callable service name for AMODE(64) invocation is CSNFPKE.

## Format

```
CALL CSNDPKE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    keyvalue_length,
    keyvalue,
    data_structure_length,
    data_structure,
    PKA_key_identifier_length,
    PKA_key_identifier,
    PKA_enciphered_keyvalue_length,
    PKA_enciphered_keyvalue)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. This value can be 1 or 2.

### **rule\_array**

## PKA Encrypt

Direction: Input

Type: String

A keyword that provides control information to the callable service. The keyword is left-justified in an 8-byte field and padded on the right with blanks.

Table 81. Keywords for PKA Encrypt

Keyword	Meaning
<i>Formatting Method (required)</i> specifies the method to use to format the key value prior to encryption.	
PKCS-1.2	RSA DSI PKCS #1 block type 02 format will be used to format the supplied key value.
ZERO-PAD	The key value will be padded on the left with binary zeros to the length of the PKA key modulus. The exponent of the public key must be odd.
MRP	The key value will be padded on the left with binary zeros to the length of the PKA key modulus. The RSA public key may have an even or odd exponent. This keyword requires May 2004 or later version of Licensed Internal Code (LIC) or a z890.  For PCICAs, the LIC code update is not required.
<i>Key Rule (Optional)</i>	
KEYIDENT	This indicates that the value in the <i>keyvalue</i> field is the label of clear tokens in the CKDS. The <i>keyvalue_length</i> must be 64.

### keyvalue\_length

Direction: Input

Type: Integer

The length of the *keyvalue* parameter. The maximum field size is 512 bytes. The actual maximum size depends on the modulus length of *PKA\_key\_identifier* and the formatting method you specify in the *rule\_array* parameter. When key rule KEYIDENT is specified, then the *keyvalue\_length* parameter is required to be 64 bytes.

### keyvalue

Direction: Input

Type: String

This field contains the supplied clear key value to be encrypted under the *PKA\_key\_identifier*. When key rule KEYIDENT is specified, the *keyvalue* parameter is assumed to contain a label for a valid CKDS clear key token.

### data\_structure\_length

Direction: Input

Type: Integer

This value must be 0.

### data\_structure

Direction: Input

Type: String

This field is currently ignored.

### PKA\_key\_identifier\_length

Direction: Input

Type: Integer

The length of the *PKA\_key\_identifier* parameter. When the *PKA\_key\_identifier* is a key label, this field specifies the length of the label. The maximum size that you can specify is 3500 bytes.

### PKA\_key\_identifier

Direction: Input

Type: String

The RSA public or private key token or the label of the RSA public or private key to be used to encrypt the supplied key value.

### PKA\_enciphered\_keyvalue\_length

Direction: Input/Output

Type: integer

The length of the *PKA\_enciphered\_keyvalue* parameter in bytes. The maximum size that you can specify is 512 bytes. On return, this field is updated with the actual length of *PKA\_enciphered\_keyvalue*.

This length should be the same as the modulus length of the *PKA\_key\_identifier*.

### PKA\_enciphered\_keyvalue

Direction: Output

Type: String

This field contains the key value protected under an RSA public key. This byte-length string is left-justified within the *PKA\_enciphered\_keyvalue* parameter.

## Restrictions

The exponent for RSA public keys must be odd. When the modulus is greater than 2048, the public key exponent must be 3 or 65537.

## Usage Notes

- SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.
- For RSA DSI PKCS #1 formatting, the key value length must be at least 11 bytes less than the modulus length of the RSA key.
- The hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit.
- The key value to be encrypted must be smaller than the modulus in the *PKA\_key\_identifier*.

## Access Control Point

The **PKA Encrypt** access control point controls the function of this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.



Table 82. PKA encrypt required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	The MRP keyword is not supported.  RSA keys with moduli greater than 1024-bit length are not supported.
	PCI Cryptographic Coprocessor	If the modulus bit length of the key specified in the <i>PKA_key_identifier</i> parameter is greater than 1024, the request is routed to the PCICC.  The MRP keyword is not supported.  RSA keys with moduli greater than 2048-bit length are not supported.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	Routed to a PCICA if one is available (ZERO-PAD and MRP only).
IBM @server zSeries 890	Crypto Express2 Coprocessor	RSA keys with moduli greater than 2048-bit length are not supported.
	PCI Cryptographic Accelerator	PKCS-1.2 keyword not supported.  RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Routed to a CEX2A if one is available (ZERO-PAD and MRP only).  RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express2 Accelerator	PKCS-1.2 keyword not supported.  RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	Routed to a CEX2A or CEX3A if one is available (ZERO-PAD and MRP only).
	Crypto Express3 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express2 Accelerator	PKCS-1.2 keyword not supported.
	Crypto Express3 Accelerator	RSA keys with moduli greater than 2048-bit length are not supported.
z196 z114	Crypto Express3 Coprocessor	Routed to a CEX2A or CEX3A if one is available (ZERO-PAD and MRP only).
	Crypto Express3 Accelerator	PKCS-1.2 keyword not supported.  RSA clear key support with moduli within the range 2048-bit to 4096-bit requires the Sep. 2011 or later licensed internal code (LIC).

Table 82. PKA encrypt required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise EC12	Crypto Express3 Coprocessor	Routed to a CEX3A or CEX4A if one is available (ZERO-PAD and MRP only).
	Crypto Express4 Coprocessor	
	Crypto Express3 Accelerator	PKCS-1.2 keyword not supported.
	Crypto Express4 Accelerator	

## Prohibit Export (CSNBPEX and CSNEPEX)

Use this service to modify an exportable internal DES key token so that it cannot be exported.

The callable service name for AMODE(64) invocation is CSNEPEX.

### Format

```
CALL CSNBPEX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier)
```

### Parameters

#### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### reason\_code

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

#### exit\_data\_length

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

## Prohibit Export

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string variable containing the internal key token to be modified. The returned *key\_identifier* will be encrypted under the current master key.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *key\_identifier* will be wrapped in the same manner as the input *key\_identifier*.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

## Access Control Point

The **Prohibit Export** access control point controls the function of this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 83. Prohibit export required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	On a PCI Cryptographic Coprocessor, the Prohibit Export service does not support NOCV key-encrypting keys, or DATA, DATAM, DATAMV, MAC, or MACVER keys with standard control vectors (for example, control vectors supported by the Cryptographic Coprocessor Feature).
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.

Table 83. Prohibit export required hardware (continued)

Server	Required cryptographic hardware	Restrictions
z196 z114	Crypto Express3 Coprocessor	DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.

## Prohibit Export Extended (CSNBPEXX and CSNEPEXX)

Use the prohibit export extended callable service to change the external token of a cryptographic key in exportable DES key token form so that it can be imported at the receiver node and is non-exportable from that node. You cannot prohibit export of DATA keys.

The inputs are an external token of the key to change in the *source\_key\_token* parameter and the label or internal token of the exporter key-encrypting key in the *KEK\_key\_identifier* parameter.

This service is a variation of the Prohibit Export service (CSNBPEX and CSNEPEX), which supports changing an *internal* token.

The callable service name for AMODE(64) invocation is CSNEPEXX.

## Format

```
CALL CSNBPEXX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    source_key_token,
    KEK_key_identifier)
```

## Parameters

### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### reason\_code

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned

## Prohibit Export Extended

to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **source\_key\_token**

Direction: Input/Output

Type: String

A 64-byte string of an external token of a key to change. It is in exportable form.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *source\_key\_token* will be wrapped in the same manner as the input *source\_key\_token*.

### **KEK\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string of an internal token or label of the exporter KEK used to encrypt the key contained in the external token specified in the previous parameter.

## Restrictions

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

## Access Control Point

The **Prohibit Export Extended** access control point controls the function of this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 84. Prohibit export extended required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	External MACD keys are not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	External MACD keys are not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	External MACD keys are not supported.
z196 z114	Crypto Express3 Coprocessor	External MACD keys are not supported.
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	External MACD keys are not supported.

## Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)

The callable service uses a cryptographic feature to generate a random number. The foundation for the random number generator is a time variant input with a very low probability of recycling.

There are two forms of the Random Number Generate callable service. One version returns an 8-byte random number. The second version allows the caller to specify the length of the random number.

The callable service names for AMODE(64) invocation are CSNERNG and CSNERNGL.

### Format

```
CALL CSNBRNG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    form,
    random_number )
```

## Random Number Generate

```
CALL CSNBRNGL(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    reserved_length,  
    reserved,  
    random_number_length,  
    random_number )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

#### **form**

Direction: Input

Type: Character string

The 8-byte keyword that defines the characteristics of the random number should be left-justify and pad on the right with blanks. The keywords are listed in Table 85.

*Table 85. Keywords for the Form Parameter*

Keyword	Meaning
EVEN	Generate a 64-bit random number with even parity in each byte.

Table 85. Keywords for the Form Parameter (continued)

Keyword	Meaning
ODD	Generate a 64-bit random number with odd parity in each byte.
RANDOM	Generate a 64-bit random number.

Parity is calculated on the 7 high-order bits in each byte and is presented in the low-order bit in the byte.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The value must be one.

**rule\_array**

Direction: Input

Type: String

The keyword that provides control information to the callable service. The recovery method is the method to use to recover the symmetric key. The keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

Table 86. Keywords for Random Number Generate Control Information

Keyword	Meaning
<i>Parity of the random number bytes (required)</i>	
EVEN	Generate a random number with even parity in each byte. Its length is the <i>random_number_length</i> .
ODD	Generate a random number with odd parity in each byte. Its length is the <i>random_number_length</i> .
RANDOM	Generate a random number. Its length is the <i>random_number_length</i> .

**reserved\_length**

Direction: Input

Type: Integer

This parameter must be zero.

**reserved**

Direction: Input

Type: Integer

This parameter is ignored.

**random\_number\_length**

Direction: Input/Output

Type: Integer

This parameter contains the desired length of the *random\_number* that is returned by the CSNBRNGL callable service. The minimum value is 1 byte; the maximum value is 8192 bytes.



## Random Number Generate

**random\_number**

Direction: Output

Type: String

The generated number returned by the CSNBRNG callable service is stored in an 8-byte variable.

The generated number returned by the CSNBRNGL callable service is stored in a variable that is at least *random\_number\_length* bytes long.

### Usage Notes

The CSNBRNGL callable service returns a value under the following conditions:

- The server has the cryptographic coprocessor that supports CSNBRNGL and the coprocessor creates the random number with the desired length. This requires a CCA Crypto Express coprocessor with a version of the licensed internal code (LIC) that supports the RNGL verb.
- The server has the cryptographic coprocessor that processes CSNBRNG requests. In this case, the CSNBRNGL callable service calls the processor to create the random number with the desired length, 8 bytes at a time.
- The server has the CP Assist for Cryptographic Functions with a feature level that supports pseudo-random number generation.

### Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 87. Random number generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 990 IBM @server zSeries 890	CP Assist for Cryptographic Functions  PCI X Cryptographic Coprocessor  Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions  Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions  Crypto Express2 Coprocessor  Crypto Express3 Coprocessor	

Table 87. Random number generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
z196 z114	CP Assist for Cryptographic Functions  Crypto Express3 Coprocessor	
IBM zEnterprise EC12	CP Assist for Cryptographic Functions  Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## Remote Key Export (CSNDRKX and CSNFRKX)

This callable service uses the trusted block to generate or export DES keys for local use and for distribution to an ATM or other remote device. RKX uses a special structure to hold encrypted symmetric keys in a way that binds them to the trusted block and allows sequences of RKX calls to be bound together as if they were an atomic operation.

The callable service name for AMODE(64) invocation is CSNFRKX.

### Format

```
CALL CSNDRKX(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    trusted_block_length,  
    trusted_block_identifier,  
    certificate_length,  
    certificate,  
    certificate_parms_length,  
    certificate_parms,  
    transport_key_length,  
    transport_key_identifier,  
    rule_id_length,  
    rule_id,  
    importer_key_length,  
    importer_key_identifier,  
    source_key_length,  
    source_key_identifier,  
    asym_encrypted_key_length,  
    asym_encrypted_key,  
    sym_encrypted_key_length,  
    sym_encrypted_key,  
    extra_data_length,  
    extra_data,  
    key_check_parameters_length,  
    key_check_parameters,  
    key_check_length,  
    key_check_value)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the specific results of processing. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. This number must be 0.

**rule\_array**

Direction: Input

Type: String

Specifies a string variable containing an array of keywords. Currently no *rule\_array* keywords are defined for this service, but you must still specify this parameter.

**trusted\_block\_length**

Direction: Input

Type: Integer

Specifies the number of bytes in the `trusted_block_identifier` parameter. The maximum length is 3500 bytes.

**trusted\_block\_identifier**

Direction: Input

Type: String

Specifies a trusted block label or trusted block token of an internal/complete trusted block constructed by the service, which is used to validate the public key certificate (certificate) and to define the rules for key generation and export.

**certificate\_length**

Direction: Input

Type: Integer

Specifies the number of bytes in the certificate parameter. The maximum is 5000 bytes.

If the `certificate_length` is zero and the trusted block's Asymmetric Encrypted Output Key Format indicates no asymmetric key output, this service will not attempt to use or validate the certificate in any way. Consequently, the output parameter `asym_encrypted_key_length` will contain zero and output parameter `asym_encrypted_key` will not be changed from its input content.

If the certificate length is zero and the trusted block's Asymmetric Encrypted Output Key Format indicates PKCS1.2 output format or RSAOAEP output format, this service will exit with an error.

If the `certificate_length` is non-zero and the trusted block's Asymmetric Encrypted Output Key Format indicates no asymmetric key output, this service will fail.

**certificate**

Direction: Input

Type: String

## Remote Key Export

Contains a public-key certificate. The certificate must contain the public key modulus and exponent in `binary_form`, as well as a digital signature. The signature in the certificate will be verified using the root public key that is in the trusted block supplied in `trusted_block_identifier` parameter.

### **certificate\_parms\_length**

Direction: Input

Type: Integer

Contains the number of bytes in the `certificate_parms` parameter. The length must be 36 bytes.

### **certificate\_parms**

Direction: Input

Type: String

Contains a structure provided by the caller used for identifying the location and length of values within the certificate in parameter `certificate`. For each of the values used by RKX, the structure contains offsets from the start of the certificate and length in bytes. It is the responsibility of the calling application program to provide these values. This method gives the greatest flexibility to support different certificate formats. The structure has this layout:

*Table 88. Structure of values used by RKX*

Offset (bytes)	Length (bytes)	Description
0	4	Offset of modulus
4	4	Length of modulus
8	4	Offset of public exponent
12	4	Length of public exponent
16	4	Offset of digital signature
20	4	Length of digital signature
24	1	Identifier for the hash algorithm used
25	1	Identifier for the digital hash formatting method <ul style="list-style-type: none"><li>• 01 - PKCS-1.0</li><li>• 02 - PKCS-1.1</li><li>• 03 - X9.31</li><li>• 04 - ISO-9796</li><li>• 05 - ZERO-PAD</li></ul>
26	2	Reserved - must be filled with 0x00 bytes
28	4	Offset of first byte of certificate data hashed to compute the digital signature
32	4	Length of the certificate data hashed to compute the digital signature

The modulus, exponent, and signature values are right-justified and padded on the left with binary zeros if necessary.

These values are defined for the hash algorithm identifier at offset 24 in the structure.

Table 89. Values defined for hash algorithm identifier at offset 24 in the structure for remote key export

Identifier	Algorithm
0X01	SHA-1
0X02	MD5
0X03	RIPEMD-160

### transport\_key\_length

Direction: Input

Type: Integer

Contains the number of bytes in the transport\_key\_identifier parameter.

### transport\_key\_identifier

Direction: Input

Type: String

Contains a label of an internal key token, or an RKX token for a Key Encrypting Key (KEK) that is used to encrypt a key exported by the RKX service. A transport key will not be used to encrypt a generated key.

For flag bit0=1 (export existing key) within Rule section and parameter rule\_id = Rule section ruleID, the transport\_key\_identifier encrypts the exported version of the key received in parameter source\_key\_identifier. The service can distinguish between the internal key token or RKX key token by virtue of the version number at offset 0x04 contained in the key token and the flag value at offset 0x00 as follows:

Table 90. Transport\_key\_identifier used by RKX

Flag Byte Offset 00	Version Number Offset 04	Description
0X01	0X00	Internal DES key token version 0
0X02	0X10	RKX Key token (Flag byte 0x02 indicates external key token)

### rule\_id\_length

Direction: Input

Type: Integer

Contains the number of bytes in the rule\_id parameter. The value must be 8.

### rule\_id

Direction: Input

Type: String

Specifies the rule in the trusted block that will be used to control key generation or export. The trusted block can contain multiple rules, each of which is identified by a rule ID value.

### importer\_key\_length

Direction: Input

Type: Integer

## Remote Key Export

Contains the number of bytes in the `importer_key_identifier` parameter. It must be zero if the Generate/Export flag in the rule section (section 0x12) of the Trusted Block is a zero, indicating a new key is to be generated.

### **importer\_key\_identifier**

Direction: Input

Type: String

Contains a key token or key label for the IMPORTER key-encrypting key that is used to decipher the key passed in parameter `source_key_identifier`. It is unused if either RKX is being used to generate a key, or if the `source_key_identifier` is an RKX key token.

### **source\_key\_length**

Direction: Input

Type: Integer

Contains the number of bytes in the `source_key_identifier` parameter. The parameter must be 0 if the trusted block Rule section `ruleID = rule_id` parameter and the flag `bit0 = 0` (Generate new key).

The parameter must be 64 if the trusted block Rule section has a flag `bit0 = 1` (Export existing key).

### **source\_key\_identifier**

Direction: Input

Type: String

Contains a label of a single or double length external or internal key token or an RKX key token for a key to be exported. It must be empty (`source_key_length=0`) if RKX is used to generate a new key. The service examines the key token to determine which form has been provided. This parameter is known as the *source\_key\_identifier* in other callable services.

*Table 91. Examination of key token for source\_key\_identifier*

Flag Byte Offset 00	Version Number Offset 04	Description
0X01	0X00	Internal DES key token version 0
0X02	0X00	External DES key token version 0
0X02	0X01	External DES key token version 1
0X02	0X10	RKX Key token (Flag byte 0x02 indicates external key token)

### **asym\_encrypted\_key\_length**

Direction: Input/Output

Type: Integer

The length of the `asym_encrypted_key` parameter. On input, it is the length of the storage to receive the output. On output, it is the length of the data returned in the `asym_encrypted_key` parameter. The maximum length is 512 bytes.

### **asym\_encrypted\_key**

Direction: Output

Type: String

The contents of this field is ignored on input. A string buffer RKX will use to return a generated or exported key that is encrypted under the public (asymmetric) key passed in parameter certificate. An error will be returned if the caller's buffer is too small to hold the value that would be returned.

### **sym\_encrypted\_key\_length**

Direction: Input/Output

Type: Integer

On input, the `sym_encrypted_key_length` parameter is an integer variable containing the number of bytes in the `sym_encrypted_key` field. On output, that value in `sym_encrypted_key_length` is replaced with the length of the key returned in `sym_encrypted_key` field.

### **sym\_encrypted\_key**

Direction: Output

Type: String

`Sym_encrypted_key` is the string buffer RKX uses to return a generated or exported key that is encrypted under the key-encrypting key passed in the `transport_key_identifier` parameter. The value returned will be 64 bytes. An error will be returned if the caller's buffer is smaller than 64 bytes, and so too small to hold the value that would be returned. The `sym_encrypted_key` may be an RKX key token or a key token depending upon the value of the Symmetric Encrypted Output Key Format value of the Rule section within the `trusted_block_identifier` parameter.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The *sym\_encrypted\_key* will be wrapped in the same manner as the *source\_key\_identifier*.

### **extra\_data\_length**

Direction: Input

Type: Integer

Contains the number of bytes of data in the `extra_data` parameter. It must be zero if the output format for the RSA-encrypted key in `asym_encrypted_key` is anything but RSAOEAP. The maximum size is 1024 bytes.

### **extra\_data**

Direction: Input

Type: String

Can be used in the OAEP key wrapping process. `Extra_data` is optional and is only applicable when the output format for the RSA-encrypted key returned in `asym_encrypted_key` is RSAOAEP.

**Note:** RSAOAEP format is specified in the rule in the trusted block.

### **key\_check\_parameters\_length**

Direction: Input

Type: Integer

Contains the number of bytes in the `key_check_parameters` parameter. Currently, none of the defined key check algorithms require any key check parameters, so this field must specify 0.

### **key\_check\_parameters**



## Remote Key Export

Direction: Input

Type: String

Contains parameters that are required to calculate a key check value parameter, which will be returned in `key_check_value`. Currently, none of the defined key check algorithms require any key check parameters, but you must still specify this parameter.

### **key\_check\_length**

Direction: Input/Output

Type: Integer

On input this parameter contains the number of bytes in the `key_check_value` parameter. On output, the value is replaced with the length of the key check value returned in the `key_check_value` parameter. The length depends on the key-check algorithm identifier. See Table 381 on page 881.

### **key\_check\_value**

Direction: Output

Type: String

Used by RKX to return a key check value that calculates on the generated or exported key. Values in the rule specified with `rule_id` can specify a key check algorithm that should be used to calculate this output value.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

## Access Control Points

The **Remote Key Export - Gen or export a non-CCA node Key** access control point controls the function of this service.

To use a NOCV IMPORTER key-encrypting key with the remote key export service, the **NOCV KEK usage for import-related functions** access control point must be enabled in addition to one or both of the access control points listed.

To use a NOCV EXPORTER key-encrypting key with the remote key export service, the **NOCV KEK usage for export-related functions** access control point must be enabled in addition to one or both of the access control points listed.

When the **RKX/TBC – Disallow triple-length MAC key** access control point is enable, this service will not import a triple-length MAC wrapped with a double-length KEK.

If the key-encrypting key identifier is a weaker key than the key being exported, then

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 92. Remote key export required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 990 IBM @server zSeries 890		This callable service is not supported.
IBM @server zSeries 900		This callable service is not supported.
IBM @server z9 EC IBM System z9 BC	Cryptographic Express 2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).  ENH-ONLY, USECONFIG, WRAP-ENC and WRAP-ECB not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).  ENH-ONLY, USECONFIG, WRAP-ENC and WRAP-ECB not supported.
	Crypto Express3 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor	
	Crypto Express4 Coprocessor	

## Restrict Key Attribute (CSNBRKA and CSNERKA)

Use the Restrict Key Attribute callable service to modify an attribute of an internal or external CCA symmetric key-token.

The callable service name for AMODE(64) is CSNERKA.

### Format

```
CALL CSNBRKA(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_identifier_length,  
    key_identifier,  
    key_encrypting_key_identifier_length,  
    key_encrypting_key_identifier,  
    opt_parameter1_length,  
    opt_parameter1,  
    opt_parameter2_length,  
    opt_parameter2 )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

#### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value must be between 1 and 10, inclusive.

#### **rule\_array**

Direction: Input

Type: String

The *rule\_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 93. Keywords for Restrict Key Attribute Control Information

Keyword	Meaning
<b>Token Type (Required)</b>	
AES	Specifies the key token is an AES key token.
DES	Specifies the key token is a DES key token.
HMAC	Specifies the key token is an HMAC key token.
<b>Attribute to Restrict (Optional)</b>	
NOEXPORT	Prohibits the key from being exported by any verb. The use of this keyword always causes each available export control attribute to be lowered. If no attribute to restrict keywords are used, this is the default.  Variable-length symmetric key-token: This keyword is equivalent to providing all of the keywords listed under Export Control for AES or HMAC (NOEX-SYM, NOEXUASY, NOEXASY, NOEX-AES, NOEX-DES, and NOEX-RSA). This is the default if no AES or HMAC attribute restriction keywords are used.  Internal DES key token: Use this keyword to set CV bit 17 = B'0' (NO-XPORT) and CV bit 27 = B'1' (NOT31XPT). This is the default if no DES attribute restriction keywords are used.
<b>For AES or HMAC keys (Optional, one or more keywords may be specified)</b>	
<b>Export control for AES and HMAC (one or more, optional)</b>	
NOEX-AES	Specifies to prohibit export using an AES key.
NOEX-DES	Specifies to prohibit export using a DES key.
NOEX-RAW	Specifies to prohibit export in RAW format.
NOEX-RSA	Specifies to prohibit export using an RSA key.
NOEX-SYM	Prohibits the key from being exported using a symmetric key.
NOEXAASY	Prohibits the key from being exported using an authenticated asymmetric key (for example, an RSA key in a trusted block token).
NOEXUASY	Prohibits the key from being exported using an unauthenticated asymmetric key.
<b>Key usage restriction for AES and HMAC (optional)</b>	
C-XLATE	Specifies that the CIPHER key can only be used for cipher text translate operations. This is only valid with AES CIPHER keys.
<b>For DES keys (Optional, one or two keywords)</b>	
<b>Export control for DES (one, optional)</b>	
CCAXPORT	For DES internal tokens, set bit 17 of the CV to 0 to prohibit any export of the key.

## Restrict Key Attribute

Table 93. Keywords for Restrict Key Attribute Control Information (continued)

Keyword	Meaning
NOT31XPT	For DES internal tokens, set bit 57 of the CV to 1 to prohibit TR-31 export of the key.
<b>Key restriction for DES (optional)</b>	
DOUBLE-O	For DES key tokens, change the control vector of a double-length key that has unique key halves (ignoring parity) to indicate that the key does not have replicated key halves. <b>Note:</b> A double-length key with replicated key halves has the effective strength of a single-length key. If the key token supplied in the <code>key_identifier</code> parameter has replicated key halves, this keyword will cause the service to fail.
<b>Input Transport Key (Optional)</b>	
IKEK-AES	Specifies the KEK is an AES transport key. This is the default for Token Types AES and HMAC, and is not allowed with Token Type DES.
IKEK-DES	Specifies the KEK is a DES transport key. This is the default for Token Type DES.
IKEK-PKA	Specifies the KEK is a PKA transport key. This is not allowed with Token Type DES.

### **key\_identifier\_length**

Direction: Input/Output

Type: Integer

The length of the `key_identifier` parameter in bytes. The maximum value is 900.

### **key\_identifier**

Direction: Input/Output

Type: String

The key for which the export control is to be updated. The parameter contains an internal or external token or the 64-byte CKDS label of an internal token. If a label is specified, the key token will be updated in the CKDS and not returned by this service.

If the key identifier supplied was an AES or DES token encrypted under the old master key, the token will be returned encrypted under the current master key.

### **key\_encrypting\_key\_identifier\_length**

Direction: Input

Type: Integer

The length of the `key_encrypting_key_identifier` parameter. When `key_identifier` is an internal token, the value must be zero.

- If `key_encrypting_key_identifier` is a label for either the CKDS (IKEK-AES or IKEK-DES rules) or PKDS (IKEK-PKA rule), the value must be 64.
- If `key_encrypting_key_identifier` is an AES KEK, the value must be between the actual length of the token and 725.
- If `key_encrypting_key_identifier` is a DES KEK, the value must be 64.
- If `key_encrypting_key_identifier` is an RSA KEK, the maximum length is 3500.

**key\_encrypting\_key\_identifier**

Direction: Input/Output

Type: String

When *key\_encrypting\_key\_identifier\_length* is non-zero, *key\_encrypting\_key\_identifier* contains an internal key token containing a key-encrypting key, or a key label.

If the key identifier supplied was an AES or DES token encrypted under the old master key, the token will be returned encrypted under the current master key.

**opt\_parameter1\_length**

Direction: Input

Type: Integer

The byte length of the *opt\_parameter1* parameter. The value must be zero.

**opt\_parameter1**

Direction: Input

Type: String

This parameter is ignored.

**opt\_parameter2\_length**

Direction: Input

Type: Integer

The byte length of the *opt\_parameter2* parameter. The value must be zero.

**opt\_parameter2**

Direction: Input

Type: String

This parameter is ignored.

## Access Control Points

The access control points in the ICSF role that control the function of this service are:

- Restrict Key Attribute - Export Control
- Restrict Key Attribute - Permit setting the TR-31 export bit

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 94. Restrict Key Attribute required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This service is not supported.

## Restrict Key Attribute

Table 94. Restrict Key Attribute required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 990 IBM @server zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	This service is not supported.
	Crypto Express3 Coprocessor	This service is not supported.
z196 z114	Crypto Express3 Coprocessor	HMAC key support requires the Nov. 2010 or later licensed internal code (LIC).  DES/AES key support requires the Sep. 2011 or later licensed internal code (LIC).  The C-XLATE and DOUBLE-O keywords are not supported.
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

---

## Secure Key Import (CSNBSKI and CSNESKI)

Use the secure key import callable service to encipher a single-length or double-length clear key under the system master key (DES or SYM-MK) or under an importer key-encrypting key. The clear key can then be imported as any of the possible key types. This service does not adjust key parity.

The callable service can execute only when ICSF is in special secure mode, which is described in “Special Secure Mode” on page 10.

To import double-length and triple-length DATA keys, or double-length MAC, MACVER, CIPHER, DECIPHER and ENCIPHER keys, use the Multiple Secure Key Import callable service. See “Multiple Secure Key Import (CSNBSKM and CSNESKM)” on page 222.

To import AES DATA keys, use the multiple secure key import service (“Multiple Secure Key Import (CSNBSKM and CSNESKM)” on page 222).

The callable service name for AMODE(64) invocation is CSNESKI.

## Format

```
CALL CSNBSKI(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    clear_key,
    key_type,
    key_form,
    importer_key_identifier,
    key_identifier )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **clear\_key**

Direction: Input

Type: String

The clear key to be enciphered. Specify a 16-byte string (clear key value). For single-length keys, the value must be left-justified and padded with zeros. For effective single-length keys, the value of the right half must equal the value of the left half. For double-length keys, specify the left and right key values.

**Note:** For key types that can be single or double-length, a single length encrypted key will be generated if a *clear\_key* value of zeros is supplied.

### **key\_type**



## Secure Key Import

Direction: Input

Type: Character string

The type of key you want to encipher under the master key or an importer key. Specify an 8-byte field that must contain a keyword from this list or the keyword TOKEN. If the key type is TOKEN, ICSF determines the key type from the CV in the *key\_identifier* parameter.

Key type values for the Secure Key Import callable service are: CIPHER, CIPHERXI, CIPHERXL, CIPHERXO, CVARDEC, CVARENC, CVARPINE, CVARXCVL, CVARXCVR, DATA, DATAXLAT, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IMP-PKA, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER. For information on the meaning of the key types, see Table 3 on page 23.

### key\_form

Direction: Input

Type: Character string

The key form you want to generate. Enter a 4-byte keyword specifying whether the key should be enciphered under the master key (OP) or the importer key-encrypting key (IM). The keyword must be left-justified and padded with blanks. Valid keyword values are OP for encryption under the master key or IM for encryption under the importer key-encrypting key. If you specify IM, you must specify an importer key-encrypting key in the *importer\_key\_identifier* parameter. For a *key\_type* of IMP-PKA, this service supports only the OP *key\_form*.

### importer\_key\_identifier

Direction: Input/Output

Type: String

The importer key-encrypting key under which you want to encrypt the clear key. Specify either a 64-byte string of the internal key format or a key label. If you specify IM for the *key\_form* parameter, the *importer\_key\_identifier* parameter is required.

### key\_identifier

Direction: Input/Output

Type: String

The generated encrypted key. The parameter is a 64-byte string. The callable service returns either an internal key token if you encrypted the clear key under the master key (*key\_form* was OP); or an external key token if you encrypted the clear key under the importer key-encrypting key (*key\_form* was IM).

**Note:** Starting with release HCR7780, if the *key\_type* parameter is not TOKEN, this parameter must be a 64-byte string of hex zero

If the imported *key\_type* is IMPORTER or EXPORTER and the *key\_form* is OP, the *key\_identifier* parameter changes direction to both input and output. If the application passes a valid internal key token for an IMPORTER or EXPORTER key in this parameter, the NOCV bit is propagated to the imported key token.

**Note:** Propagation of the NOCV bit is not performed if the service is processed on the PCI Cryptographic Coprocessor.

The secure key import service does not adjust key parity.

ICSF supports two methods of wrapping the key value in a symmetric key token: the original ECB wrapping and an enhanced CBC wrapping method which is ANSI X9.24 compliant. The output *key\_identifier* will use the default wrapping method unless a skeleton token is supplied as input. If a skeleton token is supplied as input, the wrapping method in the skeleton token will be used.

## Usage Notes — General

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

## Usage Notes — CCF Systems

To generate double-length MAC and MACVER keys in the importable form, the ANSI system keys must be installed in the CKDS.

This service will mark DATA, IMPORTER and EXPORTER key tokens with the system encryption algorithm.

- This service marks the imported DATA key token according to the system's default encryption algorithm, unless token copying overrides this.
- KEKs are marked SYS-ENC unless token copying overrides this.
- To override the default mark, supply a valid internal token of the same key type in the *key\_identifier* field. The service will copy the marks of the supplied token to the imported token.

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

*Table 95. Required access control points for Secure Key Import*

Key Form	Access control point
OP	Secure Key Import - DES, OP
IM	Secure Key Import - DES, IM

To use a NOCV key-encrypting key with the secure key import service, the **NOCV KEK usage for import-related functions** access control point must be enabled in addition to one or both of the access control points listed.

If the key-encrypting key identifier is a weaker key than the key being imported, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

## Secure Key Import

Table 96. Secure key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	Marking of data encryption algorithm bits and token copying are performed only if the service is processed on the Cryptographic Coprocessor Feature.
	PCI Cryptographic Coprocessor	ICSF routes the request to a PCI Cryptographic Coprocessor if: <ul style="list-style-type: none"><li>• The control vector of a supplied internal token cannot be processed on the Cryptographic Coprocessor Feature, or if the key type is not valid for the Cryptographic Coprocessor Feature.</li></ul>
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	<i>Key_type</i> DATAXLAT is not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	<i>Key_type</i> DATAXLAT is not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	<i>Key_type</i> DATAXLAT is not supported.
z196 z114	Crypto Express3 Coprocessor	<i>Key_type</i> DATAXLAT is not supported.
IBM zEnterprise EC12	Crypto Express3 Coprocessor	<i>Key_type</i> DATAXLAT is not supported.
	Crypto Express4 Coprocessor	

---

## Secure Key Import2 (CSNBSKI2 and CSNESKI2)

Use this service to encipher a variable-length symmetric key under the system master key or an AES IMPORTER KEK, depending on the Key Form rule provided. This service supports variable-length symmetric keys.

This service returns variable-length CCA key tokens and uses the AESKW wrapping method.

The callable service can execute only when ICSF is in special secure mode, which is described in “Special Secure Mode” on page 10.

The callable service name for AMODE(64) is CSNESKI2.

## Format

```
CALL CSNBSKI2(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    clear_key_bit_length,
    clear_key,
    key_name_length,
    key_name,
    user_associated_data_length,
    user_associated_data,
    key_encrypting_key_identifier_length,
    key_encrypting_key_identifier,
    target_key_identifier_length,
    target_key_identifier)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'FFFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value must be 3.

### **rule\_array**

## Secure Key Import2

Direction: Input

Type: String

The *rule\_array* contains keywords that provide control information to the callable service. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 97. Keywords for Secure Key Import2 Control Information

Keyword	Meaning
<i>Token algorithm (One Required)</i>	
HMAC	The target key identifier is to be an HMAC key.
AES	The target key identifier is to be an AES key.
<i>Key Form (One Required)</i>	
OP	Specifies the key should be enciphered under the master key.
IM	Specifies the key should be enciphered under the key-encrypting key.
<i>Key Type (One Required)</i>	
CIPHER	The key type of the output token will be CIPHER. Only valid for AES algorithm.
EXPORTER	The key type of the output token will be EXPORTER. Only valid for AES algorithm.
IMPORTER	The key type of the output token will be IMPORTER. Only valid for AES algorithm.
MAC	MAC generation key. Only valid for HMAC algorithm.
MACVER	MAC verify key. Only valid for HMAC algorithm.
TOKEN	The key type will be determined from the key token supplied in the <i>target_key_identifier</i> parameter. ICSF does not check for the length of the key but uses the <i>clear_key_bit_length</i> parameter to determine the length of the key.

### **clear\_key\_bit\_length**

Direction: Input

Type: Integer

The length of the value supplied in the *clear\_key* parameter in bits. Valid lengths are 80 to 2048 for HMAC keys, and 128, 192, or 256 for AES keys.

### **clear\_key**

Direction: Input

Type: String

The value of the key to be imported. The value should be left justified and padded on the right with zeros to a byte boundary if the *clear\_key\_bit\_length* is not a multiple of 8.

### **key\_name\_length**

Direction: Input

Type: Integer

The length of the *key\_name* parameter. Valid values are 0 and 64.

**key\_name**

Direction: Input

Type: String

A 64-byte key store label to be stored in the associated data structure of the token.

**user\_associated\_data\_length**

Direction: Input

Type: Integer

The length of the user-associated data. The valid values are 0 to 255 bytes.

**user\_associated\_data**

Direction: Input

Type: String

User-associated data to be stored in the associated data structure.

**key\_encrypting\_key\_identifier\_length**

Direction: Input

Type: Integer

The byte length of the *key\_encrypting\_key\_identifier* parameter. When Key Form is OP, the value must be zero. When Key Form is IM, the value must be between the actual length of the token and 725 when *key\_encrypting\_key\_identifier* is a token. The value must be 64 when *key\_encrypting\_key\_identifier* is a label.

**key\_encrypting\_key\_identifier**

Direction: Input/Output

Type: String

When the Key Form rule is OP, *key\_encrypting\_key\_identifier* is ignored. When the Key Form rule is IM, *key\_encrypting\_key\_identifier* contains an internal key token containing the AES importer key-encrypting key or a key label.

If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

**target\_key\_identifier\_length**

Direction: Input/Output

Type: Integer

On input, the byte length of the buffer for the *target\_key\_identifier* parameter. The buffer must be large enough to receive the target key token. The maximum value is 900 bytes.

On output, the parameter will hold the actual length of the target key token.

**target\_key\_identifier**

Direction: Input/Output

Type: String

The output key token. On input, this parameter is ignored except when the Key Type keyword is TOKEN. If you specify the TOKEN keyword, then this field contains a valid token of the key type you want to import. On output, when Key Form is OP, this will be an internal variable-length symmetric token.

## Secure Key Import2

When Key Form is IM, this will be an external variable-length symmetric token. See *rule\_array* for a list of valid key types.

### Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

Table 98. Required access control points for Secure Key Import2

Key Form	Access control point
OP	Secure Key Import2 – OP
IM	Secure Key Import2 – IM

When the **Symmetric Key Import2 - disallow weak import** access control point is enabled, a key token wrapped with a weaker key will not be imported. When the **Warn when weak wrap - Transport keys** access control point is enabled, the reason code will indicate when the wrapping key is weaker than the key being imported.

### Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 99. Secure Key Import2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This service is not supported.
IBM @server zSeries 990		This service is not supported.
IBM @server zSeries 890		
IBM System z9 EC		This service is not supported.
IBM System z9 BC		
IBM System z10 EC	Crypto Express2 Coprocessor	This service is not supported.
IBM System z10 BC	Crypto Express3 Coprocessor	This service is not supported.
z196	Crypto Express3 Coprocessor	HMAC key support requires the Nov. 2010 or later licensed internal code (LIC).
z114		AES key support requires the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12	Crypto Express3 Coprocessor	
	Crypto Express4 Coprocessor	

## Symmetric Key Export (CSNDSYX and CSNFSYX)

Use the symmetric key export callable service to transfer an application-supplied AES DATA (version X'04'), DES DATA, or variable-length symmetric key token key from encryption under a master key to encryption under an application-supplied RSA public key or AES EXPORTER key. The application-supplied key must be an ICSF AES, DES, or HMAC internal key token or the label of such a token in the CKDS. The Symmetric Key Import or Symmetric Key Import2 callable services can import the key encrypted under the RSA public key or AES EXPORTER at the receiving node.

The callable service name for AMODE(64) is CSNFSYX.

### Format

```
CALL CSNDSYX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    source_key_identifier_length,
    source_key_identifier,
    transporter_key_identifier_length,
    transporter_key_identifier,
    enciphered_key_length,
    enciphered_key)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String



## Symmetric Key Export

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. Value may be 1, 2, or 3.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. Table 100 lists the keywords. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

*Table 100. Keywords for Symmetric Key Export Control Information*

Keyword	Meaning
<i>Algorithm (One keyword, optional)</i>	
AES	The key being exported is an AES key. If <i>source_key_identifier</i> is a variable-length symmetric key token or label, only the PKOAEP2 and AESKW key formatting methods are supported.
DES	The key being exported is a DES key. This is the default.
HMAC	The key being exported is an HMAC key. Only the PKOAEP2 and AESKW key formatting methods are supported.
<i>Key Formatting method (One required)</i>	
AESKW	Specifies that the key is to be formatted using AESKW and placed in an external variable length CCA token. The <i>transport_key_identifier</i> must be an AES EXPORTER. This rule is not valid with the DES Algorithm keyword or with AES DATA (version X'04') keys.
PKCSOAEP	Specifies to format the key according to the method in RSA DSI PKCS #1V2 OAEP. The default hash method is SHA-1. Use the SHA-256 keyword for the SHA-256 hash method.
PKCS-1.2	Specifies to format the key according the method found in RSA DSI PKCS #1 block type 02 to recover the symmetric key.
PKOAEP2	Specifies to format the key according to the method found in RSA DSI PKCS #1 v2.1 RSAES-OAEP documentation. Not valid with DES algorithm or with AES DATA (version X'04') keys. A hash method is required.
ZERO-PAD	The clear key is right-justified in the field provided, and the field is padded to the left with zeros up to the size of the RSA encryption block (which is the modulus length).
<i>Hash Method (One, optional for PKCSOAEP, required for PKOAEP2. Not valid with any other Key Formatting method)</i>	
SHA-1	Specifies to use the SHA-1 hash method to calculate the OAEP message hash. This is the default for PKCSOAEP.

Table 100. Keywords for Symmetric Key Export Control Information (continued)

Keyword	Meaning
SHA-256	Specifies to use the SHA-256 hash method to calculate the OAEP message hash.
SHA-384	Specifies to use the SHA-384 hash method to calculate the OAEP message hash. Not valid with PKCSOAEP.
SHA-512	Specifies to use the SHA-512 hash method to calculate the OAEP message hash. Not valid with PKCSOAEP.

**source\_key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *source\_key\_identifier* parameter. The minimum size is 64 bytes. The maximum size is 725 bytes.

**source\_key\_identifier**

Direction: Input/Output

Type: String

The label or internal token of a secure AES DATA (version X'04'), DES DATA, or variable-length symmetric key token to encrypt under the supplied RSA public key or AES EXPORTER key. The key in the key identifier must match the algorithm in the *rule\_array*. DES is the default algorithm.

**transporter\_key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *transporter\_key\_identifier* parameter. The maximum size is 3500 bytes for an RSA key token or 725 for an AES EXPORTER key token. The length must be 64 if *transporter\_key\_identifier* is a label.

**transporter\_key\_identifier**

Direction: Input

Type: String

An RSA public key token, AES EXPORTER token, or label of the key to protect the exported symmetric key.

When the AESKW Key Formatting method is specified, this parameter must be an AES EXPORTER key token or label with the EXPORT bit on in the key-usage field. Otherwise, this parameter must be an RSA public key token or label.

**enciphered\_key\_length**

Direction: Input/Output

Type: Integer

The length of the *enciphered\_key* parameter. This is updated with the actual length of the *enciphered\_key* generated. The maximum size you can specify in this parameter is 900 bytes, although the actual key length may be further restricted by your hardware configuration (as shown in Table 103 on page 273).

**enciphered\_key**

Direction: Output

Type: String

## Symmetric Key Export

This field contains the exported key, protected by the RSA public or AES EXPORTER key specified in the *transporter\_key\_identifier* field.

### Restrictions

If you are running with the Cryptographic Coprocessor Feature, the enhanced system keys must be present in the CKDS.

### Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

If an RSA public key is specified as the *transporter\_key\_identifier*, the hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit.

When wrapping an AES key with an RSA public key, the RSA key used must have a modulus size greater than or equal to the total PKOAEP2 message bit length (key size + total overhead).

Table 101. Minimum RSA modulus strength required to contain a PKOAEP2 block when exporting an AES key

AES key size	Total message sizes (and therefore minimum RSA key size) when the Hash Method is:			
	SHA-1	SHA-256	SHA-384	SHA-512
128 bits	736 bits	928 bits	1184 bits	1440 bits
192 bits	800 bits	992 bits	1248 bits	1504 bits
256 bits	800 bits	1056 bits	1312 bits	1568 bits

### Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

Table 102. Required access control points for Symmetric Key Export

Key formatting method	Algorithm	Access control point
PKCSOAEP	AES	Symmetric Key Export - AES, PKCSOAEP, PKCS-1.2
	DES	Symmetric Key Export - DES, PKCS-1.2
PKCS-1.2	AES	Symmetric Key Export - AES, PKCSOAEP, PKCS-1.2
	DES	Symmetric Key Export - DES, PKCS-1.2
ZERO-PAD	AES	Symmetric Key Export - AES, ZERO-PAD
	DES	Symmetric Key Export - DES, ZERO-PAD

Table 102. Required access control points for Symmetric Key Export (continued)

Key formatting method	Algorithm	Access control point
PKOAEP2	HMAC	Symmetric Key Export - HMAC, PKOAEP2
	AES	Symmetric Key Export - AES, PKOAEP2
AESKW	AES or HMAC	Symmetric Key Export - AESKW

If the transport key identifier is a weaker key than the key being exported, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 103. Symmetric key export required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	<p>RSA keys with moduli greater than 1024-bit length are not supported.</p> <p>Encrypted AES keys are not supported.</p> <p>The DES, HMAC, and PKOAEP2 keywords are not supported.</p>
	PCI Cryptographic Coprocessor	<p>ICSF routes this service to a PCI Cryptographic Coprocessor if one is available on your server. This service will not be routed to a PCI Cryptographic Coprocessor if the modulus bit length of the RSA public key is less than 512 bits.</p> <p>Use of keyword PKCSOAEP requires the PCI Cryptographic Coprocessor and uses the SHA-1 hash method. The SHA-256 keyword is not supported for PKCSOAEP.</p> <p>RSA keys with moduli greater than 2048-bit length are not supported.</p> <p>Encrypted AES keys are not supported.</p> <p>The DES, AESKW, HMAC, and PKOAEP2 keywords are not supported.</p>

## Symmetric Key Export

Table 103. Symmetric key export required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	RSA keys with moduli greater than 2048-bit length are not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	Encrypted AES keys are not supported.  The AESKW, HMAC, and PKOAEP2 keywords are not supported.  The SHA-256 keyword is not supported for PKCSOAEP.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).  Encrypted AES key support requires the Nov. 2008 or later licensed internal code (LIC).  The AESKW, HMAC, and PKOAEP2 keywords are not supported.  The SHA-256 keyword is not supported for PKCSOAEP.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).  Encrypted AES key support requires the Nov. 2008 or later licensed internal code (LIC).  The AESKW, HMAC, and PKOAEP2 keywords are not supported.  The SHA-256 keyword is not supported for PKCSOAEP.
	Crypto Express3 Coprocessor	The AESKW, HMAC, and PKOAEP2 keywords are not supported.  The SHA-256 keyword is not supported for PKCSOAEP.
z196 z114	Crypto Express3 Coprocessor	HMAC key support requires the Nov. 2010 licensed internal code (LIC).  Variable-length AES Keys, the AESKW method, and PKCSOAEP with the SHA-256 hash method require the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12	Crypto Express3 Coprocessor	
	Crypto Express4 Coprocessor	

## Symmetric Key Generate (CSNDSYG and CSNFSYG)

Use the symmetric key generate callable service to generate an AES or DES DATA key and return the key in two forms: enciphered under the master key and encrypted under an RSA public key.

You can import the RSA public key encrypted form by using the symmetric key import service at the receiving node.

Also use the symmetric key generate callable service to generate any DES importer or exporter key-encrypting key encrypted under a RSA public key according to the PKA92 formatting structure. See “PKA92 Key Format and Encryption Process” on page 947 for more details about PKA92 formatting.

The callable service name for AMODE(64) invocation is CSNFSYG.

### Format

```
CALL CSNDSYG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_encrypting_key_identifier,
    RSA_public_key_identifier_length,
    RSA_public_key_identifier,
    local_enciphered_key_token_length,
    local_enciphered_key_token,
    RSA_enciphered_key_length,
    RSA_enciphered_key)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

## Symmetric Key Generate

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value must be 1, 2, 3, 4, 5, 6, or 7.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. Table 104 lists the keywords. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

*Table 104. Keywords for Symmetric Key Generate Control Information*

Keyword	Description	Algorithm
<i>Algorithm (one keyword, optional)</i>		
AES	The key being generated is a secure AES key.	AES
DES	The key being generated is a DES key. This is the default.	DES
<i>Key formatting method (one keyword required)</i>		
PKA92	Specifies the key-encrypting key is to be encrypted under a PKA96 RSA public key according to the PKA92 formatting structure.	DES
PKCSOAEP	Specifies using the method found in RSA DSI PKCS #1V2 OAEP. The default hash method is SHA-1. Use the SHA-256 keyword for the SHA-256 hash method.	AES or DES
PKCS-1.2	Specifies the method found in RSA DSI PKCS #1 block type 02.	AES or DES
ZERO-PAD	The clear key is right-justified in the field provided, and the field is padded to the left with zeros up to the size of the RSA encryption block (which is the modulus length).	AES or DES
<i>Key Length (optional - for use with PKA92)</i>		

Table 104. Keywords for Symmetric Key Generate Control Information (continued)

Keyword	Description	Algorithm
SINGLE-R	For key-encrypting keys, this specifies that the left half and right half of the generated key will have identical values. This makes the key operate identically to a single-length key with the same value. Without this keyword, the left and right halves of the key-encrypting key will each be generated randomly and independently.	DES
<b>Key Length (optional - for use with PKCSOAEP, PKCS-1.2, or ZERO-PAD)</b>		
SINGLE, KEYLN8	Specifies that the generated key should be 8 bytes in length.	DES
DOUBLE	Specifies that the generated key should be 16 bytes in length.	DES
KEYLN16	Specifies that the generated key should be 16 bytes in length.	AES or DES
KEYLN24	Specifies that the generated key should be 24 bytes in length.	AES or DES
KEYLN32	Specifies that the generated key should be 32 bytes in length.	AES
<b>Encipherment method for the local enciphered copy of the key (optional - for use with PKCSOAEP, PKCS-1.2, or ZERO-PAD)</b>		
OP	Enciphers the key with the master key. The DES master key is used with DES keys and the AES master key is used with AES keys.	AES or DES
EX	Enciphers the key with the EXPORTER key that is provided through the <i>key_encrypting_key_identifier</i> parameter.	DES
IM	Enciphers the key with the IMPORTER key-encrypting key specified with the <i>key_encrypting_key_identifier</i> parameter.	DES
<b>Key Wrapping Method (optional)</b>		
USECONFIG	Specifies that the system default configuration should be used to determine the wrapping method. This is the default keyword.  The system default key wrapping method can be specified using the DEFAULTWRAP parameter in the installation options data set. See the <i>z/OS Cryptographic Services ICSF System Programmer's Guide</i> .	AES and DES
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.	DES



## Symmetric Key Generate

Table 104. Keywords for Symmetric Key Generate Control Information (continued)

Keyword	Description	Algorithm
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens.	AES or DES
<b>Translation Control (optional)</b>		
ENH-ONLY	Restrict rewrapping of the <i>target_key_identifier</i> token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method.	DES
<b>Hash Method (optional - only valid with PKCSOAEP)</b>		
SHA-1	Specifies to use the SHA-1 hash method to calculate the OAEP message hash. This is the default.	AES or DES
SHA-256	Specifies to use the SHA-256 hash method to calculate the OAEP message hash.	AES or DES

### **key\_encrypting\_key\_identifier**

Direction: Input/Output

Type: String

The label or internal token of a key-encrypting key. If the *rule\_array* specifies IM, this DES key must be an IMPORTER. If the *rule\_array* specifies EX, this DES key must be an EXPORTER. Otherwise, the parameter is ignored.

### **RSA\_public\_key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *RSA\_public\_key\_identifier* parameter. If the *RSA\_public\_key\_identifier* parameter is a label, this parameter specifies the length of the label. The maximum size is 3500 bytes.

### **RSA\_public\_key\_identifier**

Direction: Input

Type: String

The token, or label, of the RSA public key to be used for protecting the generated symmetric key.

### **local\_enciphered\_key\_token\_length (was DES\_enciphered\_key\_token\_length)**

Direction: Input/Output

Type: Integer

The length in bytes of the *local\_enciphered\_key\_token*. This field is updated with the actual length of the token that is generated. The minimum length is 64-bytes and the maximum length is 128 bytes.

### **local\_enciphered\_key\_token (was DES\_enciphered\_key\_token)**

Direction: Input/Output

Type: String

This parameter contains the generated DATA key in the form of an internal or external token, depending on *rule\_array* specification. If you specify PKA92, on input specify an internal (operational) key token of an Importer or Exporter Key.

#### **RSA\_enciphered\_key\_length**

Direction: Input/Output

Type: Integer

The length of the *RSA\_enciphered\_key* parameter. This service updates this field with the actual length of the *RSA\_enciphered\_key* it generates. The maximum size is 512 bytes.

#### **RSA\_enciphered\_key**

Direction: Input/Output

Type: String

This field contains the RSA enciphered key, which is protected by the public key specified in the *RSA\_public\_key\_identifier* field.

## **Restrictions**

If the service is executed on the Cryptographic Coprocessor Feature, and you specify IM in the *rule\_array*, you must enable Special Secure Mode.

Use of PKA92 or PKCSOAEP requires a PCICC, PCIXCC, or CCA Crypto Express coprocessor.

## **Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

If the service is executed on the Cryptographic Coprocessor Feature, the generated internal DATA key token is marked according to the system default algorithm.

The hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit.

Specification of PKA92 with an input NOCV key-encrypting key token is not supported.

Use the PKA92 key-formatting method to generate a key-encrypting key. The service enciphers one key copy using the key encipherment technique employed in the IBM Transaction Security System (TSS) 4753, 4755, and AS/400 cryptographic product PKA92 implementations (see “PKA92 Key Format and Encryption Process” on page 947). The control vector for the RSA-enciphered copy of the key is taken from an internal (operational) DES key token that must be present on input in the *RSA\_enciphered\_key* variable. Only key-encrypting keys that conform to the rules for an OPEX case under the key generate service are permitted. The control vector for the local key is taken from a DES key token that must be present on input in the *local\_enciphered\_key\_token* variable. The control vector for one key copy must be from the EXPORTER class while the control vector for the other key copy must be from the IMPORTER class.

### Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

Table 105. Required access control points for Symmetric Key Generate

Key algorithm	Key formatting rule	Access control point
DES	PKCS-1.2	Symmetric Key Generate - DES, PKCS-1.2
DES	ZERO-PAD	Symmetric Key Generate - DES, ZERO-PAD
DES	PKA92	Symmetric Key Generate - DES, PKA92
AES	PKCSOAEP, PKCS-1.2	Symmetric Key Generate - AES, PKCSOAEP, PKCS-1.2
AES	ZERO-PAD	Symmetric Key Generate - AES, ZERO-PAD

When the WRAP-ECB or WRAP-ENH keywords are specified and the default key-wrapping method setting does not match the keyword, the **Symmetric Key Generate - Allow wrapping override keywords** access control point must be enabled.

If the RSA key identifier is a weaker key than the key being generated, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

### Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 106. Symmetric key generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	<p>ICSF routes this service to a PCI Cryptographic Coprocessor if one is available on your server. This service will not be routed to a PCI Cryptographic Coprocessor if the modulus bit length of the RSA public key is less than 512 bits.</p> <p>RSA keys with moduli greater than 1024-bit length are not supported.</p> <p>Secure AES keys are not supported.</p> <p>DES, ENH-ONLY, USECONFIG, WRAP-ENH, WRAP-ECB, and SHA-256 keywords not supported.</p>
	PCI Cryptographic Coprocessor	<p>Use of keyword PKA92 or PKCSOAEP requires the PCI Cryptographic Coprocessor. PKCSOAEP uses the SHA-1 hash method.</p> <p>RSA keys with moduli greater than 2048-bit length are not supported.</p> <p>Secure AES keys are not supported.</p> <p>DES, ENH-ONLY, USECONFIG, WRAP-ENH, WRAP-ECB, SHA-1, and SHA-256 keywords not supported.</p> <p>PKCSOAEP with the SHA-256 hash method is not supported.</p>
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	<p>The generated internal DATA key will not have any system encryption algorithm markings.</p>
IBM @server zSeries 890	Crypto Express2 Coprocessor	<p>RSA keys with moduli greater than 2048-bit length are not supported.</p> <p>Secure AES keys are not supported.</p> <p>ENH-ONLY, USECONFIG, WRAP-ENH, WRAP-ECB, and SHA-256 keywords not supported.</p> <p>PKCSOAEP with the SHA-256 hash method is not supported.</p>

## Symmetric Key Generate

Table 106. Symmetric key generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM Systems z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	<p>The generated internal DATA key will not have any system encryption algorithm markings.</p> <p>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</p> <p>Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).</p> <p>ENH-ONLY, USECONFIG, WRAP-ENH, WRAP-ECB, and SHA-256 not supported.</p> <p>PKCSOAEP with the SHA-256 hash method is not supported.</p>
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	<p>The generated internal DATA key will not have any system encryption algorithm markings.</p> <p>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</p> <p>Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).</p> <p>ENH-ONLY, USECONFIG, WRAP-ENH, WRAP-ECB, and SHA-256 not supported.</p> <p>PKCSOAEP with the SHA-256 hash method is not supported.</p>
	Crypto Express3 Coprocessor	<p>The generated internal DATA key will not have any system encryption algorithm markings.</p> <p>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</p> <p>Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).</p> <p>The SHA-256 keyword is not supported.</p> <p>PKCSOAEP with the SHA-256 hash method is not supported.</p>
z196 z114	Crypto Express3 Coprocessor	<p>The generated internal DATA key will not have any system encryption algorithm markings.</p> <p>PKCSOAEP with the SHA-256 hash method requires the Sep. 2011 or later licensed internal code (LIC).</p>

Table 106. Symmetric key generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	The generated internal DATA key will not have any system encryption algorithm markings.

## Symmetric Key Import (CSNDSYI and CSNFSYI)

Use the symmetric key import callable service to import a symmetric AES DATA or DES DATA key enciphered under an RSA public key. It returns the key in operational form, enciphered under the master key.

This service also supports import of a PKA92-formatted DES key-encrypting key under a PKA96 RSA public key.

The callable service name for AMODE(64) is CSNFSYI.

### Format

```
CALL CSNDSYI(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    RSA_enciphered_key_length,
    RSA_enciphered_key,
    RSA_private_key_identifier_length,
    RSA_private_key_identifier,
    target_key_identifier_length,
    target_key_identifier)
```

### Parameters

#### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### reason\_code

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

#### exit\_data\_length

## Symmetric Key Import

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value may be 1, 2, 3, 4, or 5.

### **rule\_array**

Direction: Input

Type: String

The keywords that provide control information to the callable service. Table 107 provides a list. The recovery method is the method to use to recover the symmetric key. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

*Table 107. Keywords for Symmetric Key Import Control Information*

Keyword	Meaning
<i>Algorithm (one keyword, optional)</i>	
AES	The key being imported is an AES key.
DES	The key being imported is a DES key. This is the default.
<i>Recovery Method (required)</i>	
PKA92	Supported by the DES algorithm. Specifies the key-encrypting key is encrypted under a PKA96 RSA public key according to the PKA92 formatting structure.
PKCSOAEP	Specifies to use the method found in RSA DSI PKCS #1V2 OAEP. Supported by the DES and AES algorithms. The default hash method is SHA-1. Use the SHA-256 keyword for the SHA-256 hash method.
PKCS-1.2	Specifies to use the method found in RSA DSI PKCS #1 block type 02. Supported by the DES and AES algorithms.
ZERO-PAD	The clear key is right-justified in the field provided, and the field is padded to the left with zeros up to the size of the RSA encryption block (which is the modulus length). Supported by the DES and AES algorithms.
<i>Key Wrapping Method (optional)</i>	

Table 107. Keywords for Symmetric Key Import Control Information (continued)

Keyword	Meaning
USECONFIG	Specifies that the system default configuration should be used to determine the wrapping method. This is the default keyword.  The system default key wrapping method can be specified using the DEFAULTWRAP parameter in the installation options data set. See the <i>z/OS Cryptographic Services ICSF System Programmer's Guide</i> .
WRAP-ENH	Use enhanced key wrapping method, which is compliant with the ANSI X9.24 standard.
WRAP-ECB	Use original key wrapping method, which uses ECB wrapping for DES key tokens and CBC wrapping for AES key tokens.
<b>Translation Control (optional)</b>	
ENH-ONLY	Restrict rewrapping of the <i>target_key_identifier</i> token. Once the token has been wrapped with the enhanced method, it cannot be rewrapped using the original method.
<b>Hash Method (optional - only valid with PKCSOAEP)</b>	
SHA-1	Specifies to use the SHA-1 hash method to calculate the OAEP message hash. This is the default.
SHA-256	Specifies to use the SHA-256 hash method to calculate the OAEP message hash.

**RSA\_enciphered\_key\_length**

Direction: Input

Type: integer

The length of the *RSA\_enciphered\_key* parameter. The maximum size is 512 bytes.

**RSA\_enciphered\_key**

Direction: Input

Type: String

The key to import, protected under an RSA public key. The encrypted key is in the low-order bits (right-justified) of a string whose length is the minimum number of bytes that can contain the encrypted key. This string is left-justified within the *RSA\_enciphered\_key* parameter.

**RSA\_private\_key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *RSA\_private\_key\_identifier* parameter. When the *RSA\_private\_key\_identifier* parameter is a key label, this field specifies the length of the label. The maximum size is 3500 bytes.

**RSA\_private\_key\_identifier**

Direction: Input

Type: String



## Symmetric Key Import

An internal RSA private key token or label whose corresponding public key protects the symmetric key.

### **target\_key\_identifier\_length**

Direction: Input/Output

Type: Integer

The length of the *target\_key\_identifier* parameter. This field is updated with the actual length of the *target\_key\_identifier* that is generated. The size must be 64 bytes.

### **target\_key\_identifier**

Direction: Output

Type: String

This field contains the internal token of the imported symmetric key. Except for PKA92 processing, this service produces a DATA key token with a key of the same length as that contained in the imported token.

## Restrictions

The exponent of the RSA public key must be odd.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

If the service is executed on the Cryptographic Coprocessor Feature, the generated internal DATA key token is marked according to the default system encryption algorithm unless token copying overrides this. Token copying is accomplished by supplying a valid DATA token with the desired algorithm marks in the *target\_key\_identifier* field.

The hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit. The service will fail with return code 12 and reason code 11020.

Specification of PKA92 with an input NOCV key-encrypting key token is not supported.

During initialization of a PCICC, PCIXCC, or CCA Crypto Express coprocessor, an Environment Identification, or EID, of zero will be set in the coprocessor. This will be interpreted by the PKA Symmetric Key Import service to mean that environment identification checking is to be bypassed. Thus it is possible on a OS/390 system for a key-encrypting key RSA-enciphered at a node (EID) to be imported at the same node.

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

*Table 108. Required access control points for Symmetric Key Import*

Key algorithm	Key formatting rule	Access control point
DES	PKCS-1.2	Symmetric Key Import - DES, PKCS-1.2

Table 108. Required access control points for Symmetric Key Import (continued)

Key algorithm	Key formatting rule	Access control point
DES	PKA92 KEK	Symmetric Key Import - DES, PKA92 KEK
DES	ZERO-PAD	Symmetric Key Import - DES, ZERO-PAD
AES	PKCSOAEP, PKCS-1.2	Symmetric Key Import - AES, PKCSOAEP, PKCS-1.2
AES	ZERO-PAD	Symmetric Key Import - AES, ZERO-PAD

When the WRAP-ECB or WRAP-ENH keywords are specified and the default key-wrapping method setting does not match the keyword, the **Symmetric Key Import - Allow wrapping override keywords** access control point must be enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

## Symmetric Key Import

Table 109. Symmetric key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	<p>Request routed to the CCF when -</p> <ul style="list-style-type: none"> <li>• The <i>RSA_private_key_identifier</i> is a modulus-exponent form private key with a private section ID of X'02'</li> <li>• The key modulus bit length is less than 512</li> </ul> <p>RSA keys with moduli greater than 1024-bit length are not supported.</p> <p>Encrypted AES keys are not supported.</p> <p>DES, ENH-ONLY, USECONFIG, WRAP-ENH and WRAP-ECB keywords not supported.</p>
	PCI Cryptographic Coprocessor	<p>Request routed to PCICC when</p> <ul style="list-style-type: none"> <li>• The <i>RSA_private_key_identifier</i> is a modulus-exponent form private key with a private section ID of X'06'</li> <li>• The <i>RSA_private_key_identifier</i> is a CRT form private key with a private section ID of X'08'</li> <li>• The <i>RSA_private_key_identifier</i> is a retained key</li> <li>• PKA92 recovery method specified</li> <li>• PKCSOAEP recovery method (which uses the SHA-1 hash method) specified</li> </ul> <p>RSA keys with moduli greater than 2048-bit length are not supported.</p> <p>Encrypted AES keys are not supported.</p> <p>DES, ENH-ONLY, USECONFIG, WRAP-ENH, WRAP-ECB, and SHA-256 keywords not supported.</p> <p>PKCSOAEP with the SHA-256 hash method is not supported.</p>

Table 109. Symmetric key import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 990  IBM @server zSeries 890	PCI X Cryptographic Coprocessor  Crypto Express2 Coprocessor	<p>The imported internal DATA key will not have any system encryption markings. Old RSA private keys encrypted under the CCF KMMK is not usable if the KMMK is not the same as the PCIXCC/CEX2C ASYM-MK.</p> <p>RSA keys with moduli greater than 2048-bit length are not supported.</p> <p>Encrypted AES keys are not supported.</p> <p>ENH-ONLY, USECONFIG, WRAP-ENH, WRAP-ECB, and SHA-256 keywords not supported.</p> <p>PKCSOAEP with the SHA-256 hash method is not supported.</p>
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	<p>The imported internal DATA key will not have any system encryption markings. Old RSA private keys encrypted under the CCF KMMK is not usable if the KMMK is not the same as the CEX2C ASYM-MK.</p> <p>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</p> <p>Encrypted AES keys are not supported.</p> <p>ENH-ONLY, USECONFIG, WRAP-ENH, WRAP-ECB, and SHA-256 keywords not supported.</p> <p>PKCSOAEP with the SHA-256 hash method is not supported.</p>

## Symmetric Key Import

Table 109. Symmetric key import required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	<p>The imported internal DATA key will not have any system encryption markings. Old RSA private keys encrypted under the CCF KMMK is not usable if the KMMK is not the same as the CCA Crypto Express coprocessor ASYM-MK.</p> <p>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</p> <p>Encrypted AES key support requires the Nov. 2008 or later licensed internal code (LIC).</p> <p>ENH-ONLY, USECONFIG, WRAP-ENH, WRAP-ECB, and SHA-256 keywords not supported.</p> <p>PKCSOAEP with the SHA-256 hash method is not supported.</p>
	Crypto Express3 Coprocessor	<p>The imported internal DATA key will not have any system encryption markings. Old RSA private keys encrypted under the CCF KMMK is not usable if the KMMK is not the same as the CCA Crypto Express coprocessor ASYM-MK.</p> <p>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</p> <p>Encrypted AES key support requires the Nov. 2008 or later licensed internal code (LIC).</p> <p>The SHA-256 keyword is not supported.</p> <p>PKCSOAEP with the SHA-256 hash method is not supported.</p>
z196 z114	Crypto Express3 Coprocessor	<p>The imported internal DATA key will not have any system encryption markings. Old RSA private keys encrypted under the CCF KMMK is not usable if the KMMK is not the same as the CCA Crypto Express coprocessor ASYM-MK.</p> <p>PKCSOAEP with the SHA-256 hash method requires the Sep. 2011 or later licensed internal code (LIC).</p>
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	<p>The imported internal DATA key will not have any system encryption markings. Old RSA private keys encrypted under the CCF KMMK is not usable if the KMMK is not the same as the CCA Crypto Express coprocessor ASYM-MK.</p>

## Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2)

Use the Symmetric Key Import2 callable service to import an HMAC or AES key enciphered under an RSA public key or AES EXPORTER key. It returns the key in operational form, enciphered under the master key.

This service returns a variable-length CCA key token and uses the AESKW wrapping method.

The callable service name for AMODE(64) is CSNFSYI2.

### Format

```
CALL CSNDSYI2(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    enciphered_key_length,
    enciphered_key,
    transport_key_identifier_length,
    transport_key_identifier,
    key_name_length,
    key_name,
    target_key_identifier_length,
    target_key_identifier)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

## Symmetric Key Import2

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value must be 2.

### **rule\_array**

Direction: Input

Type: String

The keywords that provide control information to the callable service. The following table provides a list. The recovery method is the method to use to recover the symmetric key. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

*Table 110. Keywords for Symmetric Key Import2 Control Information*

Keyword	Meaning
<i>Algorithm (One required)</i>	
AES	The key being imported is an AES key.
HMAC	The key being imported is an HMAC key.
<i>Recovery Method (Required)</i>	
AESKW	Specifies the enciphered key has been wrapped with the AESKW formatting method.
PKOAEP2	Specifies to use the method found in RSA DSI PKCS #1V2 OAEP.

### **enciphered\_key\_length**

Direction: Input

Type: integer

The length of the *enciphered\_key* parameter. The maximum size is 900 bytes.

### **enciphered\_key**

Direction: Input

Type: String

The key to import, protected under either an RSA public key or an AES KEK. If the Recovery Method is PKOAEP2, the encrypted key is in the low-order bits (right-justified) of a string whose length is the minimum number of bytes that can contain the encrypted key. If the Recovery Method is AESKW, the encrypted key is an AES key or HMAC key in the external variable length key token.

### **transport\_key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *transport\_key\_identifier* parameter. When the *transport\_key\_identifier* parameter is a key label, this field must be 64. The maximum size is 3500 bytes for an RSA private key or 725 bytes for an AES IMPORTER KEK.

#### **transport\_key\_identifier**

Direction: Input

Type: String

An internal RSA private key token, internal AES IMPORTER KEK, or the 64-byte label of a key token whose corresponding key protects the symmetric key.

When the AESKW Key Formatting method is specified, this parameter must be an AES IMPORTER with the IMPORT bit on in the key-usage field. Otherwise, this parameter must be an RSA private key.

#### **key\_name\_length**

Direction: Input

Type: Integer

The length of the *key\_name* parameter for *target\_key\_identifier*. Valid values are 0 and 64.

#### **key\_name**

Direction: Input

Type: String

A 64-byte key store label to be stored in the associated data structure of *target\_key\_identifier*.

#### **target\_key\_identifier\_length**

Direction: Input/Output

Type: Integer

On input, the byte length of the buffer for the *target\_key\_identifier* parameter. The buffer must be large enough to receive the target key token. The maximum value is 725 bytes.

On output, the parameter will hold the actual length of the target key token.

#### **target\_key\_identifier**

Direction: Output

Type: String

This parameter contains the internal token of the imported symmetric key.

## **Restrictions**

The exponent of the RSA public key must be odd.

## **Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This is the message layout used to encode the key material exported with the new PKOAEP2 formatting method.



## Symmetric Key Import2

Table 111. PKCS#1 OAEP encoded message layout (PKOAEP2)

Field	Size	Value
Hash field	32 Bytes	SHA-256 hash of associated data section in the source key identifier
Key Bit Length	2 Bytes	variable
Key Material	Byte length of the key material (rounded up to the nearest byte)	variable

### Hash field

The associated data for the HMAC variable length token is hashed using SHA-256. Specifically referring to `varToken.h`, this is the "VarAssocData\_t AD" section of the `VarKeyTkn_t` structure, for the full length indicated in the 'SectLn' field of the `VarAssocData_t`.

### Key Bit Length

A 2 Byte key bit length field.

### Key Material

The key material is padded to the nearest byte with '0' bits.

## Access Control Points

This table lists the access control points in the ICSF role that control the function for this service.

Table 112. Symmetric Key Import2 Access Control Points

Key formatting method	Algorithm	Access control point
PKOAEP2	HMAC, AES	Symmetric Key Import2 - HMAC/AES, PKOAEP2
AESKW	HMAC, AES	Symmetric Key Import2 - HMAC/AES, AESKW

When the **Symmetric Key Import2 - disallow weak import** access control point is enabled, a key token wrapped with a weaker key will not be imported. When the **Warn when weak wrap - Transport keys** access control point is enabled, the reason code will indicate when the wrapping key is weaker than the key being imported.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 113. Symmetric key import2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This service is not supported.

Table 113. Symmetric key import2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 990 IBM @server zSeries 890		This service is not supported.
IBM System z9 EC IBM System z9 BC		This service is not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	This service is not supported.
	Crypto Express3 Coprocessor	This service is not supported.
z196 z114	Crypto Express3 Coprocessor	HMAC key support requires the Nov. 2010 or later licensed internal code (LIC).  AES key support and the AESKW wrapping method require the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## Transform CDMF Key (CSNBTK and CSNETCK)

This callable service is only supported on an IBM @server zSeries 900.

Use the transform CDMF key callable service to change a CDMF DATA key in an internal or external token to a transformed shortened DES key. You can also use the key label of a CKDS record as input.

The Cryptographic Coprocessor Feature on IBM @server zSeries 900, S/390 Enterprise Servers and S/390 Multiprise is configured as either CDMF or DES-CDMF. This callable service ignores the input internal DATA token markings, and it marks the output internal token for use in the DES.

If the input DATA key is in an external token, the operational KEK must be marked as DES or SYS-ENC. The service fails for an external DATA key encrypted under a KEK that is marked as CDMF.

The callable service name for AMODE(64) invocation is CSNETCK.

### Format

```
CALL CSNBTK(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    source_key_identifier,  
    kek_key_identifier,  
    target_key_identifier )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

#### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. This number must be 0.

#### **rule\_array**

Direction: Input

Type: String

Currently no *rule\_array* keywords are defined for this service, but you still must specify this parameter.

**source\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string of the internal token, external token or key label that contains the DATA key to transform. Token markings on this key token are ignored.

**kek\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string of the internal token or a key label of a key encrypting key under which the *source\_key\_identifier* is encrypted.

**Note:** If you supply a label for this parameter, the label must be unique in the CKDS.

**target\_key\_identifier**

Direction: Output

Type: String

A 64-byte string where the internal token or external token of the transformed shortened DES key is returned. The internal token is marked as DES.

## Restrictions

This service is available on S/390 Enterprise Servers and S/390 Multiprise with Cryptographic Coprocessor Features. These systems may be configured as either CDMF or DES-CDMF.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This service transforms a CDMF DATA key to a transformed shortened DES DATA key to allow interoperability to a DES-only capable system. The algorithm is described in Transform CDMF Key Algorithm.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 114. Transform CDMF key required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990		This callable service is not supported.
IBM @server zSeries 890		
IBM System z9 EC		This callable service is not supported.
IBM System z9 BC		

## Transform CDMF Key

Table 114. Transform CDMF key required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC		This callable service is not supported.
z196 z114		This callable service is not supported.
IBM zEnterprise EC12		This callable service is not supported.

---

## Trusted Block Create (CSNDTBC and CSNFTBC)

This callable service is used to create a trusted block in a two step process. The block will be in external form, encrypted under an IMP-PKA transport key. This means that the MAC key contained within the trusted block will be encrypted under the IMP-PKA key.

The callable service name for AMODE(64) invocation is CSNFTBC.

### Format

```
CALL CSNDTBC(  
    return_code,  
    reason_code  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    input_block_length  
    input_block_identifier  
    transport_key_identifier,  
    trusted_block_length,  
    trusted_block_identifier )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, ICSF and TSS Return and Reason Codes lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the specific results of processing. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

**exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. This number must be 1.

**rule\_array**

Direction: Input

Type: String

Specifies a string variable containing an array of keywords. The keywords are 8 bytes long and must be left-justified and right padded with blanks

This table lists the *rule\_array* keywords for this callable service.

*Table 115. Rule\_array keywords for Trusted Block Create (CSNDTBC)*

Keyword	Meaning
<i>Operational Keywords - One Required</i>	
INACTIVE	Create the trusted block, but in inactive form. The MAC key is randomly generated, encrypted with the transport key, and inserted into the block. The ACTIVE flag is set to False (0), and the MAC is calculated over the block and inserted in the appropriate field. The resulting block is fully formed and protected, but it is not usable in any other CCA services. Use of the INACTIVE keyword is authorized by the 0x030F access control point.
ACTIVATE	This makes the trusted block usable in CCA services. Use of the ACTIVATE keyword is authorized by the 0x0310 access control point.

**input\_block\_length**

Direction: Input/Output

Type: String

Specifies the number of bytes of data in the *input\_block\_identifier* parameter. The maximum length is 3500 bytes.

**input\_block\_identifier**

Direction: Input

Type: String

Specifies a trusted block label or complete trusted block token, which will be updated by the service and returned in *trusted\_block\_identifier*. The length is indicated by *input\_block\_length*. Its content depends on the rule array keywords supplied to the service.

## Trusted Block Create

When rule\_array is INACTIVE the block is complete but typically does not have MAC protection. If MAC protection is present due to recycling an existing trusted block, then the MAC key and MAC value will be overlaid by the new MAC key and MAC value. The input\_block\_identifier includes all fields of the trusted block token, but the MAC key and MAC will be filled in by the service. The Active flag will be set to False (0) in the block returned in trusted\_block\_identifier.

When the rule\_array is ACTIVATE the block is complete, including the MAC protection which is validated during execution of the service. The Active flag must be False (0) on input. On output, the block will be returned in trusted\_block\_identifier provided the identifier is a token, with the Active flag changed to True (1), and the MAC value recalculated using the same MAC key. If the trusted\_block\_identifier is a label, the block will be written to the PKDS.

### transport\_key\_identifier

Direction: Input

Type: String

Specifies a key label or key token for an IMP-PKA key that is used to protect the trusted block.

### trusted\_block\_length

Direction: Input/Output

Type: Integer

Specifies the number of bytes of data in trusted\_block\_identifier parameter. The maximum length is 3500 bytes.

### trusted\_block\_identifier

Direction: Output

Type: String

Specifies a trusted block label or trusted block token for the trusted block constructed by the service. On input, the trusted\_block\_length contains the size of this buffer. On output, the trusted\_block\_length is updated with the actual byte length of the trusted block written to the buffer if the trusted\_block\_identifier is a token. The trusted block consists of the data supplied in input\_block\_identifier, but with the MAC protection and Active flag updated according to the rule array keyword that is provided. See Table 115 on page 299 for details on the actions. If the trusted\_block\_identifier is a label identifying a key record in key storage, the returned trusted block token will be written to the PKDS.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

Table 116. Required access control points for Trusted Block Create

Rule array keyword	Access control point
INACTIVE	Trusted Block Create - Create Block in Inactive form
ACTIVATE	Trusted Block Create - Activate an Inactive Block

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 117. Trusted Block Create required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 990 IBM @server zSeries 890		This callable service is not supported.
IBM @server zSeries 900		This callable service is not supported.
IBM System z9 EC IBM System z9 BC	Cryptographic Express2 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	

## TR-31 Export (CSNBT31X and CSNET31X)

Use the TR-31 Export callable service to convert a CCA token to TR-31 format for export to another party. Since there is not always a one-to-one mapping between the key attributes defined by TR-31 and those defined by CCA, the caller may need to specify the attributes to attach to the exported key through the rule array.

The callable service name for AMODE(64) is CSNET31X.



## Format

```
CALL CSNBT31X(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_version_number,
    key_field_length,
    source_key_identifier_length,
    source_key_identifier,
    unwrap_kek_identifier_length,
    unwrap_kek_identifier,
    wrap_kek_identifier_length,
    wrap_kek_identifier,
    opt_blks_length,
    opt_blocks,
    tr31_key_block_length,
    tr31_key_block )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The *rule\_array\_count* parameter must be 3, 4, or 5.

**rule\_array**

Direction: Input

Type: String

The *rule\_array* contains keywords that provide control information to the callable service. The keywords are 8 bytes in length and must be left-aligned and padded on the right with space characters. The *rule\_array* keywords for this callable service are shown in the following table. See Table 119 on page 308 for valid combinations of Usage and Mode

*Table 118. Keywords for TR-31 Export Rule Array Control Information*

Keyword	Meaning
<b>TR-31 key block protection method – one required</b>	
VARXOR-A	Use the variant method corresponding to a TR-31 Key Block Version ID of “A” (0x41)
VARDRV-B	Use the key derivation method corresponding to a TR-31 Key Block Version ID of “B” (0x42)
VARXOR-C	Use the variant method corresponding to a TR-31 Key Block Version ID of “C” (0x43)
<b>TR-31 key usage values for output key – one required</b>	
<b>Note:</b> If ATTR-CV is specified from the Control Vector Transport group, then usage keyword must not be specified. The proprietary usage ‘10’ will be used.	
BDK	Base Derivation Key (BDK) – ( B0 )
CVK	Card Verification Key (CVK) – ( C0 )
ENC	Data encryption key – ( D0 )
EMVACMK	EMV application cryptogram master key – ( E0 )
EMVSCMK	EMV secure messaging for confidentiality master key – ( E1 )
EMVSIMK	EMV secure messaging for integrity master key – ( E2 )
EMVDAMK	EMV data authentication code key – ( E3 )
EMVDNMK	EMV dynamic numbers master key – ( E4 )
EMVCPMK	EMV card personalization master key – ( E5 )
KEK	Key-encrypting key – ( K0 )
KEK-WRAP	Key-encrypting key for wrapping TR-31 blocks (for ‘B’ and ‘C’ TR-31 Key Block Version IDs only) – ( K1 )
ISOMAC0	Key for ISO 16609 MAC algorithm 1 using TDES – ( M0 )
ISOMAC1	Key for ISO 9797-1 MAC algorithm 1– ( M1 )
ISOMAC3	Key for ISO 9797-1 MAC algorithm 3– ( M3 )
PINENC	PIN encryption key – ( P0 )
PINVO	PIN verification key, “other” algorithm – ( V0 )
PINV3624	PIN verification key for IBM 3624 algorithm – ( V1 )
VISAPVV	PIN verification key, VISA PVV algorithm – ( V2 )
<b>TR-31 modes of key use – one required</b>	
<b>Note:</b> If ATTR-CV is specified from the Control Vector Transport group, then mode keyword must not be specified. The proprietary mode ‘1’ will be used.	
ENCDEC	Encrypt and decrypt – ( B )
DEC-ONLY	Decrypt only – ( D )
ENC-ONLY	Encrypt only – ( E )

Table 118. Keywords for TR-31 Export Rule Array Control Information (continued)

Keyword	Meaning
GENVER	MAC or PIN generate and verify – ( C ) <ul style="list-style-type: none"> <li>MAC key must have Gen and Ver bits on</li> <li>PIN key must have any PINGEN bit and EPINVER bit on</li> </ul>
GEN-ONLY	MAC or PIN generate only – ( G ) <ul style="list-style-type: none"> <li>MAC key must have only Gen bit on</li> <li>PIN key must have any PINGEN bit on and EPINVER bit off</li> </ul>
VER-ONLY	MAC or PIN verify only – ( V ) <ul style="list-style-type: none"> <li>MAC key must have only Ver bit on</li> <li>PIN key must have all PINGEN bits off and EPINVER bit on</li> </ul>
DERIVE	Key Derivation(for 'B' and 'C' TR-31 Key Block Version IDs only) – ( X )
ANY	Any mode allowed – ( N )
<b>Export control to set export field in TR-31 key block – optional</b>	
EXP-ANY	Export allowed using any key-encrypting key. This is the default.
EXP-TRST	Export allowed using a trusted key-encrypting key, as defined in TR-31. <b>Note:</b> A CCA key wrapped in the X9.24 compliant CCA key block is considered a trusted key.
EXP-NONE	Export prohibited
<b>Control vector transport control – optional</b> <b>Note:</b> If no keyword from this group is supplied, the CV in the <i>source_key_identifier</i> is still verified to agree with the 'key usage' and 'mode of use' keywords specified from the groups above.	
INCL-CV	Include the CCA Control Vector as an optional field in the TR-31 key block header. The TR-31 usage and mode of use fields will indicate the key attributes, and those attributes (derived from the keywords passed from the above groups) will be verified by the callable service to be compatible with the ones in the included control vector.
ATTR-CV	Include the CCA Control Vector as an optional field in the TR-31 key block header. The TR-31 usage will be set to the proprietary ASCII value "10" ('3130'x) to indicate usage information is specified in the included CV, and the mode of use will be set to the proprietary ASCII value "1" ('31'x) to indicate that mode is likewise specified in the CV. <b>Note:</b> If this keyword is specified, then usage and mode keywords from the preceding groups must not be specified. The proprietary values will be used.

**key\_version\_number**

Direction: Input

Type: String

The two bytes from this parameter are copied into the Key Version Number field of the output TR-31 key block. If no key version number is needed, the value must be 0x3030 ("00"). If the CCA key in parameter *source\_key\_identifier* is a key part (CV bit 44 is 1) then the key version number in the TR-31 key block is set to "c0" (0x6330) according to the TR-31 standard, which indicates

that the TR-31 block contains a key part. In this case, the value passed to the callable service in the *key\_version\_number* parameter is ignored.

### **key\_field\_length**

Direction: Input

Type: Integer

This parameter specifies the length of the key field which is encrypted in the TR-31 block. The length must be a multiple of 8, the DES cipher block size, and it must be greater than or equal to the length of the cleartext key passed with parameter *source\_key\_identifier* plus the length of the 2-byte key length that precedes this key in the TR-31 block. For example, if the source key is a double-length TDES key of length 16 bytes, then the key field length must be greater than or equal to (16+2) bytes, and must also be a multiple of 8. This means that the minimum *key\_field\_length* in this case would be 24. TR-31 allows a variable number of padding bytes to follow the cleartext key, and the caller may choose to pad with more than the minimum number of bytes needed to form a block that is a multiple of 8. This is generally done to hide the length of the cleartext key from those who cannot decipher that key. Most often, all keys – single, double, or triple length – are padded to the same length so that it is not possible to determine which length is carried in the TR-31 block by examining the encrypted block.

Note that this parameter is not expected to allow for ASCII encoding of the encrypted data stored in the key field according to the TR-31 specification. For example when the user passes a value of 24 here, following the minimum example above, the length of the final ASCII-encoded encrypted data in the key field in the output TR-31 key block will be 48 Bytes.

### **source\_key\_identifier\_length**

Direction: Input

Type: Integer

This parameter specifies the length of the *source\_key\_identifier* parameter, in bytes. The value in this parameter must currently be 64, since only CCA key tokens are supported for the source key parameter.

### **source\_key\_identifier**

Direction: Input/Output

Type: String

This parameter contains either the label or the key token for the key that is to be exported. The key must be a CCA internal or external token. If the source key is an external token, an identifier for the KEK that wraps the source key must be passed in the *unwrap\_kek\_identifier* parameter. Only DES/TDES keys are supported. If a key token is passed which is wrapped under the old master key, it will be updated on output so that it is wrapped under the current master key.

### **unwrap\_kek\_identifier\_length**

Direction: Input

Type: Integer

This parameter specifies the length of the *unwrap\_kek\_identifier* parameter, in bytes. If the *source\_key\_identifier* is an external CCA token, then this parameter must be 64. Otherwise, this parameter must be 0.

### **unwrap\_kek\_identifier**

Direction: Input/Output

Type: String

When the *source\_key\_identifier* is an external CCA token, this parameter contains either the label or the key token for the KEK which the *source\_key\_identifier* is currently wrapped under. It must be a CCA internal DES

KEK token of type EXPORTER or OKEYXLAT. If the *source\_key\_identifier* is not an external CCA token, this parameter is ignored. If a key token is passed which is wrapped under the old master key, it will be updated on output so that it is wrapped under the current master key.

### **wrap\_kek\_identifier\_length**

Direction: Input

Type: Integer

This parameter specifies the length of the *wrap\_kek\_identifier* parameter, in bytes. If the *unwrap\_kek\_identifier* is also to be used to wrap the output TR-31 key block, specify 0 for this parameter. Otherwise, this parameter must be 64.

### **wrap\_kek\_identifier**

Direction: Input/Output

Type: String

When *wrap\_kek\_identifier\_length* is 0, this parameter is ignored and the *unwrap\_kek\_identifier* is also to be used to wrap the output TR-31 key block. Otherwise, this parameter contains either the label or the key token for the KEK to use for wrapping the output TR-31 key block. It must be a CCA internal token for a KEK EXPORTER or OKEYXLAT type and must have the same clear key as the *unwrap\_kek\_identifier*. If a key token is passed which is wrapped under the old master key, it will be updated on output so that it is wrapped under the current master key.

**Note:** ECB-mode wrapped DES keys (CCA legacy wrap mode) cannot be used to wrap/unwrap TR-31 version 'B'/'C' key blocks that have/will have 'E' exportability, because ECB-mode does not comply with ANSI X9.24 Part 1.

This parameter exists to allow for KEK separation, it is possible that KEKs will be restricted as to what they can wrap, such that a KEK for wrapping CCA external keys may not be usable for wrapping TR-31 external keys, or vice versa.

### **opt\_blks\_length**

Direction: Input

Type: Integer

This parameter specifies the length of parameter *opt\_blocks* in bytes. If no optional data is to be included in the TR-31 key block, this parameter must be set to zero.

### **opt\_blocks**

Direction: Input

Type: String

This parameter contains optional block data which is to be included in the output TR-31 key block. The optional block data is prepared using the TR-31 Optional Data Build callable service, and must be in ASCII. This parameter is ignored if *opt\_blks\_length* is zero.

### **TR31\_key\_block\_length**

Direction: Input/Output

Type: Integer

This parameter specifies the length of the *TR31\_key\_block* parameter, in bytes. On input, it must specify the size of the buffer available for the output TR-31 key block, and on return it is updated to contain the actual length of that returned key block. If the provided buffer is not large enough for the output TR-31 key block an error is returned. The maximum size of the output TR-31 key block is 9992 bytes.

**TR31\_key\_block**

Direction: Output

Type: String

This parameter specifies the location of the exported TR-31 key block wrapped with the export key provided in the *wrap\_kek\_identifier* parameter.

**Restrictions**

This callable service only exports DES and TDES keys.

Proprietary values for the TR-31 header fields are not supported by this callable service with the exception of the proprietary values used by IBM CCA when carrying a control vector in an optional block in the header.

**Usage Notes**

Unless otherwise noted, all String parameters that are either written to, or read from, a TR-31 key block will be in EBCDIC format. Input parameters are converted to ASCII before being written to the TR-31 key block and output parameters are converted to EBCDIC before being returned (see Appendix G, “EBCDIC and ASCII Default Conversion Tables,” on page 959). TR-31 key blocks themselves are always in printable ASCII format as required by the ANSI TR-31 specification.

If keyword INCL-CV or ATTR-CV is specified, the service inserts the CCA control vector from the source key into an optional data field in the TR-31 header. The TR-31 Import callable service can extract this CV and use it as the CV for the CCA key it creates when importing the TR-31 block. This provides a way to use TR-31 for transport of CCA keys and to make the CCA key have identical control vectors on the sending and receiving nodes. The difference between INCL-CV and ATTR-CV is that INCL-CV is a normal TR-31 export in which the TR-31 key attributes are set based on the supplied rule array keywords but the CV is also included in the TR-31 block to provide additional detail. In contrast, the ATTR-CV causes the service to include the CV but to set both the TR-31 usage and mode of use fields to proprietary values which indicate that the usage and mode information are specified in the CV and not in the TR-31 header. For option INCL-CV, the export operation is still subject to the restrictions imposed by the settings of the relevant access control points. For option ATTR CV, those access control points are not checked and any CCA key can be exported as long as the export control fields in the CV permit it.

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

Note that the optional data, if present, must not already contain a padding Block, ID “PB”. A Padding Block of the appropriate size, if needed, will be added when building the TR-31 key block. If this callable service encounters a padding block in the optional block data, an error will occur.

**Access Control Points**

The access control points in the ICSF role that control the general function of this service are:

- TR31 Export – Permit version A TR-31 key blocks
- TR31 Export – Permit version B TR-31 key blocks
- TR31 Export – Permit version C TR-31 key blocks

## TR-31 Export

If the wrap KEK identifier is a weaker key than the key being exported, then:

- the service will fail if the **Prohibit weak wrapping - Transport keys** access control point is enabled.
- the service will complete successfully with a warning return code if the **Warn when weak wrap - Transport keys** access control point is enabled.

The following table lists the valid attribute translations for export of CCA keys to TR-31 key blocks along with the access control points which govern those translations. Any translation not listed here will result in an error. If an individual cell is blank, it represents the value of the cell immediately above it.

**Note:** In order to export a CCA key to a TR-31 key block, the appropriate key block version ACP needs to be enabled in addition to any required translation specific ACPs from below.

Table 119. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs)

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
Any Exportable Key							
Permit export of any CCA key (for allowable export scenarios as defined by this table) as long as the TR-31 key block will have the CCA Control Vector (CV) included as an optional block (the INCL-CV keyword was supplied on the callable service).							
Normally export of CCA keys to TR-31 key blocks is controlled by ACPs specific to the translation or a small set of translations, to give fine control. This ACP allows any allowable export to occur as long as the CV is included, thus overriding the specific ACPs.							
Notes:							
1. Some target systems, produced by other vendors may not accept TR-31 key blocks with the proprietary optional CV block.							
2. The ATTR-CV keyword does not require any ACPs.							
DUKPT Base Derivation Keys							
KEYGENKY	UKPT	BDK + ANY	B0	A	N	T	Permit KEYGENKY:UKPT to B0
KEYGENKY	UKPT	BDK + DERIVE	B0	B,C	X	T	
Note: These are the base keys from which DUKPT initial keys are derived for individual devices such as PIN pads							
Card Verification Keys							
MAC	AMEX-CSC, gen bit(20)=1	CVK + GEN-ONLY	C0	A,B,C	G	D,T	Permit MAC/MACVER:AMEX-CSC to C0:G/C/V
	AMEX-CSC, gen bit(20)=0, ver bit(21)=1	CVK + VER-ONLY		A,B,C	V	D,T	
	AMEX-CSC, gen bit(20)=1, ver bit(21)=1	CVK + GENVER		A,B,C	C	D,T	
	CVVKEY-A, gen bit(20)=1	CVK + GEN-ONLY		A,B,C	G	T	Permit MAC/MACVER:CVV-KEYA to C0:G/C/V
	CVVKEY-A, gen bit(20)=0, ver bit(21)=1	CVK + VER-ONLY		A,B,C	V	T	
	CVVKEY-A, gen bit(20)=1, ver bit(21)=1	CVK + GENVER		A,B,C	C	T	



Table 119. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs) (continued)

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
	ANY-MAC, gen bit(20)=1	CVK + GEN-ONLY		A,B,C	G	T	Permit MAC/MACVER:ANY-MAC to C0:G/C/V
	ANY-MAC, gen bit(20)=0, ver bit(21)=1	CVK + VER-ONLY		A,B,C	V	T	
	ANY-MAC, gen bit(20)=1, ver bit(21)=1	CVK + GENVER		A,B,C	C	T	
DATA	gen bit(20)=1 or zeroCV	CVK + GEN-ONLY		A,B,C	G	T	Permit DATA to C0:G/C
	gen bit(20)=1, ver bit(21)=1 or zeroCV	CVK + GENVER		A,B,C	C	T	

**Notes:**

- Keys for computing or verifying (against supplied value) a card verification code with the CVV, CVC, CVC2 and CVV2 algorithms. In CCA, this corresponds to keys used with two different APIs.
  - Visa CVV and MasterCard CVC codes are computed with CVV\_Generate and verified with CVV\_Verify. Keys must be DATA or MAC with sub-type (in bits 0-3) "ANY-MAC", "CVVKEY-A" or "CVVKEY-B". The GEN bit (20) or VER bit (21) must be set appropriately.
  - American Express CSC codes are generated and verified with the Transaction\_Validate verb. The key must be a MAC or MACVER key with sub-type "ANY-MAC" or "AMEX-CSC". The GEN bit (20) or VER bit (21) must be set appropriately.
- CCA and TR-31 represent CVV keys incompatibly. CCA represents the "A" and "B" keys as two 8 B keys, while TR-31 represents these as one 16 B key. The CVV generate and verify verbs now accept a 16 B CVV key, using left and right parts as A and B. Current Visa standards require this.
- Import and export of the 8 B CVVKEY-A and CVVKEY-B types will only be allowed using the proprietary TR-31 usage+mode values to indicate encapsulation of the IBM CV in an optional block, since the 8 B CVVKEY-A is meaningless / useless as a TR-31 C0 usage key of any mode.
- It is possible to convert a CCA CVV key into a CSC key or vice-versa, since the translation from TR 31 usage "C0" is controlled by rule array keywords on the import verb. This can be restricted by using ACPs, but if both of translation types are required they cannot be disabled and control is up to the development, deployment, and execution of the applications themselves

**Data Encryption Keys**

ENCIPHER	(none)	ENC + ENC-ONLY	D0	A,B,C	E	D, T	Permit ENCIPHER/DECIPHER/CIPHER to D0:E/D/B
DECIPHER	(none)	ENC + DEC-ONLY		A,B,C	D	D, T	
CIPHER	(none)	ENC + ENCDEC		A,B,C	B	D, T	
DATA	enc bit(18)=1, dec bit(19)=1 or zeroCV	ENC + ENCDEC		A,B,C	B	D, T	Permit DATA to D0:B

**Note:** There is asymmetry in the TR-31 to CCA and CCA to TR-31 translation. CCA keys can be exported to TR-31 'D0' keys from CCA type ENCIPHER, DECIPHER, or CIPHER, or type DATA with proper Encipher and Decipher CV bits on. A TR-31 'D0' key can only be imported to CCA types ENCIPHER, DECIPHER, or CIPHER, not the lower security DATA key type. This eliminates conversion to the lower security DATA type by export / re-import.

**Key Encrypting Keys**

EXPORTER or OKEYXLAT		KEK + ENC-ONLY	K0	A,B,C	E	T	Permit EXPORTER/OKEYXLAT to K0:E
IMPORTER or IKEYXLAT		KEK + DEC-ONLY	K0	A,B,C	D	T	Permit IMPORTER/IKEYXLAT to K0:D



## TR-31 Export

Table 119. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs) (continued)

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
EXPORTER or OKEYXLAT		KEK-WRAP + ENC-ONLY	K1	B,C	E	T	Permit EXPORTER/OKEYXLAT to K1:E
IMPORTER or IKEYXLAT		KEK-WRAP + DEC-ONLY	K1	B,C	D	T	Permit IMPORTER/IKEYXLAT to K1:D
<b>Notes:</b> <ol style="list-style-type: none"> <li>To be exported a KEK must have either the EXPORTER/IMPORTER bit or the XLAT bit on in the CV. A KEK with only the Key Generate bits on will not be exportable.</li> <li>'K1' keys are not distinguished from 'K0' keys within CCA. The 'K1' key is a particular KEK for deriving keys used in the 'B' or 'C' version wrapping of TR-31 key blocks. CCA does not distinguish between targeted protocols currently and so there is no good way to represent the difference; also note that most wrapping mechanisms now involve derivation or key variation steps</li> <li>The CCA KEK to TR-31 K0-B transition for export will not be allowed for security reasons, even with ACP control this gives an immediate path to turn a CCA EXPORTER to an IMPORTER and vice versa. Export of NO-CV KEKs will be allowed, exporter keys become 'E' mode normal K0 keys, importer keys become 'D' mode K0 keys. A user can turn any KEK to a NO-CV KEK by setting the flag bit and recalculating the TVV, the flag is not bound to the key like the CV is.</li> </ol>							
<b>MAC Keys</b>							
MAC	gen bit(20)=1	ISOMAC0 + GEN-ONLY	M0	A,B,C	G	T	Permit MAC/DATA/DATAM to M0:G/C
DATA	gen bit(20)=1 or zeroCV	ISOMAC0 + GEN-ONLY		A,B,C	G	T	
MAC	gen bit(20)=1, ver bit(21)=1	ISOMAC0 + GENVER		A,B,C	C	T	
DATAM	gen bit(20)=1, ver bit(21)=1	ISOMAC0 + GENVER		A,B,C	C	T	
DATA	gen bit(20)=1, ver bit(21)=1 or zeroCV	ISOMAC0 + GENVER		A,B,C	C	T	
MACVER	gen bit(20)=0, ver bit(21)=1	ISOMAC0 + VER-ONLY		A,B,C	V	T	Permit MACVER/DATAMV to M0:V
DATAMV	gen bit(20)=0, ver bit(21)=1	ISOMAC0 + VER-ONLY		A,B,C	V	T	
MAC	gen bit(20)=1	ISOMAC1 + GEN-ONLY	M1	A,B,C	G	D,T	Permit MAC/DATA/DATAM to M1:G/C
DATA	gen bit(20)=1 or zeroCV	ISOMAC1 + GEN-ONLY		A,B,C	G	D,T	
MAC	gen bit(20)=1, ver bit(21)=1	ISOMAC1 + GENVER		A,B,C	C	D,T	
DATAM	gen bit(20)=1, ver bit(21)=1	ISOMAC1 + GENVER		A,B,C	C	D,T	
DATA	gen bit(20)=1, ver bit(21)=1 or zeroCV	ISOMAC1 + GENVER		A,B,C	C	D,T	
MACVER	gen bit(20)=0, ver bit(21)=1	ISOMAC1 + VER-ONLY		A,B,C	V	D,T	Permit MACVER/DATAMV to M1:V
DATAMV	gen bit(20)=0, ver bit(21)=1	ISOMAC1 + VER-ONLY		A,B,C	V	D,T	

Table 119. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs) (continued)

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error		T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
MAC	gen bit(20)=1	ISOMAC3 + GEN-ONLY	M3	A,B,C	G	D,T	Permit MAC/DATA/DATAM to M3:G/C
DATA	gen bit(20)=1 or zeroCV	ISOMAC3 + GEN-ONLY		A,B,C	G	D,T	
MAC	gen bit(20)=1, ver bit(21)=1	ISOMAC3 + GENVER		A,B,C	C	D,T	
DATAM	gen bit(20)=1, ver bit(21)=1	ISOMAC3 + GENVER		A,B,C	C	D,T	
DATA	gen bit(20)=1, ver bit(21)=1 or zeroCV	ISOMAC3 + GENVER		A,B,C	C	D,T	
MACVER	gen bit(20)=0, ver bit(21)=1	ISOMAC3 + VER-ONLY		A,B,C	V	D,T	Permit MACVER/DATAMV to M3:V
DATAMV	gen bit(20)=0, ver bit(21)=1	ISOMAC3 + VER-ONLY		A,B,C	V	D,T	
<b>Notes:</b>							
1. M0 and M1 are identical (ISO 16609 based on ISO 9797) normal DES/TDES (CBC) MAC computation, except M1 allows 8 byte and 16 byte keys while M0 allows only 16 byte keys. Mode M3 is the X9.19 style triple-DES MAC.							
2. CCA does not support M2, M4, or M5.							
3. Although export of DATAM/DATAMV keys to TR-31 M0/M1/M3 key types is allowed, import to DATAM/DATAMV CCA types is not allowed since they are obsolete types.							
PIN Keys							
OPINENC	(none)	PINENC + ENC-ONLY	P0	A,B,C	E	T	Permit OPINENC to P0:E
IPINENC	(none)	PINENC + DEC-ONLY		A,B,C	D	T	Permit IPINENC to P0:D
(none)	(none)	(none)	(none)	A,B,C	B	T	(none)
PINVER	NO-SPEC	PINVO + ANY	V0	A	N	T	Permit PINVER:NO-SPEC to V0, Permit PINGEN/PINVER to V0/V1/V2:N
	[no GEN bits on in CV]	PINVO + VER-ONLY		A,B,C	V		Permit PINVER:NO-SPEC to V0
PINGEN	NO-SPEC	PINVO + ANY		A	N	T	Permit PINGEN:NO-SPEC to V0, Permit PINGEN/PINVER to V0/V1/V2:N
	[EPINVER bit off in CV]	PINVO + GEN-ONLY		A,B,C	G		Permit PINGEN:NO-SPEC to V0
	[EPINVER bit on in CV]	PINVO + GENVER		A,B,C	C		Permit PINGEN:NO-SPEC to V0
PINVER	IBM or NO-SPEC	PINV3624 + ANY	V1	A	N	T	Permit PINVER:NO-SPEC/IBM-PIN/IBM-PINO to V1, Permit PINGEN/PINVER to V0/V1/V2:N
	[no GEN bits on in CV]	PINV3624 + VER-ONLY		A,B,C	V		Permit PINVER:NO-SPEC/IBM-PIN/IBM-PINO to V1
PINGEN	IBM or NO-SPEC	PINV3624 + ANY		A	N	T	Permit PINGEN:NO-SPEC/IBM-PIN/IBM-PINO to V1, Permit PINGEN/PINVER to V0/V1/V2:N
	[EPINVER bit off in CV]	PINV3624 + GEN-ONLY		A,B,C	G		Permit PINGEN:NO-SPEC/IBM-PIN/IBM-PINO to V1

## TR-31 Export

Table 119. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs) (continued)

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
	[EPINVER bit on in CV]	PINV3624 + GENVER		A,B,C	C		Permit PINGEN:NO-SPEC/IBM-PIN/IBM-PINO to V1
PINVER	VISAPVV or NO-SPEC	VISAPVV + ANY	V2	A	N	T	Permit PINVER:NO-SPEC/VISA-PVV to V2, Permit PINGEN/PINVER to V0/V1/V2:N
	[no GEN bits on in CV]	VISAPVV + VER-ONLY		A,B,C	V		Permit PINVER:NO-SPEC/VISA-PVV to V2
PINGEN	VISAPVV or NO-SPEC	VISAPVV + ANY		A	N	T	Permit PINGEN:NO-SPEC/VISA-PVV to V2, Permit PINGEN/PINVER to V0/V1/V2:N
	[EPINVER bit off in CV]	VISAPVV + PINGEN		A,B,C	G		Permit PINGEN:NO-SPEC/VISA-PVV to V2
	[EPINVER bit on in CV]	VISAPVV + PINGEN		A,B,C	C		Permit PINGEN:NO-SPEC/VISA-PVV to V2
<p><b>Note:</b> There is a subtle difference between TR-31 V0 mode and CCA 'NO-SPEC' subtype. V0 mode restricts keys from 3224 or PVV methods, while CCA 'NO-SPEC' allows any method.</p> <p>Turning on the ACP(s) controlling export of PINVER to usage:mode V*:N and import of V*:N to PINGEN at the same time will allow changing PINVER keys to PINGEN keys. This is not recommended. This is possible because legacy (TR-31 2005-based) implementations used the same mode 'N' for PINGEN as well as PINVER keys.</p>							
<b>EMV Chip / Issuer Master Keys</b>							
DKYGENKY	DKYL0 + DMAC	EMVACMK + ANY	E0	A	N	T	Permit DKYGENKY:DKYL0+DMAC to E0
		EMVACMK + DERIVE		B,C	X	T	
	DKYL0 + DMV	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL0+DMV to E0 0x019A
		EMVACMK + DERIVE		B,C	X	T	
	DKYL0 + DALL	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL0+DALL to E0 0x019B
		EMVACMK + DERIVE		B,C	X	T	
	DKYL1 + DMAC	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL1+DMAC to E0
		EMVACMK + DERIVE		B,C	X	T	
	DKYL1 + DMV	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL1+DMV to E0
		EMVACMK + DERIVE		B,C	X	T	
	DKYL1 + DALL	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL1+DALL to E0
		EMVACMK + DERIVE		B,C	X	T	
DKYGENKY	(DKYL0 + DDATA)	EMVSCMK + ANY	E1	A	N	T	Permit DKYGENKY:DKYL0+DDATA to E1
		EMVSCMK + DERIVE		B,C	X	T	

Table 119. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs) (continued)

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
	(DKYL0 + DMPIN)	EMVSCMK + ANY		A	N	T	Permit DKYGENKY:DKYL0+DMPIN to E1
		EMVSCMK + DERIVE		B,C	X	T	
	DKYL0 + DALL	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL0+DALL to E1
		EMVACMK + DERIVE		B,C	X	T	
	(DKYL1 + DDATA)	EMVSCMK + ANY		A	N		Permit DKYGENKY:DKYL1+DDATA to E1
		EMVSCMK + DERIVE		B,C	X		
	(DKYL1 + DMPIN)	EMVSCMK + ANY		A	N		Permit DKYGENKY:DKYL1+DMPIN to E1
		EMVSCMK + DERIVE		B,C	X		
	DKYL1 + DALL	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL1+DALL to E1
		EMVACMK + DERIVE		B,C	X	T	
DKYGENKY	DKYL0 + DMAC	EMVSIMK + ANY	E2	A	N	T	Permit DKYGENKY:DKYL0+DMAC to E2
		EMVSIMK + DERIVE		B,C	X	T	
	DKYL0 + DALL	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL0+DALL to E2
		EMVACMK + DERIVE		B,C	X	T	
	DKYL1 + DMAC	EMVSIMK + ANY		A	N	T	Permit DKYGENKY:DKYL1+DMAC to E2
		EMVSIMK + DERIVE		B,C	X	T	
	DKYL1 + DALL	EMVACMK + ANY		A	N	T	Permit DKYGENKY:DKYL1+DALL to E2
		EMVACMK + DERIVE		B,C	X	T	

## TR-31 Export

Table 119. Valid CCA to TR-31 Export Translations and Required Access Control Points (ACPs) (continued)

Export CCA Type (CSNBCVG keywords)	CCA Usage in CSNBCVG keywords	CSNBT31X Keywords all 'usage' + 'mode' here, else error	T31Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Required "TR31 Export" ACP
DATA	(none)	EMVDAMK + ANY	E3	A	N	T	Permit DATA/MAC/CIPHER/ENCIPHER to E3
		EMVDAMK + DERIVE		B,C	X		
MAC (not MACVER)	(none)	EMVDAMK + ANY		A	N		
		EMVDAMK + MACGEN		A	G		
		EMVDAMK + DERIVE		B,C	X		
CIPHER	(none)	EMVDAMK + ANY		A	N		
		EMVDAMK + DERIVE		B,C	X		
ENCIPHER		EMVDAMK + ENC-ONLY		A	E		
		EMVDAMK + DERIVE		B,C	X		
DKYGENKY	DKYL0 +DDATA	EMVDNMK + ANY	E4	A	N	T	Permit DKYGENKY:DKYL0+DDATA to E4
		EMVDNMK + DERIVE		B,C	X		
	DKYL0 +DALL	EMVDNMK + ANY		A	N	T	Permit DKYGENKY:DKYL0+DALL to E4
		EMVDNMK + DERIVE		B,C	X		
DKYGENKY	DKYL0 + DEXP	EMVCPMK + ANY	E5	A	N	T	Permit DKYGENKY:DKYL0+DEXP to E5
		EMVCPMK + DERIVE		B,C	X		
	DKYL0 + DMAC	EMVCPMK + ANY		A	N		Permit DKYGENKY:DKYL0+DMAC to E5
		EMVCPMK + DERIVE		B,C	X		
	DKYL0 +DDATA	EMVCPMK + ANY		A	N		Permit DKYGENKY:DKYL0+DDATA to E5
		EMVCPMK + DERIVE		B,C	X		
	DKYL0 +DALL	EMVDNMK + ANY		A	N	T	Permit DKYGENKY:DKYL0+DALL to E5
		EMVDNMK + DERIVE		B,C	X		

**Note:** EMV Chip Card Master Keys are used by the chip cards to perform cryptographic operations, or in some cases to derive keys used to perform operations. In CCA, these are:

- Key Gen Keys of level DKYL0 or DYKL1 allowing derivation of operational keys, or
- operational keys.

EMV support in CCA is significantly different from TR-31. CCA key types do not match TR-31 types.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 120. TR-31 export required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This service is not supported.
IBM @server zSeries 990		This service is not supported.
IBM @server zSeries 890		
IBM System z9 EC		This service is not supported.
IBM System z9 BC		
IBM System z10 EC		This service is not supported.
IBM System z10 BC		
z196 z114	Crypto Express3 Coprocessor	TR-31 key support requires the Sep. 2011 or later LIC.
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## TR-31 Import (CSNBT31I and CSNET31I)

Use the TR-31 Import callable service to convert a TR-31 key block to a CCA token. Since there is not always a one-to-one mapping between the key attributes defined by TR-31 and those defined by CCA, the caller may need to specify the attributes to attach to the imported key through the rule array.

The callable service name for AMODE(64) is CSNET31I.

## Format

```
CALL CSNBT31I(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    TR31_key_block_length,
    TR31_key_block,
    unwrap_kek_identifier_length,
    unwrap_kek_identifier,
    wrap_kek_identifier_length,
    wrap_kek_identifier,
    output_key_identifier_length,
    output_key_identifier,
    num_opt_blks,
    cv_source,
    protection_method )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The *rule\_array\_count* parameter must be 1, 2, 3, 4, or 5.

**rule\_array**

Direction: Input

Type: String

The *rule\_array* contains keywords that provide control information to the callable service. The keywords are 8 bytes in length and must be left-aligned and padded on the right with space characters. The *rule\_array* keywords for this callable service are shown in the following table. One keyword from one CCA output key usage subgroup shown in the following table is required based on TR-31 input key usage, unless the CV is included in the TR-31 key block as an optional block. If the CV is included in the TR-31 key block as an optional block, the included CV will be used in the output key block as long as it does not conflict with the TR-31 header data.

See Table 123 on page 323 for valid combinations of Usage and Mode

Table 121. Keywords for TR-31 Import Rule Array Control Information

Keyword	Meaning
<b>Key Wrapping Method (One Required)</b>	
INTERNAL	Desired <i>output_key_identifier</i> is a CCA internal key token, wrapped using the card master key.
EXTERNAL	Desired <i>output_key_identifier</i> is a CCA external key token, wrapped using the key represented by the <i>unwrap_kek_identifier</i> .
<b>CCA Output Key Usage Subgroups ( One keyword from one CCA output key usage subgroup shown in the following table is required based on TR-31 input key usage, unless the CV is included in the TR-31 key block as an optional block. If the CV is included in the TR-31 key block as an optional block, the included CV will be used in the output key block as long as it does not conflict with the TR-31 header data.)</b>	
<i>C0 Subgroup (One Required for this TR-31 key usage)</i>	
CVK-CVV	Convert TR-31 CVK to a CCA key for use with CVV/CVC. The CCA key will be a MAC key with subtype CVVKEY-A.
CVK-CSC	Convert TR-31 CVK to a CCA key for use with CSC. The CCA key will be a MAC key with subtype AMEX CSC.
<i>K0 Subgroup (One Required for this TR-31 key usage)</i>	
EXPORTER	For TR-31 K0-E or K0-B usage+mode keys. Convert TR-31 KEK to a CCA wrapping key. The key will convert to a CCA EXPORTER key. Note that the K0-B key import has a unique ACP.
OKEYXLAT	For TR-31 K0-E or K0-B usage+mode keys. Convert TR-31 KEK to a CCA wrapping key. The key will convert to a CCA OKEYXLAT key. Note that the K0-B key import has a unique ACP.
IMPORTER	For TR-31 K0-D or K0-B usage+mode keys. Convert TR-31 KEK to a CCA unwrapping key. The key will convert to a CCA IMPORTER key. Note that the K0-B key import has a unique ACP.
IKEYXLAT	For TR-31 K0-D or K0-B usage+mode keys. Convert TR-31 KEK to a CCA unwrapping key. The key will convert to a CCA IKEYXLAT key. Note that the K0-B key import has a unique ACP.
<i>V0/V1/V2 Subgroup (One Required for these TR-31 key usages)</i>	
PINGEN	Convert a TR-31 PIN verification key to a CCA PINGEN key.



Table 121. Keywords for TR-31 Import Rule Array Control Information (continued)

Keyword	Meaning
PINVER	Convert a TR-31 PIN verification key to a CCA PINVER key.
<i>E0/E2,F0/F2 Subgroup (One Required for these TR-31 key usages)</i>	
DMAC	Convert TR-31 EMV master key (chip card or issuer) for Application Cryptograms or Secure Messaging for Integrity to CCA DKYGENKY type DMAC
DMV	Convert TR-31 EMV master key (chip card or issuer) for Application Cryptograms or Secure Messaging for Integrity to CCA DKYGENKY type DMV
<i>E1,F1 Subgroup (One Required for these TR-31 key usages)</i>	
DMPIN	Convert TR-31 EMV master key (chip card or issuer) for Secure Messaging for Confidentiality to CCA DKYGENKY type DMPIN
DDATA	Convert TR-31 EMV master key (chip card or issuer) for Secure Messaging for Confidentiality to CCA DKYGENKY type DDATA
<i>E5 Subgroup (One Required for this TR-31 key usage)</i>	
DMAC	Convert TR-31 EMV master key (issuer) for Card Personalization to CCA DKYGENKY type DMAC.
DMV	Convert TR-31 EMV master key (issuer) for Card Personalization to CCA DKYGENKY type DMV.
DEXP	Convert TR-31 EMV master key (issuer) for Card Personalization to CCA DKYGENKY type DEXP.
<b>Key Derivation Level (One Required with E0, E1, E2 TR-31 key usages unless the CV is included in the TR-31 key block as an optional block. If the CV is included in the TR-31 key block, the included CV will be used in the output key block as long as it does not conflict with the TR-31 header data.)</b>	
DKYL0	Convert TR-31 EMV master key (chip card or issuer) to CCA DKYGENKY at derivation level DKYL0.
DKYL1	Convert TR-31 EMV master key (chip card or issuer) to CCA DKYGENKY at derivation level DKYL1.
<b>Key Type Modifier (Optional)</b>	
NOOFFSET	Valid only for V0/V1 TR-31 key usage values. Import the PINGEN or PINVER key into a key token that cannot participate in the generation or verification of a PIN when an offset or the Visa PVV process is requested.
<b>Key Wrapping Method (Optional)</b> <b>Note:</b> Conflicts between wrapping keywords used and a CV passed in an optional data block of the TR-31 token will result in errors being returned. The main example of this is a CV that indicates 'enhanced-only' in bit 56 when the user or configured default specifies ECB for key wrapping.	
USECONFIG	Specifies that the configuration setting for the default wrapping method is to be used to wrap the key. This is the default.
WRAP-ENH	Specifies that the new enhanced wrapping method is to be used to wrap the key.
WRAP-ECB	Specifies that the original wrapping method is to be used.
<b>Translation Control (One Optional)</b>	

Table 121. Keywords for TR-31 Import Rule Array Control Information (continued)

Keyword	Meaning
ENH-ONLY	<p>Specify this keyword to indicate that the key once wrapped with the enhanced method cannot be wrapped with the original method. This restricts translation to the original method. If the keyword is not specified translation to the original method will be allowed. This turns on bit 56 in the control vector. This keyword is not valid if processing a zero CV data key.</p> <p><b>Notes:</b></p> <ol style="list-style-type: none"> <li>1. If the TR-31 block contains a CV in the optional data block that does not have bit 56 turned on, bit 56 will be turned on in the output token, since with this keyword the user is asking for this behavior. The exception to this is for CVs of all 0x00 bytes, for this case no error will be generated but the CV will remain all 0x00 bytes.</li> <li>2. Conflicts between wrapping keywords used and a CV passed in an optional data block of the TR-31 token will result in errors being returned. The main example of this is a CV that indicates 'enhanced-only' in bit 56 when the user or configured default specifies ECB for key wrapping. If the default wrapping method is ECB mode, but the enhanced mode and the ENH-ONLY restriction are desired for a particular key token, combine the ENH-ONLY keyword with the WRAP-ENH keyword.</li> </ol>

**TR31\_key\_block\_length**

Direction: Input

Type: Integer

This parameter specifies the length of the TR31\_key\_block parameter, in bytes. The length field in the TR-31 block is a 4-digit decimal number, so the maximum acceptable length is 9992 bytes.

**TR31\_key\_block**

Direction: Input

Type: String

This parameter contains the TR-31 key block that is to be imported. The key block is protected with the key passed in parameter *unwrap\_kek\_identifier*.

**unwrap\_kek\_identifier\_length**

Direction: Input

Type: Integer

This parameter specifies the length of the *unwrap\_kek\_identifier* parameter, in bytes. The value in this parameter must currently be 64, since only CCA internal key tokens are supported for the *unwrap\_kek\_identifier* parameter.

**unwrap\_kek\_identifier**

Direction: Input/Output

Type: String

This parameter contains either the label or the key token for the key that is used to unwrap and check integrity of the imported key passed in the TR31\_key\_block parameter. The key must be a CCA internal token for a KEK IMPORTER or IKEYXLAT type. If a key token is passed which is wrapped under the old master key, it will be updated on output so that it is wrapped under the current master key.

**Note:** ECB-mode wrapped DES keys (CCA legacy wrap mode) cannot be used to wrap/unwrap TR-31 version 'B'/'C' key blocks that have, or will have, 'E' exportability. This is because ECB-mode does not comply with ANSI X9.24 Part 1.

### wrap\_kek\_identifier\_length

Direction: Input

Type: Integer

This parameter specifies the length of the *wrap\_kek\_identifier* parameter, in bytes. If the *unwrap\_kek\_identifier* is also to be used to wrap the output CCA token, specify 0 for this parameter. Otherwise, this parameter must be 64.

### wrap\_kek\_identifier

Direction: Input/Output

Type: String

When *wrap\_kek\_identifier\_length* is 0, this parameter is ignored and the *unwrap\_kek\_identifier* is also to be used to wrap the output CCA token. Otherwise, this parameter contains either the label or the key token for the KEK to use for wrapping the output CCA token. It must be a CCA internal token for a KEK EXPORTER or OKEYXLAT type and must have the same clear key as the *unwrap\_kek\_identifier*. If a key token is passed which is wrapped under the old master key, it will be updated on output so that it is wrapped under the current master key.

**Note:** ECB-mode wrapped DES keys (CCA legacy wrap mode) cannot be used to wrap/unwrap TR-31 version 'B'/'C' key blocks that have/will have 'E' exportability. This is because ECB-mode does not comply with ANSI X9.24 Part 1.

### output\_key\_identifier\_length

Direction: Input/Output

Type: Integer

This parameter specifies the length of the *output\_key\_identifier* parameter, in bytes. On input, it specifies the length of the buffer represented by the *output\_key\_identifier* parameter and must be at least 64 bytes long. On output, it contains the length of the token returned in the *output\_key\_identifier* parameter.

### output\_key\_identifier

Direction: Output

Type: String

This parameter contains the key token that is to receive the imported key. The output token will be a CCA internal or external key token containing the key received in the TR-31 key block.

### num\_opt\_blocks

Direction: Output

Type: Integer

This parameter contains the number of optional blocks that are present in the TR-31 key block.

### cv\_source

Direction: Output

Type: Integer

This parameter contains information about how the control vector in the output key token was created. It can be one of the following three values:

**X'00000000'**

No CV was present in an optional block, and the output CV was

created by the callable service based on input parameters and on the attributes in the TR-31 key block header.

#### **X'00000001'**

A CV was obtained from an optional block in the TR-31 key block, and the key usage and mode of use were also specified in the TR-31 header. The callable service verified compatibility of the header values with the CV and then used that CV in the output key token.

#### **X'00000002'**

A CV was obtained from an optional block in the TR-31 key block, and the key usage and mode of use in the TR-31 header held the proprietary values indicating that key use and mode should be obtained from the included CV. The CV from the TR-31 token was used as the CV for the output key token.

Any value other than these are reserved for future use and are currently invalid.

### **protection\_method**

Direction: Output

Type: Integer

This parameter contains information about what method was used to protect the input TR-31 key block. It can have one of the following values:

#### **X'00000000'**

The TR-31 key block was protected using the variant method as identified by a Key Block Version ID value of "A" (0x41).

#### **X'00000001'**

The TR-31 key block was protected using the derived key method as identified by a Key Block Version ID value of "B" (0x42).

#### **X'00000002'**

The TR-31 key block was protected using the variant method as identified by a Key Block Version ID value of "C" (0x43). Functionally this method is the same as 'A', but to maintain consistency a different value will be returned here for 'C'.

Any value other than these are reserved for future use and are currently invalid.

## **Restrictions**

This callable service only imports DES and TDES keys.

Proprietary values for the TR-31 header fields are not supported by this callable service with the exception of the proprietary values used by IBM CCA when carrying a control vector in an optional block in the header.

## **Usage Notes**

Unless otherwise noted, all String parameters that are either written to, or read from, a TR-31 key block will be in EBCDIC format. Input parameters are converted to ASCII before being written to the TR-31 key block and output parameters are converted to EBCDIC before being returned (see Appendix G, "EBCDIC and ASCII Default Conversion Tables," on page 959). TR-31 key blocks themselves are always in printable ASCII format as required by the ANSI TR-31 specification.

### TR-31 Import

If the TR-31 key block is marked as a key component, the resulting CCA key will have the Key Part bit (bit 44) in the control vector set to 1.

The exportability attributes of the imported CCA token are set based on attributes in the TR-31 key block as described in the following table.

Table 122. Export attributes of an imported CCA token

TR-31 export attribute value	CCA action on import
Non-exportable ("N")	CCA imports the key to an internal CCA key token. CV bit 17 (export) is set to zero to indicate that the key is not exportable. CV bit 57 (TR-31 export) is set to one to indicate that the key is not exportable to TR-31.
Exportable under trusted key ("E")	If the TR-31 token is wrapped with a CCA KEK in the old ECB format, the request is rejected because that KEK is not a trusted key. If the CCA KEK is in a newer X9.24 compliant CCA key block, then the TR-31 key is imported to CCA in exactly the same way as described below for keys that are exportable under any key.
Exportable under any key ("S")	CCA imports the key to an internal CCA key token. CV bit 17 (export) is set to one to indicate that the key is exportable. CV bit 57 (TR-31 export) is set to zero to indicate that the key is also exportable to TR-31.

If necessary, use the Prohibit Export, Prohibit Exported Extended, or Restrict Key Attribute callable service to alter the export attributes of the CCA token after import.

If the TR-31 key block contains an optional block with a CCA CV of '00007D00030000000000000000000000' for a single length key or '00007D000341000000000000000000000000007D00032100000000000000000000' for a double length key, the resulting CCA token will be a zero CV DATA token.

The TR-31 key block can contain a CCA control vector in an optional data field in the header. If the CV is present, the service will check that CV for compatibility with the TR-31 key attributes to ensure the CV is valid for the key and if there are no problems it will use that CV in the CCA key token that is output by the service. If a CV is received, the import operation is not subject to any ACP controlling the importation of specific key types. The CV may be present in the TR-31 key block in two different ways, depending on options used when creating that block.

- If the TR-31 Export callable service was called with option INCL-CV, the control vector is included in the TR-31 key block and the TR-31 key usage and mode of use fields contain attributes from the set defined in the TR-31 standard. The TR-31 Import callable service checks that those TR-31 attributes are compatible with the CV included in the block. It also verifies that no rule array keywords conflict with the CV contained in the TR-31 block.
- If the TR-31 Export callable service was called with option ATTR-CV, the control vector is included in the TR-31 key block and the TR-31 key usage and mode of use fields contain proprietary values (ASCII "10" and "1", respectively) to indicate that the usage and mode information is contained in the included control vector. In this case, the TR-31 Import service uses the included CV as the control vector for the CCA key token it produces. It also verifies that the CV does not conflict with rule array keywords passed

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

## Access Control Points

The access control points in the ICSF role that control the general function of this service are:

- TR31 Import – Permit version A TR-31 key blocks
- TR31 Import – Permit version B TR-31 key blocks
- TR31 Import – Permit version C TR-31 key blocks

When the WRAP-ECB or WRAP-ENH keywords are specified and the default key-wrapping method setting does not match the keyword, the **TR31 Import - Permit override of default wrapping method** access control point must be enabled.

When the **Disallow 24-byte DATA wrapped with 16-byte Key** access control point is enabled, this service will fail if the source key is a triple-length DATA key and the DES master key is a 16-byte key.

The following table lists the valid attribute translations for import of TR-31 key blocks to CCA keys along with the access control points which govern those translations. Any translation not listed here will result in an error. If an individual cell is blank, it represents the value of the cell immediately above it.

**Note:** In order to import a TR-31 key block to a CCA key, the appropriate key block version ACP needs to be enabled in addition to any required translation specific ACPs from below.

Table 123. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
DUKPT Base Derivation Keys							
B0	A	N	T	(none)	KEYGENKY	UKPT	(none)
B0	B,C	X	T	(none)	KEYGENKY	UKPT	
B1	B,C	(none)	(none)	(none)	(none)	(none)	
Note: These are the base keys from which DUKPT initial keys are derived for individual devices such as PIN pads.							
Card Verification Keys							
C0	A,B,C	G, C	D	CVK-CSC	MAC	AMEX-CSC	Permit C0 to MAC/MACVER:AMEX-CSC
	A,B,C		T	CVK-CSC	MAC	AMEX-CSC	
	A,B,C	V	D	CVK-CSC	MACVER	AMEX-CSC	
	A,B,C		T	CVK-CSC	MACVER	AMEX-CSC	
	A,B,C	G, C	T	CVK-CVV	MAC	CVVKEY-A	Permit C0 to MAC/MACVER:CVVKEY-A
	A,B,C	V	T	CVK-CVV	MACVER	CVVKEY-A	



## TR-31 Import

Table 123. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
<p>The card verification keys are keys for computing or verifying (against supplied value) a card verification code with the CVV, CVC, CVC2 and CVV2 algorithms.</p> <p><b>Notes:</b></p> <ol style="list-style-type: none"><li>In CCA, this corresponds to keys used with two different APIs.<ul style="list-style-type: none"><li>Visa CVV and MasterCard CVC codes are computed with CVV_Generate and verified with CVV_Verify. Keys must be DATA or MAC with sub-type (in bits 0-3) "ANY-MAC" , "CVVKEY-A" or "CVVKEY-B". The GEN bit (20) or VER bit (21) must be set appropriately.</li><li>American Express CSC codes are generated and verified with the Transaction_Validate verb. The key must be a MAC or MACVER key with sub-type "ANY-MAC" or "AMEX-CSC". The GEN bit (20) or VER bit (21) must be set appropriately.</li></ul></li><li>CCA and TR-31 represent CVV keys incompatibly. CCA represents the "A" and "B" keys as two 8 B keys, while TR-31 represents these as one 16 B key. The CVV generate and verify verbs now accept a 16 B CVV key, using left and right parts as A and B. Current Visa standards require this.</li><li>Import and export of the 8 B CVVKEY-A and CVVKEY-B types will only be allowed using the proprietary TR-31 usage+mode values to indicate encapsulation of the IBM CV in an optional block, since the 8 B CVVKEY-A is meaningless / useless as a TR-31 C0 usage key of any mode.</li><li>Import of a TR-31 key of usage C0 to CCA key type 'ANY-MAC' will not be allowed, although the ANY-MAC key is also usable for card verification purposes.</li><li>It is possible to convert a CCA CVV key into a CSC key or vice-versa, since the translation from TR-31 usage "C0" is controlled by rule array keywords on the import verb. This can be restricted by using ACPs, but if both of translation types are required they cannot be disabled and control is up to the development, deployment, and execution of the applications themselves.</li></ol> <p>CCA does not have a 'MAC GEN ONLY' key type, so TR-31 usage of G will translate to a full MAC key.</p>							
Data Encryption Keys							
D0	A,B,C	E	D, T	(none)	ENCIPHER	(none)	(none)
	A,B,C	D	D, T	(none)	DECIPHER	(none)	
	A,B,C	B	D, T	(none)	CIPHER	(none)	
<p><b>Notes:</b></p> <ol style="list-style-type: none"><li>There is asymmetry in the TR-31 to CCA and CCA to TR-31 translation. CCA keys can be exported to TR-31 'D0' keys from CCA type ENCIPHER, DECIPHER, or CIPHER, or type DATA with proper Encipher and Decipher CV bits on. A TR-31 'D0' key can only be imported to CCA types ENCIPHER, DECIPHER, or CIPHER, not the lower security DATA key type. This eliminates conversion to the lower security DATA type by export / re-import.</li><li>There are no ACPs controlling import since the intent of the TR-31 key's control is not interpreted, just directly translated to CCA control.</li></ol>							
Key Encrypting Keys							
K0	A,B,C	E	T	OKEYXLAT	OKEYXLAT	(none)	Permit K0:E to EXPORTER/OKEYXLAT
	A,B,C			EXPORTER	EXPORTER	(none)	
	A,B,C	D	T	IKEYXLAT	IKEYXLAT	(none)	Permit K0:D to IMPORTER/IKEYXLAT
	A,B,C			IMPORTER	IMPORTER	(none)	
	A,B,C	B	T	OKEYXLAT	OKEYXLAT	(none)	Permit K0:B to EXPORTER/OKEYXLAT
	A,B,C			EXPORTER	EXPORTER	(none)	
	A,B,C			IKEYXLAT	IKEYXLAT	(none)	Permit K0:B to IMPORTER/IKEYXLAT
	A,B,C			IMPORTER	IMPORTER	(none)	

Table 123. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
K1	B,C	E	T	OKEYXLAT	OKEYXLAT	(none)	Permit K1:E to EXPORTER/OKEYXLAT
	B,C			EXPORTER	EXPORTER	(none)	
	B,C	D	T	IKEYXLAT	IKEYXLAT	(none)	Permit K1:D to IMPORTER/IKEYXLAT
	B,C			IMPORTER	IMPORTER	(none)	
	B,C	B	T	OKEYXLAT	OKEYXLAT	(none)	Permit K1:B to EXPORTER/OKEYXLAT
	B,C			EXPORTER	EXPORTER	(none)	
	B,C			IKEYXLAT	IKEYXLAT	(none)	Permit K1:B to IMPORTER/IKEYXLAT
	B,C			IMPORTER	IMPORTER	(none)	
<b>Notes:</b>							
1. K1' keys are not distinguished from 'K0' keys within CCA. The 'K1' key is a particular KEK for deriving keys used in the 'B' or 'C' version wrapping of TR-31 key blocks. CCA does not distinguish between targeted protocols currently and so there is no good way to represent the difference; also note that most wrapping mechanisms now involve derivation or key variation steps.							
2. It is possible to convert a CCA EXPORTER key to an OKEYXLAT, or to convert an IMPORTER to an IKEYXLAT by export / re-import. This can be restricted by using ACPs, but if both translations are required they cannot be disabled and control is up to the development, deployment, and execution of the applications themselves.							
3. It will not be possible to export a CCA key to TR-31 type K0-B, in order to avoid the ability to translate a CCA EXPORTER to a CCA IMPORTER via export/import to the TR-31 token type. When a TR-31 key block does not have an included CV as an optional block, the default CV will be used to construct the output token. For IMPORTER / EXPORTER keys this means that the Key Generate bits will also be on in the KEK.							
<b>MAC Keys</b>							
M0	A,B,C	G,C	T	(none)	MAC	ANY-MAC	Permit M0/M1/M3 to MAC/MACVER:ANY-MAC
	A,B,C	V	T	(none)	MACVER	ANY-MAC	
M1	A,B,C	G,C	D, T	(none)	MAC	ANY-MAC	
	A,B,C	V	D, T	(none)	MACVER	ANY-MAC	
M3	A,B,C	G,C	D, T	(none)	MAC	ANY-MAC	
	A,B,C	V	D, T	(none)	MACVER	ANY-MAC	
<b>Notes:</b>							
1. M0 and M1 are identical (ISO 16609 based on ISO 9797) normal DES/TDES (CBC) MAC computation, except M1 allows 8 byte and 16 byte keys while M0 allows only 16 byte keys. Mode M3 is the X9.19 style triple-DES MAC.							
2. CCA does not support M2, M4, or M5.							
3. Although export of DATAM/DATAMV keys to TR-31 M0/M1/M3 key types is allowed, import to DATAM/DATAMV CCA types is not allowed since they are obsolete types							
<b>PIN Keys</b>							
P0	A,B,C	E	T	(none)	OPINENC	(none)	Permit P0:E to OPINENC
	A,B,C	D		(none)	IPINENC	(none)	Permit P0:D to IPINENC
	A,B,C	B – not supp		(none)	(none)	(none)	(none)
V0	A	N	T	PINGEN [NOOFFSET]	PINGEN	NO-SPEC [+NOOFFSET]	Permit V0 to PINGEN:NO-SPEC, Permit V0/V1/V2:N to PINGEN/PINVER



## TR-31 Import

Table 123. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
	A,B,C	G,C		[NOOFFSET]	PINGEN	NO-SPEC [+NOOFFSET]	Permit V0 to PINGEN:NO-SPEC
	A	N		PINVER [NOOFFSET]	PINVER	NO-SPEC [+NOOFFSET]	Permit V0 to PINVER:NO-SPEC, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	V		[NOOFFSET]	PINVER	NO-SPEC [+NOOFFSET]	Permit V0 to PINVER:NO-SPEC
V1	A	N	T	PINGEN [NOOFFSET]	PINGEN	IBM-PIN /IBM-PINO	Permit V1 to PINGEN:IBM-PIN/IBM-PINO, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	G,C		[NOOFFSET]	PINGEN	IBM-PIN /IBM-PINO	Permit V1 to PINGEN:IBM-PIN/IBM-PINO
	A	N		PINVER [NOOFFSET]	PINVER	IBM-PIN /IBM-PINO	Permit V1 to PINVER:IBM-PIN/IBM-PINO, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	V		[NOOFFSET]	PINVER	IBM-PIN /IBM-PINO	Permit V1 to PINVER:IBM-PIN/IBM-PINO
V2	A	N	T	PINGEN	PINGEN	VISA-PVV	Permit V2 to PINGEN:VISA-PVV, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	G,C			PINGEN	VISA-PVV	Permit V2 to PINGEN:VISA-PVV
	A	N		PINVER	PINVER	VISA-PVV	Permit V2 to PINVER:VISA-PVV, Permit V0/V1/V2:N to PINGEN/PINVER
	A,B,C	V			PINVER	VISA-PVV	Permit V2 to PINVER:VISA-PVV
<b>Notes:</b> <ol style="list-style-type: none"> <li>1. NOOFFSET keyword may be passed to specify resultant CCA key to have NOOFFSET bit (bit 37) on in CV. However this will be automatic if CV is included and has NOOFFSET bit set.</li> <li>2. NOOFFSET keyword is not supported for V2 usage since VISA-PVV algorithm does not support that concept.</li> <li>3. There is a subtle difference between TR-31 V0 mode and CCA 'NO-SPEC' subtype. V0 mode restricts keys from 3224 or PVV methods, while CCA 'NO-SPEC' allows any method.</li> <li>4. Turning on the ACP(s) controlling export of PINVER to usage:mode V*:N and import of V*:N to PINGEN at the same time will allow changing PINVER keys to PINGEN keys. This is not recommended. This is possible because legacy (TR-31 2005-based) implementations used the same mode 'N' for PINGEN as well as PINVER keys.</li> </ol>							
<b>EMV Chip / Issuer Master Keys</b>							
E0	A	N	T	DKYL0 +DMAC	DKYGENKY	DKYL0 +DMAC	Permit E0 to DKYGENKY:DKYL0+DMAC
	B,C	X		DKYL0 +DMAC		DKYL0 +DMAC	
	A	N		DKYL0 +DMV		DKYL0 +DMV	Permit E0 to DKYGENKY:DKYL0+DMV
	B,C	X		DKYL0 +DMV		DKYL0 +DMV	

Table 123. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
	A	N		DKYL1 +DMAC		DKYL1 +DMAC	Permit E0 to DKYGENKY:DKYL1+DMAC
	B,C	X		DKYL1 +DMAC		DKYL1 +DMAC	
	A	N		DKYL1 +DMV		DKYL1 +DMV	Permit E0 to DKYGENKY:DKYL1+DMV
	B,C	X		DKYL1 +DMV		DKYL1 +DMV	
E1	A	N, E, D, B	T	DKYL0 +DMPIN	DKYGENKY	DKYL0 +DMPIN	Permit E1 to DKYGENKY:DKYL0+DMPIN
	B,C	X		DKYL0 +DMPIN		DKYL0 +DMPIN	
	A	N, E, D, B		DKYL0 +DDATA		DKYL0 +DDATA	Permit E1 to DKYGENKY:DKYL0+DDATA
	B,C	X		DKYL0 +DDATA		DKYL0 +DDATA	
	A	N, E, D, B		DKYL1 +DMPIN		DKYL1 +DMPIN	Permit E1 to DKYGENKY:DKYL1+DMPIN
	B,C	X		DKYL1 +DMPIN		DKYL1 +DMPIN	
	A	N, E, D, B		DKYL1 +DDATA		DKYL1 +DDATA	Permit E1 to DKYGENKY:DKYL1+DDATA
	B,C	X		DKYL1 +DDATA		DKYL1 +DDATA	
E2	A	N	T	DKYL0 +DMAC	DKYGENKY	DKYL0 +DMAC	Permit E2 to DKYGENKY:DKYL0+DMAC
	B,C	X		DKYL0 +DMAC		DKYL0 +DMAC	
	A	N		DKYL1 +DMAC		DKYL1 +DMAC	Permit E2 to DKYGENKY:DKYL1+DMAC
	B,C	X		DKYL1 +DMAC		DKYL1 +DMAC	
E3	A	N, E, D, B, G	T	(none)	ENCIPHER	(none)	Permit E3 to ENCIPHER
	B,C	X		(none)		(none)	
E4	A	N, B	T	(none)	DKYGENKY	DKYL0 +DDATA	Permit E4 to DKYGENKY:DKYL0+DDATA
	B,C	X		(none)		DKYL0 +DDATA	
E5	A	G, C, V, E, D, B, N	T	DKYL0 +DMAC	DKYGENKY	DKYL0 +DMAC	Permit E5 to DKYGENKY:DKYL0+DMAC
	B,C	X		DKYL0 +DMAC		DKYL0 +DMAC	

## TR-31 Import

Table 123. Valid TR-31 to CCA Import Translations and Required Access Control Points (ACPs) (continued)

Import T31 Usage	T31 Key Blk Vers.	T31 Mode	T31 Alg'm	Keywords	Output CCA Type (CSNBCVG keywords)	Output CCA Usage (CSNBCVG keywords)	Required TR31 Import ACP
	A	G, C, V, E, D, B, N		DKYL0 +DDATA		DKYL0 +DDATA	Permit E5 to DKYGENKY:DKYL0+DDATA
	B,C	X		DKYL0 +DDATA		DKYL0 +DDATA	
	A	G, C, V, E, D, B, N		DKYL0 +DEXP		DKYL0 +DEXP	Permit E5 to DKYGENKY:DKYL0+DEXP
	B,C	X		DKYL0 +DEXP		DKYL0 +DEXP	
<p><b>Note:</b> EMV Chip Card Master Keys are used by the chip cards to perform cryptographic operations, or in some cases to derive keys used to perform operations. In CCA, these are:</p> <ul style="list-style-type: none"><li>• Key Gen Keys of level DKYL0 or DKYL1 allowing derivation of operational keys, or</li><li>• operational keys.</li></ul> <p>EMV support in CCA is significantly different. CCA key types do not match TR-31 types.</p>							

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 124. TR-31 export required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This service is not supported.
IBM @server zSeries 990		This service is not supported.
IBM @server zSeries 890		This service is not supported.
IBM System z9 EC		This service is not supported.
IBM System z9 BC		This service is not supported.
IBM System z10 EC		This service is not supported.
IBM System z10 BC		This service is not supported.
z196 z114	Crypto Express3 Coprocessor	TR-31 key support requires the Sept. 2011 or later LIC.
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## TR-31 Optional Data Build (CSNBT31O and CSNET31O)

A TR-31 key block can hold optional fields which are securely bound to the key block using the integrated MAC. The optional blocks may either contain information defined in the TR-31 standard, or they may contain proprietary data.

Use the TR-31 Optional Data Build callable service to construct the optional block data structure for a TR-31 key block. It builds the structure by adding one optional block with each call, until your entire set of optional blocks have been added.

With each call, the application program provides a single optional block by specifying its ID, its length, and its data in parameters *opt\_block\_id*, *opt\_block\_length*, and *opt\_block\_data* respectively. Each subsequent call appends the current optional block to any preexisting blocks in the *opt\_blocks* parameter. On the first call to the callable service, *opt\_blocks* is typically empty.

The callable service name for AMODE(64) is CSNET31O.

### Format

```
CSNBT31O(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    opt_blocks_bfr_length,
    opt_blocks_length,
    opt_blocks,
    num_opt_blocks,
    opt_block_id,
    opt_block_data_length,
    opt_block_data )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

## TR-31 Optional Data Build

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The *rule\_array\_count* parameter must be 0 since no keywords are currently defined for this callable service.

### **rule\_array**

Direction: Input

Type: String

The *rule\_array* contains keywords that provide control information to the callable service. There are no *rule\_array* keywords currently defined for this callable service.

### **opt\_blocks\_bfr\_length**

Direction: Input

Type: Integer

This parameter specifies the length of the buffer passed with the *opt\_blocks* parameter. This length is used to determine if it would overflow the buffer size when adding a new optional block to the current contents of the buffer.

### **opt\_blocks\_length**

Direction: Input/Output

Type: Integer

This parameter specifies the actual length of the set of optional blocks currently contained in the *opt\_blocks* buffer. On output, it is updated with the length after the callable service has added the new optional block.

### **opt\_blocks**

Direction: Input/Output

Type: String

This parameter specifies a buffer containing the set of optional blocks being built. In the first call, it will generally be empty. The callable service will append one optional block to the buffer with each call. Parameter *opt\_blocks\_bfr\_length* specifies the total length of this buffer, and an error will be returned if this length would be exceeded by adding the optional block in parameter *opt\_block\_data* to the current contents. This parameter is encoded in ASCII on both input and output.

### **num\_opt\_blocks**

Direction: Output

Type: Integer

This parameter contains the number of optional blocks contained in the structure returned in parameter *opt\_blocks*. This is provided as an output parameter so that it can subsequently be used as an input to the TR-31 Export callable service.

### **opt\_block\_id**

Direction: Input

Type: String

This parameter specifies a two-byte value which is the identifier (ID) of the optional block passed in parameter *opt\_block\_data*.

**opt\_block\_data\_length**

Direction: Input

Type: Integer

This parameter specifies the length of the data passed in parameter *opt\_block\_data*. Note that it is valid for this length to be zero; an optional block can have an ID and a length, but no data.

**opt\_block\_data**

Direction: Input

Type: String

This parameter specifies a buffer where the application passes the data for the optional block that is to be added to those already in the buffer in parameter *opt\_blocks*. The length of this data is specified in parameter *opt\_block\_data\_length*.

**Restrictions**

None.

**Usage Notes**

Unless otherwise noted, all String parameters that are either written to, or read from, a TR-31 key block will be in EBCDIC format. Input parameters are converted to ASCII before being written to the TR-31 key block and output parameters are converted to EBCDIC before being returned (see Appendix G, “EBCDIC and ASCII Default Conversion Tables,” on page 959). TR-31 key blocks themselves are always in printable ASCII format as required by the ANSI TR-31 specification.

Note that the Padding Block, ID “PB” is not allowed to be added by the user. A Padding Block of the appropriate size, if needed, will be added when building the TR-31 key block in TR-31 Export. If the TR-31 Export callable service encounters a padding block in the optional block data, an error will occur.

**Required Hardware**

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 125. TR-31 Optional Data Build required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None	
IBM @server zSeries 990	None	
IBM @server zSeries 890		
IBM System z9 EC	None	
IBM System z9 BC		

## TR-31 Optional Data Build

Table 125. TR-31 Optional Data Build required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC	None	
IBM System z10 BC		
z196	None	
z114		
IBM zEnterprise EC12	None	

## TR-31 Optional Data Read (CSNBT31R and CSNET31R)

A TR-31 key block can hold optional fields which are securely bound to the key block using the integrated MAC. The optional blocks may either contain information defined in the TR-31 standard, or they may contain proprietary data. A separate range of optional block identifiers is reserved for use with proprietary blocks.

Note that some of the parameters are only used with keyword INFO and others are only used with keyword DATA.

The callable service name for AMODE(64) is CSNET31R.

### Format

```
CSNBT31R(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    TR31_key_block_length,  
    TR31_key_block,  
    opt_block_id,  
    num_opt_blocks,  
    opt_block_ids,  
    opt_block_lengths,  
    opt_block_data_length,  
    opt_block_data )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

**exit\_data**

Direction: Ignored

Type: String

This field is ignored.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The *rule\_array\_count* parameter must be 1

**rule\_array**

Direction: Input

Type: String

The *rule\_array* contains keywords that provide control information to the callable service. The keywords are 8 bytes in length and must be left-aligned and padded on the right with space characters. The *rule\_array* keywords for this callable service are shown in the following table.

*Table 126. Keywords for TR-31 Optional Data Read Rule Array Control Information*

Keyword	Meaning
<i>Operation – one required</i>	
INFO	Return information about the optional blocks in the TR-31 key block.
DATA	Return the data contained in a specified optional block in the TR-31 key block.

**TR31\_key\_block\_length**

Direction: Input

Type: Integer

This parameter specifies the length of the *TR31\_key\_block* parameter, in bytes. The parameter may specify a length that is greater than the size of the key block however it can never be greater than the size of the buffer where the key block resides. This value must be between 16 and 9992 inclusive.

**TR31\_key\_block**

Direction: Input

Type: String

This parameter contains the TR-31 key block that is to be parsed. The length of the TR-31 block is specified using parameter *TR31\_key\_block\_length*.

**opt\_block\_id**



## TR-31 Optional Data Read

Direction: Input

Type: String

This parameter is only used with option DATA. It is ignored for others. It specifies a 2-byte string which contains the identifier of the block from which the application is requesting data. The callable service will locate this optional block within the TR-31 structure and copy the data from that optional block into the returned *opt\_block\_data* buffer. If the specified optional block is not found in the TR-31 key block, an error will occur.

### **num\_opt\_blocks**

Direction: Input

Type: Integer

This parameter specifies the number of optional blocks in the TR-31 key block. The value is compared to the corresponding value in the TR-31 block header and if they do not match the callable service fails with an error. This parameter is only used for option INFO and is not examined for any other options.

### **opt\_block\_ids**

Direction: Output

Type: String Array

This parameter contains an array of two-byte string values. Each of these values is the identifier (ID) of one of the optional blocks contained in the TR-31 key block. The callable service returns a list containing the ID of each optional block that is in the TR-31 block, and the list is in the order that the optional blocks appear in the TR-31 header. The total length of the returned list will be two times the number of optional blocks, and the caller must supply a buffer with a length at least twice the value it passes in parameter *num\_opt\_blocks*. This parameter is only used for option INFO and is not examined for any other options.

### **opt\_block\_lengths**

Direction: Output

Type: Array

This parameter contains an array of 16-bit integer values. Each of these values is the length in bytes of one of the optional blocks contained in the TR-31 key block. The callable service returns a list containing the length of each optional block that is in the TR-31 block, and the list is in the order that the optional blocks appear in the TR-31 header. The total length of the returned list will be four times the number of optional blocks and the application program must supply a buffer with a length at least four times the value it passes in parameter *num\_opt\_blocks*. This parameter is only used for option INFO and is not examined or altered for any other options.

### **opt\_block\_data\_length**

Direction: Input/Output

Type: Integer

This parameter specifies the length for parameter *opt\_block\_data*. On input it must be set to the length of the buffer provided by the application program, and on output it is updated to contain the length of the returned optional block data, in bytes. It is only used for option DATA.

### **opt\_block\_data**

Direction: Output

Type: String

This parameter contains a buffer where the callable service stores the data it reads from the specified optional block. The buffer must have enough space for the data, as indicated by the input value of parameter *opt\_block\_data\_length*. If not an error occurs and no changes are made to the contents of the buffer. If the size of the buffer is sufficient, the data is copied to the buffer and its length

is stored in parameter `opt_block_data_length`. It is only used for option DATA and is not examined or altered for any other options.

## Restrictions

None

## Usage Notes

Unless otherwise noted, all String parameters that are either written to, or read from, a TR-31 key block will be in EBCDIC format. Input parameters are converted to ASCII before being written to the TR-31 key block and output parameters are converted to EBCDIC before being returned (see Appendix G, “EBCDIC and ASCII Default Conversion Tables,” on page 959). TR-31 key blocks themselves are always in printable ASCII format as required by the ANSI TR-31 specification.

The TR-31 Optional Data Read callable service (CSNBT31R and CSNET31R) can be used in conjunction with the TR-31 Parse callable service (CSNBT31P and CSNET31P) to obtain both the standard header fields and any optional data blocks from the key block. This is generally a three-step process.

1. Use the TR-31 Parse callable service to determine how many optional blocks are in the TR-31 token. This is returned in the `num_opt_blocks` parameter.
2. Use keyword INFO with the TR-31 Optional Data Read callable service to obtain lists of the optional block identifiers and optional block lengths. Your buffers must be large enough to hold the returned data, but the required size can be determined from the number of blocks obtained in the step above.
3. Use keyword DATA with the TR-31 Optional Data Read callable service to obtain the data for a particular optional block, specified by the block identifier.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 127. TR-31 Optional Data Read required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None	
IBM @server zSeries 990	None	
IBM @server zSeries 890		
IBM System z9 EC	None	
IBM System z9 BC		
IBM System z10 EC	None	
IBM System z10 BC		
z196	None	
z114		
IBM zEnterprise EC12	None	

## TR-31 Parse (CSNBT31P and CSNET31P)

Use the TR-31 Parse callable service to retrieve standard header information from a TR-31 key block without importing the key.

The callable service name for AMODE(64) is CSNET31P.

### Format

```
CALL CSNBT31P(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    TR31_key_block_length,
    TR31_key_block,
    key_block_version,
    key_block_length,
    key_usage,
    algorithm,
    mode,
    key_version_number,
    exportability,
    num_opt_blocks )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

#### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

#### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The *rule\_array\_count* parameter must be 0 because no keywords are currently defined for this callable service.

**rule\_array**

Direction: Input

Type: String

A *rule\_array* contains keywords that provide control information to the callable service. No rule array keywords are currently defined for this callable service.

### TR31 key block length

Direction: Input

Type: Integer

This parameter specifies the length of the *TR31\_key\_block* parameter, in bytes. The parameter may specify a length that is greater than the size of the key block (however it can never be greater than the size of the buffer where the key block resides). This value must be between 16 and 9992 inclusive.

## TR31\_key\_block

Direction: Input

Type: String

This parameter contains the TR-31 key block that is to be parsed.

**key\_block\_version**

Direction: Output

Type: String

This parameter contains a one-byte character value that indicates the version of the TR-31 key block, parsed from the block itself. CCA only supports versions "A", "B", and "C" key blocks.

**key\_block\_length**

Direction: Output

Type: Integer

This parameter contains the length of the key block as obtained from the TR-31 key block header. Note that this may be different from the input value in parameter *TR31\_key\_block\_length*, if the application program specifies a length that is greater than the actual length of the key block.

## key usage

Direction: Output

Type: String

This parameter contains a 2-byte string value indicating the TR-31 key usage value for the key contained in the block. The value is obtained from the TR-31 key block header. The usage defines the type of function this key can be used with, such as data encryption, PIN encryption, or key wrapping.

## algorithm

Direction: Output

Type: String

## TR-31 Parse

This parameter contains a one-byte string identifying the cryptographic algorithm the wrapped key is to be used with. The value is read from the TR-31 key block header. CCA only supports "D" for a Single-DES key and "T" for a Triple-DES key.

### **mode**

Direction: Output

Type: String

This parameter contains a one-byte string indicating the TR-31 mode of use for the key contained in the block. The value is obtained from the TR-31 key block header. The mode of use describes what operations the key can perform, within the limitations specified with the key usage value. For example, a key with usage for data encryption can have a mode to indicate it may be used for encryption only, decryption only, or both encryption and decryption.

### **key\_version\_number**

Direction: Output

Type: String

This parameter contains a two-byte string obtained from the TR-31 key block header which represents versioning information about the key contained in the block.

### **exportability**

Direction: Output

Type: String

This parameter contains a one-byte string indicating the key exportability value from the TR-31 key block header. This value indicates whether the key can be exported from this system, and if so it specifies conditions under which export is permitted.

### **num\_opt\_blocks**

Direction: Output

Type: Integer

This parameter contains the number of optional blocks that are part of the TR-31 key block.

## Restrictions

None

## Usage Notes

Unless otherwise noted, all String parameters that are either written to, or read from, a TR-31 key block will be in EBCDIC format. Input parameters are converted to ASCII before being written to the TR-31 key block and output parameters are converted to EBCDIC before being returned (see Appendix G, "EBCDIC and ASCII Default Conversion Tables," on page 959). TR-31 key blocks themselves are always in printable ASCII format as required by the ANSI TR-31 specification.

The TR-31 Optional Data Read callable service (CSNBT31R and CSNET31R) can be used in conjunction with the TR-31 Parse callable service (CSNBT31P and CSNET31P) to obtain both the standard header fields and any optional data blocks from the key block. This is generally a three-step process.

1. Use the TR-31 Parse callable service to determine how many optional blocks are in the TR-31 token. This is returned in the num\_opt\_blocks parameter.
2. Use keyword INFO with the TR-31 Optional Data Read callable service to obtain lists of the optional block identifiers and optional block lengths. Your buffers must be large enough to hold the returned data, but the required size can be determined from the number of blocks obtained in the step above.
3. Use keyword DATA with the TR-31 Optional Data Read callable service to obtain the data for a particular optional block, specified by the block identifier.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 128. TR-31 Parse required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None	
IBM @server zSeries 990	None	
IBM @server zSeries 890		
IBM System z9 EC	None	
IBM System z9 BC		
IBM System z10 EC	None	
IBM System z10 BC		
z196	None	
z114		
IBM zEnterprise EC12	None	

## Unique Key Derive (CSNBUKD and CSNEUKD)

Unique Key Derive (CSNBUKD and CSNEUKD) will perform the key derivation process as defined in ANSI X9.24 Part 1.

The process derives keys from two values — The base derivation key and the derivation data:

- The base derivation key is the key from which the others are derived. This must be a KEYGENKY with the UKPT bit (bit 18) set to 1 in the Control Vector.
- The derivation data is used to make the derived key specific to a particular device and to a specific transaction from that device. The derivation data, called the Current Key Serial Number (CKSN), is the 80-bit concatenation of the device's 59-bit Initial Key Serial Number value and the 21-bit value of the current encryption counter which the device increments for each new transaction.

Rule array keywords determine the types and number of keys derived on a particular call. See the Rule Array parameter description for more information.

## Unique Key Derive

Output keys are wrapped using the mode configured as the default wrapping mode, either enhanced wrapping mode (WRAP-ENH) or legacy ECB wrapping mode (WRAP-ECB).

## Format

```
CALL CSNBKUD(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    base_derivation_key_identifier_length,  
    base_derivation_key_identifier,  
    derivation_data_length,  
    derivation_data,  
    generated_key_identifier1_length,  
    generated_key_identifier1,  
    generated_key_identifier2_length,  
    generated_key_identifier2,  
    generated_key_identifier3_length,  
    generated_key_identifier3,  
    reserved1_length,  
    reserved1,  
    reserved2_length,  
    reserved2,  
    reserved3_length,  
    reserved3,  
    reserved4_length,  
    reserved4,  
    reserved5_length,  
    reserved5,  
    reserved6_length,  
    reserved6)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'FFFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

**exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the rule\_array parameter. Values are 1 through 5.

**rule\_array**

Direction: Input

Type: String

The *rule\_array* parameter is an array of keywords. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The *rule\_array* keywords are:

*Table 129. Keywords for Unique Key Derive*

Keyword	Meaning
<i>Algorithm (One, optional. The default is DES)</i>	
DES	Specifies that the keys to be generated are DES (Triple DES) keys. All input skeleton tokens must be DES tokens and all generated output tokens will be DES tokens.
<i>Key wrapping method (One, optional. The default is USECONFIG. The Access Control Point Unique Key Derive – Override Default Wrapping Method must be enabled to specify these keywords.</i>	
USECONFIG	Specifies to wrap the key using the configuration setting for the default wrapping method.
WRAP-ECB	Specifies to wrap the key using the legacy wrapping method.
WRAP-ENH	Specifies to wrap the key using the enhanced wrapping method.
<i>Output Key Selection Keywords. (One required, up to 3 can be specified.)</i> The PIN-DATA keyword cannot be specified with any other Output Key Selection keywords. Any combination of the other keywords (K1DATA, K2MAC, and K3PIN) can be specified, enabling a program to produce up to 3 different output keys with one call.	



## Unique Key Derive

Table 129. Keywords for Unique Key Derive (continued)

Keyword	Meaning
PIN-DATA	<p>The returned key type for this keyword is a PIN key, which is returned in a DATA key token. This is an output key selection keyword for the <i>generated_key_identifier3_length</i> and <i>generated_key_identifier3</i> parameters.</p> <p>Output value <i>generated_key_identifier3</i> will be created and will be a DATA key. The skeleton token provided in that parameter on input must be one of the permitted "PIN key with rule keyword PIN-DATA" key types for this callable service. For valid values, see Table 130 on page 345.</p> <p>To use this option:</p> <ul style="list-style-type: none"> <li>Control Vector bit 61 (Not-CCA) will be set to a one.</li> <li>Access Control Point Unique Key Derive – Allow PIN-DATA processing must be enabled.</li> </ul>
K1DATA	<p>The returned key type for this keyword is a DATA ENCRYPTION key. This is the output key selection keyword for the <i>generated_key_identifier1_length</i> and <i>generated_key_identifier1</i> parameters.</p> <p>Output value <i>generated_key_identifier1</i> will be created and will be a data encryption key. The skeleton token provided in that parameter on input must be one of the permitted "Data encryption key" types for this callable service. For valid values see Table 130 on page 345.</p>
K2MAC	<p>The returned key type for this keyword is a MAC key. This is the output key selection keyword for the <i>generated_key_identifier2_length</i> and <i>generated_key_identifier2</i> parameters.</p> <p>Output value <i>generated_key_identifier2</i> will be created and will be a MAC key. The skeleton token provided in that parameter on input must be one of the permitted MAC key types for this callable service. For valid values, see Table 130 on page 345.</p>
K3PIN	<p>The returned key type for this keyword is a PIN key. This is an output key selection keyword for the <i>generated_key_identifier3_length</i> and <i>generated_key_identifier3</i> parameters.</p> <p>Output value <i>generated_key_identifier3</i> will be created and will be a PIN key. The skeleton token provided in that parameter on input must be one of the permitted PIN key types for this callable service. For valid values see Table 130 on page 345.</p>

### base\_derivation\_key\_identifier\_length

Direction: Input

Type: Integer

The length of the base\_derivation\_key parameter. This value must be 64

### base\_derivation\_key\_identifier

Direction: Input/Output

Type: String

The base derivation key is the key from which the operational keys are derived using the DUKPT algorithms defined in ANSI X9.24 Part 1. The base derivation key must be an internal key token or the label of an internal key token containing a double-length KEYGENKY key with the UKPT bit (bit 18) set to 1 in the Control Vector.

**derivation\_data\_length**

Direction: Input	Type: Integer
------------------	---------------

The length of the *derivation\_data* parameter. This value must be 10.

**derivation\_data**

Direction: Input	Type: String
------------------	--------------

The derivation data is an 80-bit (10-byte) string that contains the Current Key Serial Number (CKSN) of the device concatenated with the 21-bit value of the current Encryption Counter which the device increments for each new transaction.

**generated\_key\_identifier1\_length**

Direction: Input/Output	Type: Integer
-------------------------	---------------

The length of the *generated\_key\_identifier1* parameter. Values are 0 and 64.

**generated\_key\_identifier1**

Direction: Input/Output      Type: String

On input, this must be a DES Data encryption key token or a skeleton token of a DES Data encryption key, with one of the Data encryption control vectors as shown in Table 130 on page 345.

On output, *generated\_key\_identifier1* will contain the data encryption token with the derived data encryption key.

**generated\_key\_identifier2\_length**

Direction: Input/Output	Type: Integer
-------------------------	---------------

The length of the *generated\_key\_identifier2* parameter. Values are 0 and 64.

**generated\_key\_identifier2**

Direction: Input/Output      Type: String

On input, this must be a DES MAC key token or a skeleton token of a DES MAC key, with one of the MAC control vectors as shown in Table 130 on page 345. On output, *generated\_key\_identifier2* will contain the MAC token with the derived MAC key.

**generated\_key\_identifier3\_length**

Direction: Input/Output	Type: Integer
-------------------------	---------------

The length of the *generated\_key\_identifier3* parameter. Values are 0 and 64.

generation key identifier3

Direction: Input/Output Type: String

The input and output values for this parameter depends on the keyword specified in the *rule\_array* parameter. The *rule\_array* keyword for the

## Unique Key Derive

*generation\_key\_identifier3* parameter can be either PIN-DATA or K3PIN.

- When Rule Array Keyword is PIN-DATA, input must be a Data key token or skeleton token of a Data key with one of the “PIN key with rule keyword PIN-DATA” control vectors as shown in Table 130 on page 345. On output, this parameter will contain the Data token with the derived PIN key.
- When Rule Array Keyword is K3PIN, input must be a DES PIN key token or a skeleton token of a DES PIN key, with one of the PIN control vectors as shown in Table 130 on page 345. On output, this parameter will contain the PIN token with the derived PIN key.

**reserved1\_length**

Direction: Input

Type: Integer

This parameter must be zero.

**reserved1**

Direction: Ignored

Type: String

This parameter is ignored.

**reserved2** 1 length

Direction: Input

Type: Integer

This parameter must be zero.

## reserved2

Direction: Ignored

Type: String

This parameter is ignored.

**reserved3\_length**

Direction: Input

Type: Integer

This parameter must be zero.

**reserved3**

Direction: Ignored

Type: String

This parameter is ignored.

**reserved4\_length**

Direction: Input

Type: Integer

This parameter must be zero.

**reserved4**

Direction: Ignored

Type: String

This parameter is ignored.

**reserved5** 1 length

Direction: Input

Type: Integer

This parameter must be zero.

## reserved5

Direction: Ignored

Type: String

This parameter is ignored.

**reserved6\_length**

Direction: Input

Type: Integer

This parameter must be zero.

**reserved6**

Direction: Ignored

Type: String

This parameter is ignored.

## Restrictions

The following table shows the valid skeleton tokens depending on the key type to be derived.

Table 130. Valid Control Vectors for Derived Keys

Key to be derived	Supported key types in the skeleton token		
Data encryption key	CIPHER	00 03 71 00 03 41 00 00	00 03 71 00 03 21 00 00
	ENCIPHER	00 03 60 00 03 41 00 00	00 03 60 00 03 21 00 00
	DECIPHER	00 03 50 00 03 41 00 00	00 03 50 00 03 21 00 00
Message authentication (MAC) key	MAC	00 05 4D 00 03 41 00 00	00 05 4D 00 03 21 00 00
	MACVER	00 05 44 00 03 41 00 00	00 05 44 00 03 21 00 00
PIN key	IPINENC	00 21 5F 00 03 41 00 00	00 21 5F 00 03 21 00 00
	OPINENC	00 24 77 00 03 41 00 00	00 24 77 00 03 21 00 00
PIN key with rule keyword PIN-DATA	DATA PIN	00 00 7D 00 03 41 00 00	00 00 7D 00 03 21 00 00

Note that the following bits of the control vector are not checked and may have a value of either 0 or 1.

- Bit 17 - Export control
- Bit 56 – Enhanced wrapping control
- Bit 57 – TR-31 export control
- Bits 4 and 5 – UDX

Additional control vector bit that is not checked for PIN key with rule keyword PIN-DATA.

- Bit 61 - Not-CCA

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal key tokens that are stored in the CKDS.

The following table indicates the variants used for each output key type to be derived.

## Unique Key Derive

Table 131. Derivation Variants

Key Type	DUKPT derivation variant	DUKPT key usage description
IPINENC OPINENC PIN key (using PIN-DATA rule array keyword)	00000000000000FF 00000000000000FF	PIN Encryption
MAC	000000000000FF00 000000000000FF00	MAC, request or both ways
MACVER	00000000FF000000 00000000FF000000	MAC, response only
CIPHER ENCIPHER	0000000000FF0000 0000000000FF0000	Data Encryption, request or both ways
DECIPHER	000000FF00000000 000000FF00000000	Data Encryption, response only

## Access Control Points

The Unique Key Derive access control point controls the function of this service. Specifying a “Key wrapping method” in the rule array requires the **Unique Key Derive – Override Default Wrapping Method** access control point to be enabled in the active role.

Specifying the PIN-DATA rule array keyword requires the **Unique Key Derive – Allow PIN-DATA** access control point to be enabled in the active role.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 132. Unique Key Derive required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This callable service is not supported.
IBM @server zSeries 990		This callable service is not supported.
IBM @server zSeries 890		
IBM System z9 EC		This callable service is not supported.
IBM System z9 BC		
IBM System z10 EC		This callable service is not supported.
IBM System z10 BC		
z196		This callable service is not supported.
z114		

Table 132. Unique Key Derive required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## User Derived Key (CSFUDK and CSFUDK6)

This callable service is not supported on an IBM @server zSeries 990, IBM @server zSeries 890, z9 EC and z9 BC, z10 EC and z10 BC. Diversified key generate callable service can be used to perform this processing.

Use the user derived key callable service to generate a single-length or double-length MAC key or to update an existing user derived key. A single-length MAC key can be used to compute a MAC following the ANSI X9.9, ANSI X9.19, or the Europay, MasterCard and VISA (EMV) Specification MAC processing rules. A double-length MAC key can be used to compute a MAC following either the ANSI X9.19 optional double MAC processing rule or the EMV Specification MAC processing rule.

This service updates an existing user derived key by XORing it with data you supply in the *data\_array* parameter. This is called SESSION MAC key generation by VISA.

This service adjusts the user derived key or SESSION MAC key to odd parity. The parity of the supplied derivation key is not tested.

The callable service name for AMODE(64) invocation is CSFUDK6.

## Format

```
CALL CSFUDK(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_type,
    rule_array_count,
    rule_array,
    derivation_key_identifier,
    source_key_identifier,
    data_array,
    generated_key_identifier)
```

## Parameters

### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

## User Derived Key

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **key\_type**

Direction: Input

Type: String

The 8-byte keyword of 'MAC ' or 'MACD ' that specifies the key type to be generated. The keyword must be left-justified and padded on the right with blanks. MAC specifies an 8-byte, single-length MAC key which is used in the ANSI X9.9-1 or the ANSI X9.19 basic MAC processing rules. MACD specifies a 16-byte, double-length internal MAC key that uses the single-length control vector for both the left and right half of the key (MAC || MAC). The double-length MAC key is used in the ANSI X9.19 optional double-key MAC processing rules. The keyword 'TOKEN ' is also accepted. If you specify TOKEN with a *rule\_array* of VISA or NOFORMAT, the key type is determined by the valid internal token of the single-length or double-length MAC key in the *generated\_key\_identifier* parameter. If you specify TOKEN with a *rule\_array* of SESS-MAC, the key type is determined by the valid internal token of the single-length or double-length MAC key in the *source\_key\_identifier*.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords specified in the *rule\_array* parameter. The value must be 1.

### **rule\_array**

Direction: Input

Type: Character string

The process rule for the user derived key in an 8-byte field. The keywords must be in 8 bytes of contiguous storage, left-justified and padded on the right with blanks.

The keywords are shown in Table 133 on page 349.

Table 133. Keywords for User Derived Key Control Information

Keyword	Meaning
<b>User Derived Key Process Rules (required)</b>	
NOFORMAT	For generating a user derived key with no formatting done on the array before encryption under the <i>derivation_key_identifier</i> .
SESS-MAC	To update an existing user derived key supplied in the <i>source_key_identifier</i> parameter with data provided in the <i>data_array</i> parameter.
VISA	For generating a user derived key using the VISA algorithm to format the data array input before encryption under the <i>derivation_key_identifier</i> . For guidance information refer to the VISA Integrated Circuit Card Specification, V1.3 Aug 31, 1996.

**derivation\_key\_identifier**

Direction: Input/Output

Type: String

For a *rule\_array* value of VISA or NOFORMAT, this is a 64-byte key label or internal key token of the derivation key used to generate the user derived key. The key must be an EXPORTER key type. For any other keyword, this field must be a null token.

**source\_key\_identifier**

Direction: Input/Output

Type: String

For a *rule\_array* value of SESS-MAC, this is a 64-byte internal token of a single-length or double-length MAC key. For any other keyword, this field must be a null token.

**data\_array**

Direction: Input

Type: String

Two 16-byte data elements required by the corresponding *rule\_array* and *key\_type* parameters. The data array consists of two 16-byte hexadecimal character fields whose specification depends on the process rule and key type. VISA requires only one 16-byte hexadecimal character input. Both NOFORMAT and SESS-MAC require one 16-byte input for a key type of MAC and two 16-byte inputs for a key type of MACD. If only one 16-byte field is required, then the rest of the data array is ignored by the callable service.

**generated\_key\_identifier**

Direction: Input/Output

Type: String

The 64-byte internal token of the generated single-length or double-length MAC key. This is an input field only if TOKEN is specified for *key\_type*.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.



## User Derived Key

This service requires that the ANSI system keys be installed in the CKDS.

### Required Hardware

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 134. User derived key required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990 IBM @server zSeries 890		This callable service is not supported.
IBM System z9 EC and z9 BC		This callable service is not supported.
IBM System z10 EC and z10 BC		This callable service is not supported.
z196 z114		This callable service is not supported.
IBM zEnterprise EC12		This callable service is not supported.

---

## Chapter 6. Protecting Data

Use ICSF to protect sensitive data stored on your system, sent between systems, or stored off your system on magnetic tape. To protect data, encipher it under a key. When you want to read the data, decipher it from ciphertext to plaintext form.

ICSF provides *encipher* and *decipher callable services* to perform these functions. If you use a key to encipher data, you must use the same key to decipher the data. To use clear keys directly, ICSF provides *symmetric key decipher*, *symmetric key encipher*, *encode* and *decode callable services*. These services encipher and decipher with clear keys. You can use clear keys indirectly by first using the clear key import callable service, and then using the encipher and decipher callable services.

This topic describes these services:

- “Ciphertext Translate (CSNBCTT or CSNBCTT1 and CSNECTT or CSNECTT1)” on page 354
- “Ciphertext Translate2 (CSNBCTT2, CSNBCTT3, CSNECTT2, CSNECTT3)” on page 357
- “Decipher (CSNBDEC or CSNBDEC1 and CSNEDEC or CSNEDEC1)” on page 370
- “Decode (CSNBDCO and CSNEDCO)” on page 377
- “Encipher (CSNBENC or CSNBENC1 and CSNEENC or CSNEENC1)” on page 379
- “Encode (CSNBECO and CSNEECO)” on page 387
- “Symmetric Algorithm Decipher (CSNBSAD or CSNBSAD1 and CSNESAD or CSNESAD1)” on page 389
- “Symmetric Algorithm Encipher (CSNBSAE or CSNBSAE1 and CSNESAE or CSNESAE1)” on page 396
- “Symmetric Key Decipher (CSNBSYD or CSNBSYD1 and CSNESYD or CSNESYD1)” on page 402
- “Symmetric Key Encipher (CSNBSYE or CSNBSYE1 and CSNESYE or CSNESYE1)” on page 412

---

### Modes of Operation

To encipher or decipher data or keys, ICSF uses either the U.S. National Institute of Standards and Technology (NIST) Data Encryption Standard (DES) algorithm or the Commercial Data Masking Facility (CDMF). The DES algorithm is documented in *Federal Information Processing Standard #46*. CDMF provides DES cryptography using an effectively shortened DATA key. See “System Encryption Algorithm” on page 52 for more information.

To encipher or decipher data, ICSF also uses the U.S. National Institute of Standards and Technology (NIST) Advanced Encryption Standard (AES) algorithm. The AES algorithm is documented in *Federal Information Processing Standard 197*.

ICSF enciphers and deciphers using several modes of operation. Some of the modes have variations related to padding or blocking of the data. The text in parentheses is the processing rule associated with that mode.

The supported modes are:

- Electronic code book (ECB)

- Cipher block chaining (CBC)
  - Cipher block chaining with ciphertext stealing (CBC-CS)
  - Cipher block chaining compatible with CUSP/PCF (CUSP)
  - Cipher block chaining compatible with IPS (IPS)
  - Cipher block chaining using PKCS#7 padding (PKCS-PAD)
  - Cipher block chaining using ANSI X9.23 padding (X9.23)
  - Cipher block chaining using IBM 4700 padding (4700-PAD)
- Cipher Feedback (CFB)
  - Cipher Feedback with a non-blocksize segment (CFB-LCFB)
- Output Feedback (OFB)
- Galois/Counter Mode (GCM)

## Electronic Code Book (ECB) Mode

In the ECB mode, each block of plaintext is separately enciphered and each block of the ciphertext is separately deciphered. In other words, the encipherment or decipherment of a block is totally independent of other blocks. ICSF uses the ECB encipherment mode for enciphering and deciphering data with clear keys using the encode and decode callable services.

ICSF does not support ECB encipherment mode on CDMF-only systems.

## Cipher Block Chaining (CBC) Mode

The CBC mode uses an initial chaining vector (ICV) in its processing. The CBC mode only processes blocks of data in exact multiples of the blocksize. The ICV is exclusive ORed with the first block of plaintext prior to the encryption step; the block of ciphertext just produced is exclusive-ORed with the next block of plaintext, and so on. You must use the same ICV to decipher the data. This disguises any pattern that may exist in the plaintext. CBC mode is the default for encrypting and decrypting data using the Encipher and Decipher callable services. “Cipher Processing Rules” on page 940 describes the CBC-specific processing rules in detail.

## Cipher Feedback (CFB) Mode

The CFB mode uses an initial chaining vector (ICV) in its processing. CFB mode performs cipher feedback encryption. CFB mode operates on segments instead of blocks. The segment length (called *s*) is between one bit and the block size (called *b*) for the underlying algorithm (DES or AES), inclusive. ICSF only allows segment sizes which are a multiple of eight bits (complete bytes). Each encryption step takes an input block, enciphers it with the key provided to generate an output block, takes the most significant *s* bits of the output block, and then exclusive ORs that with the plaintext segment. The first input block is the ICV and each subsequent input block is formed by concatenating the (*b-s*) least significant bits of the previous input block and the ciphertext (*s* bits) from the previous step to form a full block. The input text can be of any length. The output text will have the same length as the input text.

## Output Feedback (OFB) Mode

The OFB mode uses an initial chaining vector (ICV) in its processing. OFB mode requires that the ICV is a nonce (the ICV must be unique for each execution of the mode under the given key). Each encryption step takes an input block, enciphers it with the key provided to generate an output block, and then exclusive ORs the

output block with the plaintext block. The first input block is the ICV and each subsequent input block is the previous output block. The input text can be of any length. The output text will have the same length as the input text.

## Galois/Counter Mode (GCM)

The GCM mode uses an initialization vector (IV) in its processing. This mode is used for authenticated encryption with associated data. GCM provides confidentiality and authenticity for the encrypted data and authenticity for the additional authenticated data (AAD). The AAD is not encrypted. GCM mode requires that the IV is a nonce, i.e., the IV must be unique for each execution of the mode under the given key. The steps for GCM encryption are:

1. The hash subkey for the GHASH function is generated by applying the block cipher to the “zero” block.
  2. The pre-counter block ( $J_0$ ) is generated from the IV. In particular, when the length of the IV is 96 bits, then the padding string  $0^{31} || 1$  is appended to the IV to form the pre-counter block. Otherwise, the IV is padded with the minimum number of ‘0’ bits, possibly none, so that the length of the resulting string is a multiple of 128 bits (the block size); this string in turn is appended with 64 additional ‘0’ bits, followed by the 64-bit representation of the length of the IV, and the GHASH function is applied to the resulting string to form the pre-counter block.
  3. The 32-bit incrementing function is applied to the pre-counter block to produce the initial counter block for an invocation of the GCTR function on the plaintext. **The output of this invocation of the GCTR function is the ciphertext.**
  4. The AAD and the ciphertext are each appended with the minimum number of ‘0’ bits, possibly none, so that the bit lengths of the resulting strings are multiples of the block size. The concatenation of these strings is appended with the 64-bit representations of the lengths of the AAD and the ciphertext to produce block  $u$ .
  5. The GHASH function is applied to block  $u$  to produce a single output block.
  6. This output block is encrypted using the GCTR function with the pre-counter block that was generated in Step 2, and **the result is truncated to the specified tag length to form the authentication tag.**
  7. The ciphertext and the tag are returned as the output.
- The plaintext can be of any length. The ciphertext will have the same length as the plaintext.

For GCM decryption, the tag is an input parameter. ICSF calculates a tag using the same process as encryption and compares that to the parameter passed by the caller. If they match, the decryption will proceed.

## Triple DES Encryption

Triple-DES encryption uses a triple-length DATA key comprised of three 8-byte DES keys to encipher 8 bytes of data using this method:

- Encipher the data using the first key
- Decipher the result using the second key
- Encipher the second result using the third key

The procedure is reversed to decipher data that has been triple-DES enciphered:

- Decipher the data using the third key
- Encipher the result using the second key

- Decipher the second result using the first key

ICSF uses the triple-DES encryption in the CBC encipherment mode.

A variation of the triple DES algorithm supports the use of a double-length DATA key comprised of two 8-byte DATA keys. In this method, the first 8-byte key is reused in the last encipherment step.

Due to export regulations, triple-DES encryption may not be available on your processor.

---

## Ciphertext Translate (CSNBCTT or CSNBCTT1 and CSNECTT or CSNECTT1)

This callable service is only supported on the IBM @server zSeries 900.

ICSF provides a ciphertext translate callable service on DES-capable systems. The callable service decipheres encrypted data (ciphertext) under one data translation key and reenciphers it under another data translation key without having the data appear in the clear outside the Cryptographic Coprocessor Feature. ICSF uses the data translation key as either the input or the output data transport key. Such a function is useful in a multiple node network, where sensitive data is passed through multiple nodes prior to it reaching its final destination.

“Using the Ciphertext Translate and Ciphertext Translate2 Callable Service” on page 70 provides some tips on using the callable service.

Use the ciphertext translate callable service to decipher text under an “input” key and then to encipher the text under an “output” key. The callable service uses the cipher block chaining (CBC) mode of the DES. This service is available only on a DES-capable system.

### Choosing Between CSNBCTT and CSNBCTT1

CSNBCTT and CSNBCTT1 provide identical functions. When choosing the service to use, consider this:

- **CSNBCTT** requires the input text and output text to reside in the caller's primary address space. Also, a program using CSNBCTT adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

The callable service name for AMODE(64) invocation is CSNECTT.

- **CSNBCTT1** allows the input text and output text to reside either in the caller's primary address space or in a data space. This allows you to translate more data with one call. However, a program using CSNBCTT1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

The callable service name for AMODE(64) invocation is CSNECTT1.

For CSNBCTT1 and CSNECTT1, *text\_id\_in* and *text\_id\_out* are access list entry token (ALET) parameters of the data spaces containing the input text and output text.

## Format

```
CALL CSNBCTT(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier_in,
    key_identifier_out,
    text_length,
    text_in,
    initialization_vector_in,
    initialization_vector_out,
    text_out )
```

```
CALL CSNBCTT1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier_in,
    key_identifier_out,
    text_length,
    text_in,
    initialization_vector_in,
    initialization_vector_out,
    text_out,
    text_id_in,
    text_id_out )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

## Ciphertext Translate

The data that is passed to the installation exit.

### **key\_identifier\_in**

Direction: Input/Output

Type: String

The 64-byte string of the internal key token containing the data translation (DATAXLAT) key, or the label of the CKDS record containing the DATAXLAT key used to encipher the input string.

### **key\_identifier\_out**

Direction: Input/Output

Type: String

The 64-byte string of an internal key token containing the DATAXLAT key, or the label of the CKDS record containing the DATAXLAT key, used to reencipher the encrypted text.

### **text\_length**

Direction: Input

Type: Integer

The length of the ciphertext that is to be processed. The text length must be a multiple of 8 bytes. The maximum length of text is 2,147,836,647 bytes.

**Note:** The MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced.

### **text\_in**

Direction: Input

Type: String

The text that is to be translated. The text is enciphered under the data translation key specified in the *key\_identifier\_in* parameter.

### **initialization\_vector\_in**

Direction: Input

Type: String

The 8-byte initialization vector that is used to decipher the input data. This parameter is the initialization vector used at the previous cryptographic node.

### **initialization\_vector\_out**

Direction: Input

Type: String

The 8-byte initialization vector that is used to encipher the input data. This is the new initialization vector used when the callable service enciphers the plaintext.

### **text\_out**

Direction: Output

Type: String

The field where the callable service returns the translated text.

### **text\_id\_in**

Direction: Input

Type: Integer

For CSNBCTT1 only, the ALET of the text to be translated.

**text\_id\_out**

Direction: Input

Type: Integer

For CSNBCTT1 only, the ALET of the *text\_out* field that the application supplies.

## Restrictions

The input ciphertext length must be an exact multiple of 8 bytes. The minimum length of the ciphertext that can be translated is 8 bytes.

You cannot use this service on a CDMF-only system.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

The initialization vectors must have already been established between the communicating applications or must be passed with the data.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 135. Ciphertext translate required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990		This callable service is not supported.
IBM @server zSeries 890		
IBM System z9 EC		This callable service is not supported.
IBM System z9 BC		
IBM System z10 EC		This callable service is not supported.
IBM System z10 BC		
z196		This callable service is not supported.
z114		
IBM zEnterprise EC12		This callable service is not supported.

## Ciphertext Translate2 (CSNBCTT2, CSNBCTT3, CSNECTT2, CSNECTT3)

This callable service deciphers encrypted data (ciphertext) under one cipher text translation key and reenciphers it under another cipher text translation key without having the data appear in the clear outside the cryptographic coprocessor. ICSF uses the ciphertext translation key as either the input or the output data



## Ciphertext Translate2

transport key. Such a function is useful in a multiple node network, where sensitive data is passed through multiple nodes prior to it reaching its final destination.

“Using the Ciphertext Translate and Ciphertext Translate2 Callable Service” on page 70 provides some tips on using the callable service.

Use the ciphertext translate2 callable service to decipher text under an “input” key and then to encipher the text under an “output” key. Both AES and DES algorithms are supported. Translation between AES and DES is allowed with restrictions controlled by access control points.

The encryption modes supported are:

- DES – CBC, CUSP. and IPS
- AES – CBC and ECB

The padding methods supported are:

- DES – X9.23
- AES – PKCSPAD

## Choosing Between CSNBCTT2 and CSNBCTT3

CSNBCTT2 and CSNBCTT3 provide identical functions. When choosing the service to use, consider this:

- **CSNBCTT2** requires the input text and output text to reside in the caller's primary address space. Also, a program using CSNBCTT2 adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface. The callable service name for AMODE(64) invocation is CSNECTT2.
- **CSNBCTT3** allows the input text and output text to reside either in the caller's primary address space or in a data space. This allows you to translate more data with one call. However, a program using CSNBCTT3 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface. The callable service name for AMODE(64) invocation is CSNECTT3. For CSNBCTT3 and CSNECTT3, *text\_id\_in* and *text\_id\_out* are access list entry token (ALET) parameters of the data spaces containing the input text and output text.

## Format

```
CALL CSNBCTT2(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_in_length,
    key_identifier_in,
    initialization_vector_in_length,
    initialization_vector_in,
    cipher_text_in_length,
    cipher_text_in,
    chaining_vector_length,
    chaining_vector,
    key_identifier_out_length,
    key_identifier_out,
    initialization_vector_out_length,
    initialization_vector_out,
    cipher_text_out_length,
    cipher_text_out,
    reserved1_length,
    reserved1,
    reserved2_length,
    reserved2 )
```

```
CALL CSNBCTT3(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_in_length,
    key_identifier_in,
    initialization_vector_in_length,
    initialization_vector_in,
    cipher_text_in_length,
    cipher_text_in,
    chaining_vector_length,
    chaining_vector,
    key_identifier_out_length,
    key_identifier_out,
    initialization_vector_out_length,
    initialization_vector_out,
    cipher_text_out_length,
    cipher_text_out,
    reserved1_length,
    reserved1,
    reserved2_length,
    reserved2,
    text_id_in,
    text_id_out )
```

## Parameters

**return\_code**

Direction: Output

Type: Integer

## Ciphertext Translate2

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value must be 4 or 5.

### **rule\_array**

Direction: Input

Type: String

The keywords that provide control information to the callable service. The following table provides a list. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 136. Keywords for Ciphertext Translate2

Keyword	Meaning
<i>Inbound Processing Rule (One required)</i>	
I-CBC	Specifies encryption using CBC mode for the inbound ciphertext. The text length must be a multiple of the block size. The DES block size is 8 bytes. The AES block size is 16 bytes.
I-CUSP	Specifies that CBC with CUSP processing for the inbound ciphertext. The ciphertext may be any length. The ciphertext is the same length as the plaintext. This keyword is only valid with DES.
I-ECB	Specifies encryption using ECB mode for the inbound ciphertext. The text must be a multiple of the block size. This keyword is only valid for AES encryption.

Table 136. Keywords for Ciphertext Translate2 (continued)

Keyword	Meaning
I-IPS	Specifies that CBC with IPS processing has been used for the inbound ciphertext. The ciphertext may be any length. The ciphertext is the same length as the plaintext. This keyword is only valid with DES.
IPKCSPAD	Specifies that CBC with PKCS padding was used for the inbound ciphertext. The text was padded on the right with 1 - 16 bytes of pad characters, making the padded text a multiple of the AES block size, before the data was enciphered. Each pad character is valued to the number of pad characters added. This keyword is only valid for AES encryption.
I-X923	Specifies that CBC with X9.24 padding was used for the inbound ciphertext. This is compatible with the requirements in ANSI Standard X9.23. This keyword is only valid for DES encryption.
<b>Outbound Processing Rule (One required)</b>	
O-CBC	Specifies encryption in CBC mode will be used for the outbound ciphertext. The text length must be a multiple of the block size. The DES block size is 8 bytes. The AES block size is 16 bytes.
O-CUSP	Specifies that CBC with CUSP processing will be used for the outbound text. The outbound ciphertext will be the same length as the plaintext. This keyword is only valid with DES.
O-ECB	Specifies encryption using ECB mode will be used for the outbound ciphertext. The text must be a multiple of the block size. This keyword is only valid for AES encryption.
O-IPS	Specifies that CBC with IPS processing will be used for the outbound text. The outbound ciphertext will be the same length as the plaintext. This keyword is only valid with DES.
OPKCSPAD	Specifies that CBC with PKCS padding will be used for the outbound text. The outbound text will be padded on the right with 1 - 16 bytes of pad characters, making the padded text a multiple of the AES block size, before the data was enciphered. Each pad character is valued to the number of pad characters added. This keyword is only valid for AES encryption.
O-X923	Specifies that CBC with X9.24 padding will be used for the outbound text. This is compatible with the requirements in ANSI Standard X9.23. This keyword option is only valid for DES encryption.
<b>Segmenting Control (One optional)</b>	
CONTINUE	Specifies the initialization vectors are taken from the chaining vector. The chaining vector will be updated and must not be modified between calls. This keyword is ignored for I-ECB and O-ECB processing rules. The CONTINUE keyword is not valid with the I-X923 or O-X923 keywords.

Table 136. Keywords for Ciphertext Translate2 (continued)

Keyword	Meaning
INITIAL	Specifies the initialization vectors will be taken from the <i>initialization_vector_in</i> and <i>initialization_vector_out</i> parameters. This is the default. This keyword is ignored for I-ECB and O-ECB processing rules.
<b>Inbound Key Identifier (One Required)</b>	
IKEY-DES	Specifies that the inbound key identifier is a DES key.
IKEY-AES	Specifies that the inbound key identifier is an AES key.
<b>Outbound Key Identifier (One Required)</b>	
OKEY-DES	Specifies that the outbound key identifier is a DES key.
OKEY-AES	Specifies that the outbound key identifier is an AES key.

**key\_identifier\_in\_length**

Direction: Input

Type: Integer

Length of the *key\_identifier\_in* field in bytes. The value is 64 when a label is supplied. When the key identifier is a key token, the value is the length of the token. The maximum value is 725.

**key\_identifier\_in**

Direction: Input/Output

Type: String

An internal key token or the label of the CKDS record containing the cipher translation key for the inbound ciphertext.

Acceptable DES key types are DATA, CIPHER, CIPHERXI, CIPHERXL, and DECIPHER. The keys must have bit 19 for "DECIPHER" set on in the control vector. The key may be a single-, double-, or triple-length key. If the Cipher Text translate2 - Allow only cipher text translate types access control point is enabled, only CIPHERXI and CIPHERXL are allowed.

Acceptable AES key types include the 64-byte AES DATA key and the variable length token CIPHER key with the DECRYPT bit on in the key usage field. The C-XLATE bit can optionally be on. If the Cipher Text translate2 - Allow only cipher text translate types access control point is enabled, the C-XLATE bit must be turned on in the key usage field.

**initialization\_vector\_in\_length**

Direction: Input

Type: Integer

Length of the *initialization\_vector\_in* field in bytes. For AES keys, the length is 16. For DES keys, the length is 8. When the initialization vector is not required (segmenting rule CONTINUE, processing rule I-ECB), the value must be 0.

**initialization\_vector\_in**

Direction: Input

Type: String

The initialization vector that is used to decipher the input data. This parameter is the initialization vector used at the previous cryptographic node. This parameter is required for segmenting rule INITIAL.

**ciphertext\_in\_length**

Direction: Input

Type: Integer

The length of the ciphertext to be processed. See the table of ciphertext length restrictions in the Usage Notes.

#### **ciphertext\_in**

Direction: Input

Type: String

The text that is to be translated. The text is enciphered under the cipher key specified in the *key\_identifier\_in* parameter.

#### **chaining\_vector\_length**

Direction: Input

Type: Integer

The length of the *chaining\_vector* parameter in bytes. The *chaining\_vector* field must be 128 bytes long.

#### **chaining\_vector**

Direction: Input/Output

Type: String

The *chaining\_vector* parameter is a work area used by the service to carry segmented data between procedure calls. This area must not be modified between calls to the service.

#### **key\_identifier\_out\_length**

Direction: Input

Type: Integer

Length of the *key\_identifier\_out* field in bytes. This value is 64 when a label is supplied. When the key identifier is a key token, the value is the length of the token. The maximum value is 725.

#### **key\_identifier\_out**

Direction: Input/Output

Type: String

An internal key token or the label of the CKDS record containing the cipher translation key for the outbound ciphertext.

Acceptable DES key types are DATA, CIPHER, CIPHERXL, CIPHERXO, and ENCIPHER. The key may be a double- or triple-length key. If the **Cipher Text translate2 – Allow only cipher text translate types** access control point is enabled, only CIPHERXO and CIPHERXL are allowed. Acceptable DES key types are DATA, CIPHER, CIPHERXL, CIPHERXO, and ENCIPHER. The keys must have bit 18 for “ENCIPHER” set on in the control vector. The key may be a double- or triple-length key. If the Cipher Text translate2 - Allow only cipher text translate types access control point is enabled, only CIPHERXO and CIPHERXL are allowed.

Acceptable AES key types include the 64-byte AES DATA key and the variable length token CIPHER key with the ENCRYPT bit on in the key usage field. The C-XLATE bit can optionally be on. If the **Cipher Text translate2 – Allow only cipher text translate types** access control point is enabled, the C-XLATE bit must be turned on in the key usage field.

#### **initialization\_vector\_out\_length**

Direction: Input

Type: Integer

Length of the *initialization\_vector\_out* field in bytes. For AES keys, the length is 16. For DES keys, the length is 8. When the initialization vector is not required (segmenting rule CONTINUE, processing rule O-ECB), the value must be 0.

<b>initialization_vector_out</b>	
Direction: Input	Type: String
The initialization vector that is used to encipher the input data. This is the new initialization vector used when the callable service enciphers the plaintext. This parameter is required for segmenting rule INITIAL.	
<b>ciphertext_out_length</b>	
Direction: Input/Output	Type: Integer
Length of the <i>ciphertext_out</i> in bytes. This parameter will updated with the actual length of the data in the <i>ciphertext_out</i> parameter. Note that padding may require this value to be larger than the <i>ciphertext_in_length</i> parameter. See the table of ciphertext length restrictions in the Usage Notes.	
<b>ciphertext_out</b>	
Direction: Output	Type: String
The field where the callable service returns the translated text.	
<b>reserved1_length</b>	
Direction: Input	Type: Integer
The length of the <i>reserved1</i> parameter in bytes. The value must be zero.	
<b>reserved1</b>	
Direction: Input	Type: String
This parameter is ignored.	
<b>reserved2_length</b>	
Direction: Input	Type: Integer
The length of the <i>reserved2</i> parameter in bytes. The value must be zero.	
<b>reserved2</b>	
Direction: Input	Type: String
This parameter is ignored.	
<b>text_id_in</b>	
Direction: Input	Type: Integer
For CSNBCTT3 only, the ALET of the <i>ciphertext_in</i> parameter.	
<b>text_id_out</b>	
Direction: Input	Type: Integer
For CSNBCTT3 only, the ALET of the <i>ciphertext_out</i> parameter.	

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

The initialization vectors must have already been established between the communicating applications or must be passed with the data.

The following table outlines the restrictions for the *ciphertext\_in\_length* and *ciphertext\_out\_length* parameters. The DES blocks referred to in this table are 8 bytes. The AES blocks referred to in this table are 16 bytes.

Table 137. Restrictions for *ciphertext\_in\_length* and *ciphertext\_out\_length*

Input cipher method	Output cipher method	Input ciphertext length restriction[s]	Output ciphertext length restriction[s]
DES CBC	DES CBC X9.23	Input ciphertext must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the sum of the length of the input ciphertext and a DES block.
DES CBC	AES CBC PKCSPAD	Input cipher text must be a multiple of a DES block.	<p>If the input ciphertext is NOT a multiple of an AES block, then the output ciphertext length must be greater than or equal to the sum of the input ciphertext length and a DES block.</p> <p>If the input ciphertext is a multiple of an AES block, then the output ciphertext length must be greater than or equal to the sum of the input ciphertext length and an AES block.</p>
DES CBC	DES CUSP or IPS	Input cipher text must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC	DES CBC	Input cipher text must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC	AES CBC	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC	AES ECB	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC CUSP or IPS	DES CBC CUSP or IPS	No restrictions	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC CUSP or IPS	DES CBC	Input cipher text must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.



Table 137. Restrictions for ciphertext\_in\_length and ciphertext\_out\_length (continued)

Input cipher method	Output cipher method	Input ciphertext length restriction[s]	Output ciphertext length restriction[s]
DES CBC CUSP or IPS	AES CBC or ECB	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC CUSP or IPS	DES CBC X9.23	No restrictions	Output ciphertext length must be greater than or equal to the sum of the input ciphertext length and a DES block.
DES CBC CUSP or IPS	AES CBC PKCSPAD	No restrictions	Output ciphertext length must be greater than or equal to the sum of the input ciphertext length and a AES block.
DES CBC X9.23	DES CBC X9.23	Input ciphertext must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC X9.23	AES CBC PKCSPAD	Input ciphertext must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the sum of the input ciphertext length and a DES block.
DES CBC X9.23	DES CBC CUSP or IPS	Input ciphertext must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
DES CBC X9.23	DES CBC	Input ciphertext must be a multiple of a DES block.	Output ciphertext length must be greater than or equal to the input ciphertext length. <b>Note:</b> This operation will not be possible if the padding is determined by the adapter to be from 1-7 bytes.
DES CBC X9.23	AES CBC	Input ciphertext must be a multiple of a DES block but must not be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length. <b>Note:</b> This operation will not be possible if the padding is determined by the adapter to be from 1-7 bytes.

Table 137. Restrictions for ciphertext\_in\_length and ciphertext\_out\_length (continued)

Input cipher method	Output cipher method	Input ciphertext length restriction[s]	Output ciphertext length restriction[s]
DES CBC X9.23	AES ECB	Input ciphertext must be a multiple of a DES block but must not be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length. <b>Note:</b> This operation will not be possible if the padding is determined by the adapter to be from 1-7 bytes.
AES CBC or ECB	DES CBC X9.23	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the sum of the input ciphertext length and a DES block.
AES CBC or ECB	AES CBC PKCSPAD	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the sum of the input ciphertext length and an AES block.
AES CBC or ECB	DES CBC CUSP or IPS	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
AES CBC or ECB	DES CBC	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
AES CBC or ECB	AES CBC	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
AES CBC or ECB	AES ECB	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
AES CBC PKCSPAD	DES CBC X9.23	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
AES CBC PKCSPAD	AES CBC PKCSPAD	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length.
AES CBC PKCSPAD	DES CBC CUSP or IPS	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length minus 1.

Table 137. Restrictions for ciphertext\_in\_length and ciphertext\_out\_length (continued)

Input cipher method	Output cipher method	Input ciphertext length restriction[s]	Output ciphertext length restriction[s]
AES CBC PKCSPAD	DES CBC	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length minus the length of a DES block. <b>Note:</b> This operation will not be possible if the padding is determined by the adapter to be from 1-7 bytes or 9-15 bytes.
AES CBC PKCSPAD	AES CBC	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length minus the length of a AES block. <b>Note:</b> This operation will not be possible if the padding is determined by the adapter to be from 1-15 bytes.
AES CBC PKCSPAD	AES ECB	Input cipher text must be a multiple of an AES block.	Output ciphertext length must be greater than or equal to the input ciphertext length minus the length of a AES block. <b>Note:</b> This operation will not be possible if the padding is determined by the adapter to be from 1-15 bytes.

There are requirements for the keys for the *key\_identifier\_in* and *key\_identifier\_out* parameters. The *key\_identifier\_in* key must be able to decipher text. The *key\_identifier\_out* key must be able to encipher text.

The following table shows the valid key types which are allowed for the *key\_identifier\_in* and *key\_identifier\_out* parameters. In the table, a variable length key token cipher key is denoted by vCIPHER. vCIPHER is the default which has the ENCRYPT and DECRYPT bits on in the usage field. vCIPHERe has only the ENCRYPT bit on in the usage field. vCIPHERd has only the DECRYPT bit on in the usage field. Adding x to either of the preceding names means the TRANSLAT bit is on in the usage field for that key. (For example, vCIPHERex means a variable length token with the ENCRYPT and TRANSLAT bits turned on.)

AESDATA is the 64-byte AES DATA key type.

Table 138. Ciphertext translate2 key usage

key_identifier_in (DEC bit except DATA and AESDATA)	key_identifier_out (ENC bit except DATA and AESDATA)
DATA CIPHER DECIPHER CIPHERXI CIPHERXL	DATA CIPHER ENCIPHER CIPHERXO CIPHERXL AESDATA vCIPHER vCIPHERe vCIPHERex vCIPHERedx
AESDATA vCIPHER vCIPHERd vCIPHERdx vCIPHERdex	DATA (must be at least double-length key with ACP) CIPHER (requires ACP to be enabled) ENCIPHER (requires ACP to be enabled) CIPHERXO (requires ACP to be enabled) CIPHERXL (requires ACP to be enabled) AESDATA vCIPHER vCIPHERe vCIPHERex vCIPHERedx

**Notes:**

1. Translation from stronger encryption to single-key DES is not allowed.
2. Translation from a triple-length DES key to a double-length DES key requires the **Ciphertext translate2 – Allow translate to weaker DES** access control point to be enabled.
3. When the **Ciphertext translate2 – Allow only cipher text translate key types** access control point is enabled, only CIPHERXI, CIPHERXL, and CIPHERXO DES key types are allowed and the C-XLATE key usage bit must be on for AES CIPHER keys.

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

Table 139. Ciphertext translate2 Access Control Points

Access control point	Description
Cipher Text translate2	Enable Ciphertext Translate2 service
Cipher Text translate2 – Allow translate from AES to TDES	Allow translation from an AES key to 2 or 3 key triple DES key.
Cipher Text translate2 – Allow translate to weaker AES	Allow translation from a stronger to weaker AES key. (For example, IN key AES256 and OUT key AES128.)
Cipher Text translate2 – Allow translate to weaker DES	Allow translation from a stronger to weaker DES key. The only supported translation is from 3-key TDES to 2-key TDES.

Table 139. Ciphertext translate2 Access Control Points (continued)

Access control point	Description
Cipher Text translate2 – Allow only cipher text translate types	When enabled, the <i>key_identifiers</i> parameters must be a key with key type CIPHERXI, CIPHERXL, or CIPHERXO for DES and key type CIPHER with the C-XLATE key usage bit on for AES.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 140. Ciphertext translate2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This callable service is not supported.
IBM @server zSeries 990		This callable service is not supported.
IBM @server zSeries 890		
IBM System z9 EC		This callable service is not supported.
IBM System z9 BC		
IBM System z10 EC		This callable service is not supported.
IBM System z10 BC		
z196		This callable service is not supported.
z114		
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## Decipher (CSNBDEC or CSNBDEC1 and CSNEDEC or CSNEDEC1)

Use the decipher callable service to decipher data in an address space or a data space using the cipher block chaining mode. ICSF supports these processing rules to decipher data. You choose the type of processing rule that the decipher callable service should use for block chaining.

### Processing Rule

ANSI X9.23

### Purpose

For cipher block chaining. The ciphertext must be an exact multiple of 8 bytes, but the plaintext will be 1 to 8 bytes shorter than the ciphertext. The *text\_length* will also be reduced to show the original length of the plaintext.

<b>CBC</b>	For cipher block chaining. The ciphertext must be an exact multiple of 8 bytes, and the plaintext will have the same length.
<b>CUSP</b>	For cipher block chaining, but the ciphertext can be of any length. The plaintext will be the same length as the ciphertext.
<b>IBM 4700</b>	For cipher block chaining. The ciphertext must be an exact multiple of 8 bytes, but the plaintext will be 1 to 8 bytes shorter than the ciphertext. The <i>text_length</i> will also be reduced to show the original length of the plaintext.
<b>IPS</b>	For cipher block chaining, but the ciphertext can be of any length. The plaintext will be the same length as the ciphertext.

The cipher block chaining (CBC) mode uses an initial chaining value (ICV) in its processing. The first 8 bytes of ciphertext is deciphered and then the ICV is exclusive ORed with the resulting 8 bytes of data to form the first 8-byte block of plaintext. Thereafter, the 8-byte block of ciphertext is deciphered and exclusive ORed with the previous 8-byte block of ciphertext until all the ciphertext is deciphered.

The selection between single-DES decryption mode and triple-DES decryption mode is controlled by the length of the key supplied in the *key\_identifier* parameter. If a single-length key is supplied, single-DES decryption is performed. If a double-length or triple-length key is supplied, triple-DES decryption is performed.

A different ICV may be passed on each call to the decipher callable service. However, the same ICV that was used in the corresponding encipher callable service must be passed.

Short blocks are text lengths of 1 to 7 bytes. A short block can be the only block. Trailing short blocks are blocks of 1 to 7 bytes that follow an exact multiple of 8 bytes. For example, if the text length is 21, there are two 8-byte blocks and a trailing short block of 5 bytes. Because the DES and CDMF process only text in exact multiples of 8 bytes, some special processing is required to decipher such short blocks. Short blocks and trailing short blocks of 1 to 7 bytes of data are processed according to the Cryptographic Unit Support Program (CUSP) rules, or by the record chaining scheme devised by and used in the Information Protection System (IPS) in the IPS/CMS product.

These methods of treating short blocks and trailing short blocks do not increase the length of the ciphertext over the plaintext. If the plaintext was *padded* during encipherment, the length of the ciphertext will always be an exact multiple of 8 bytes.

ICSF supports these padding schemes:

- ANSI X9.23
- 4700-PAD

## Choosing Between CSNBDEC and CSNBDEC1

CSNBDEC and CSNBDEC1 provide identical functions. When choosing which service to use, consider this:

## Decipher

- **CSNBDEC** requires the ciphertext and plaintext to reside in the caller's primary address space. Also, a program using CSNBDEC adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface. The callable service name for AMODE(64) invocation is CSNEDEC.
  - **CSNBDEC1** allows the ciphertext and plaintext to reside either in the caller's primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using CSNBDEC1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface. The callable service name for AMODE(64) invocation is CSNEDEC1.
- For CSNBDEC1 and CSNEDEC1, *cipher\_text\_id* and *clear\_text\_id* are access list entry token (ALET) parameters of the data spaces containing the ciphertext and plaintext.

## Format

```
CALL CSNBDEC(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    key_identifier,  
    text_length,  
    cipher_text,  
    initialization_vector,  
    rule_array_count,  
    rule_array,  
    chaining_vector,  
    clear_text )
```

```
CALL CSNBDEC1(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    key_identifier,  
    text_length,  
    cipher_text,  
    initialization_vector,  
    rule_array_count,  
    rule_array,  
    chaining_vector,  
    clear_text,  
    cipher_text_id,  
    clear_text_id )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

## exit\_data\_length

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

## exit data

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

**key identifier**

Direction: Input/Output

Type: String

A 64-byte string that is the internal key token containing the data-encrypting key, or the label of a CKDS record containing a data-encrypting key, to be used for deciphering the data. If the key token or key label contains a single-length key, single-DES decryption is performed. If the key token or key label contains a double-length or triple-length key, triple-DES decryption is performed.

On the IBM @server zSeries 990, IBM @server zSeries 890, z9 EC and z9 BC single and double length CIPHER and DECIPHER keys are also supported.

**text length**

Direction: Input/Output

Type: Integer

On entry, you supply the length of the ciphertext. The maximum length of text is 214783647 bytes. A zero value for the *text\_length* parameter is not valid. If the returned deciphered text (*clear\_text* parameter) is a different length because of the removal of padding bytes, the value is updated to the length of the plaintext.

**Note:** The MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced.

The application program passes the length of the ciphertext to the callable service. The callable service returns the length of the plaintext to your application program.

**cipher text**

Direction: Input

Type: String

The text to be deciphered.

**initialization\_vector**



## Decipher

Direction: Input

Type: String

The 8-byte supplied string for the cipher block chaining. The first block of the ciphertext is deciphered and exclusive ORed with the initial chaining vector (ICV) to get the first block of cleartext. The input block is the next ICV. To decipher the data, you must use the same ICV used when you enciphered the data.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supply in the *rule\_array* parameter. The value must be 1, 2, or 3.

### **rule\_array**

Direction: Input

Type: Character string

An array of 8-byte keywords providing the processing control information. The array is positional. See the keywords in Table 141. The first keyword in the array is the processing rule. You choose the processing rule you want the callable service to use for deciphering the data. The second keyword is the ICV selection keyword. The third keyword (or the second if the ICV selection keyword is allowed to default) is the encryption algorithm to use.

The service will fail if keyword DES is specified in the *rule\_array* in a CDMF-only system. The service will likewise fail if keyword CDMF is specified in the *rule\_array* in a DES-only system.

*Table 141. Keywords for the Decipher Rule Array Control Information*

Keyword	Meaning
<b>Processing Rule (required)</b>	
Rules CUSP, IPS, X9.23, and 4700-PAD should be specified only when there is one request or on the last request of a sequence of chained requests	
CBC	Performs ANSI X3.102 cipher block chaining. The data must be a multiple of 8 bytes. An OCV is produced and placed in the <i>chaining_vector</i> parameter. If the ICV selection keyword CONTINUE is specified, the CBC OCV from the previous call is used as the ICV for this call.
CUSP	Performs deciphering that is compatible with IBM's CUSP and PCF products. The data can be of any length and does not need to be in multiples of 8 bytes. The ciphertext will be the same length as the plaintext. The CUSP/PCF OCV is placed in the <i>chaining_vector</i> parameter. If the ICV selection keyword CONTINUE is specified, the CUSP/PCF OCV from the previous call is used as the ICV for this call.
IPS	Performs deciphering that is compatible with IBM's IPS product. The data can be of any length and does not need to be in multiples of 8 bytes. The ciphertext will be the same length as the plaintext. The IPS OCV is placed in the <i>chaining_vector</i> parameter. If the ICV selection keyword CONTINUE is specified, the IPS OCV from the previous call is used as the ICV for this call.

Table 141. Keywords for the Decipher Rule Array Control Information (continued)

Keyword	Meaning
X9.23	Deciphers with cipher block chaining and text length reduced to the original value. This is compatible with the requirements in ANSI standard X9.23. The ciphertext length must be an exact multiple of 8 bytes. Padding is removed from the plaintext.
4700-PAD	Deciphers with cipher block chaining and text length reduced to the original value. The ciphertext length must be an exact multiple of 8 bytes. Padding is removed from the plaintext.
<b>ICV Selection (optional)</b>	
CONTINUE	This specifies taking the initialization vector from the output chaining vector (OCV) contained in the work area to which the <i>chaining_vector</i> parameter points. CONTINUE is valid only for processing rules CBC, IPS, and CUSP.
INITIAL	This specifies taking the initialization vector from the <i>initialization_vector</i> parameter. INITIAL is the default value.
<b>Encryption Algorithm (optional)</b>	
CDMF	This specifies using the Commercial Data Masking Facility and ignoring the token marking. You cannot use double- or triple-length keys with CDMF. The CDMF keyword, or tokens marked as CDMF, are only supported on an IBM @server zSeries 900.
DES	This specifies using the data encryption standard and ignoring the token marking.
TOKEN	This specifies using the data encryption algorithm in the DATA key token. This is the default.

“Cipher Processing Rules” on page 940 describes the cipher processing rules in detail.

### **chaining\_vector**

Direction: Input/Output

Type: String

An 18-byte field that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector holds the output chaining vector (OCV) from the caller. The OCV is the first 8 bytes in the 18-byte string.

The direction is output if the ICV selection keyword of the *rule\_array* parameter is INITIAL. The direction is input/output if the ICV selection keyword of the *rule\_array* parameter is CONTINUE.

### **clear\_text**

Direction: Output

Type: String

The field where the callable service returns the deciphered text.

### **cipher\_text\_id**

## Decipher

Direction: Input

Type: Integer

For CSNBDEC1/CSNEDEC1 only, the ALET of the ciphertext to be deciphered.

**clear\_text\_id**

Direction: Input

Type: Integer

For CSNBDEC1/CSNEDEC1 only, the ALET of the clear text supplied by the application.

## Restrictions

The service will fail under these conditions:

- If the keyword DES is specified in the *rule\_array* parameter in a CDMF-only system
- If the keyword CDMF is specified in the *rule\_array* parameter in a DES-only system
- If the key token contains double or triple-length keys and triple-DES is not enabled.
- If the keyword CDMF is specified on a PCIXCC or CCA Crypto Express coprocessor.
- If a token is marked CDMF on a PCIXCC or CCA Crypto Express coprocessor.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

On a CCF system, only a DATA key token or DATA key label can be used in this service.

Single and double length CIPHER and DECIPHER keys are supported on a PCIXCC or CCA Crypto Express coprocessor.

You **cannot** overlap the plaintext and ciphertext fields. For example:

pppppp  
cccccc is not supported.

cccccc  
pppppp is not supported.

ppppppcccccc is supported.

P represents the plaintext and c represents the ciphertext.

On z990, z890, z9 EC or z9 BC system, the PCIXCC/CEX2C will support non destructive overlap. For example:

pppppp  
cccccc is supported.

“Cipher Processing Rules” on page 940 discusses the cipher processing rules.

The Encipher callable services are described under “Encipher (CSNBENC or CSNBENC1 and CSNEENC or CSNEENC1)” on page 379.

## Access Control Point

The **Decipher - DES** access control point controls the function of this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 142. Decipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	If keyword CDMF is specified or if the token is marked as CDMF, the service fails.
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	If keyword CDMF is specified or if the token is marked as CDMF, the service fails.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	If keyword CDMF is specified or if the token is marked as CDMF, the service fails.
z196 z114	Crypto Express3 Coprocessor	If keyword CDMF is specified or if the token is marked as CDMF, the service fails.
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	If keyword CDMF is specified or if the token is marked as CDMF, the service fails.

## Decode (CSNBDCO and CSNEDCO)

Use this callable service to decipher an 8-byte string using a clear key. The callable service uses the electronic code book (ECB) mode of the DES. (This service is available only on a DES-capable system.)

The callable service name for AMODE(64) invocation is CSNEDCO.

## Considerations

If you have only a clear key, you are *not* limited to using only the encode and decode callable services.

- You can pass your clear key to the clear key import service, and get back a token that will allow you to use the encipher and decipher callable services.
- On an IBM @server zSeries 990 and subsequent releases, consider using the Symmetric Key Decipher service (“Symmetric Key Decipher (CSNBSYD or CSNBSYD1 and CSNESYD or CSNESYD1)” on page 402).

## Format

```
CALL CSNBDCO(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    clear_key,
    cipher_text,
    clear_text)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **clear\_key**

Direction: Input

Type: String

The 8-byte clear key value that is used to decode the data.

### **cipher\_text**

Direction: Input

Type: String

The ciphertext that is to be decoded. Specify 8 bytes of text.

### **clear\_text**

Direction: Output

Type: String

The 8-byte field where the plaintext is returned by the callable service.

## Restrictions

You cannot use this service on a CDMF-only system.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 143. Decode required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990 IBM @server zSeries 890	CP Assist for Cryptographic Functions	
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions	
z196 z114	CP Assist for Cryptographic Functions	
IBM zEnterprise EC12	CP Assist for Cryptographic Functions	

## Encipher (CSNBENC or CSNBENC1 and CSNEENC or CSNEENC1)

Use the encipher callable service to encipher data in an address space or a data space using the cipher block chaining mode. ICSF supports these processing rules to encipher data. You choose the type of processing rule that the encipher callable service should use for the block chaining.

Processing Rule	Purpose
ANSI X9.23	For block chaining not necessarily in exact multiples of 8 bytes. This process rule pads the plaintext so that ciphertext produced is an exact multiple of 8 bytes.
CBC	For block chaining in exact multiples of 8 bytes.
CUSP	For block chaining not necessarily in exact multiples of 8 bytes. The ciphertext will be the same length as the plaintext.
IBM 4700	For block chaining not necessarily in exact

multiples of 8 bytes. This process rule pads the plaintext so that the ciphertext produced is an exact multiple of 8 bytes.

### IPS

For block chaining not necessarily in exact multiples of 8 bytes. The ciphertext will be the same length as the plaintext.

For more information about the processing rules, see Table 144 on page 384 and “Cipher Processing Rules” on page 940.

The cipher block chaining (CBC) mode of operation uses an initial chaining vector (ICV) in its processing. The ICV is exclusive ORed with the first 8 bytes of plaintext prior to the encryption step, and thereafter, the 8-byte block of ciphertext just produced is exclusive ORed with the next 8-byte block of plaintext, and so on. This disguises any pattern that may exist in the plaintext.

The selection between single-DES encryption mode and triple-DES encryption mode is controlled by the length of the key supplied in the *key\_identifier* parameter. If a single-length key is supplied, single-DES encryption is performed. If a double-length or triple-length key is supplied, triple-DES encryption is performed.

To nullify the CBC effect on the first 8-byte block, supply 8 bytes of zero. However, the ICV may require zeros.

Cipher block chaining also produces a resulting chaining value called the output chaining vector (OCV). The application can pass the OCV as the ICV in the next encipher call. This results in *record chaining*.

Note that the OCV that results is the same, whether an encipher or a decipher callable service was invoked, assuming the same text, ICV, and key were used.

Short blocks are text lengths of 1 to 7 bytes. A short block can be the only block. Trailing short blocks are blocks of 1 to 7 bytes that follow an exact multiple of 8 bytes. For example, if the text length is 21, there are two 8-byte blocks, and a trailing short block of 5 bytes. Short blocks and trailing short blocks of 1 to 7 bytes of data are processed according to the Cryptographic Unit Support Program (CUSP) rules, or by the record chaining scheme devised by and used by the Information Protection System (IPS) in the IPS/CMS program product. These methods of treating short blocks and trailing short blocks do not increase the length of the ciphertext over the plaintext.

An alternative method is to pad the plaintext and produce a ciphertext that is longer than the plaintext. The plaintext can be padded with up to 8 bytes using one of several padding schemes. This padding produces a ciphertext that is an exact multiple of 8 bytes long.

If the ciphertext is to be transmitted over a network, where one or more intermediate nodes will use the ciphertext translate callable service, the ciphertext *must* be produced using one of these methods of padding:

- ANSI X9.23
- 4700

If the cleartext is already a multiple of 8, the ciphertext can be created using any processing rule.

Because of padding, the returned ciphertext length is longer than the provided plaintext; the *text\_length* parameter *will have been modified*. The returned ciphertext field should be 8 bytes longer than the length of the plaintext to accommodate the maximum amount of padding. You should provide this extension in your installation's storage because ICSF cannot detect whether the extension was done.

The minimum length of data that can be enciphered is one byte.

**Attention:** If you lose the data-encrypting key under which the data (plaintext) is enciphered, the data enciphered under that key (ciphertext) **cannot** be recovered.

## Choosing between CSNBENC and CSNBENC1

CSNBENC and CSNBENC1 provide identical functions. When choosing which service to use, consider this:

- **CSNBENC** requires the cleartext and ciphertext to reside in the caller's primary address space. Also, a program using CSNBENC adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface. The callable service name for AMODE(64) invocation is CSNEENC.
- **CSNBENC1** allows the cleartext and ciphertext to reside either in the caller's primary address space or in a data space. This can allow you to encipher more data with one call. However, a program using CSNBENC1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

The callable service name for AMODE(64) invocation is CSNEENC1.

For CSNBENC1 and CSNEENC1, *clear\_text\_id* and *cipher\_text\_id* are access list entry token (ALET) parameters of the data spaces containing the cleartext and ciphertext.

## Format

```
CALL CSNBENC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    clear_text,
    initialization_vector,
    rule_array_count,
    rule_array,
    pad_character,
    chaining_vector,
    cipher_text )
```



```
CALL CSNBENC1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    clear_text,
    initialization_vector,
    rule_array_count,
    rule_array,
    pad_character,
    chaining_vector,
    cipher_text,
    clear_text_id,
    cipher_text_id )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string that is the internal key token containing the data-encrypting key, or the label of a CKDS record containing the data-encrypting key, to be used for encrypting the data. If the key token or key label contains a single-length key, single-DES encryption is performed. If the key token or key label contains a double-length or triple-length key, triple-DES encryption is performed.

On an IBM @server zSeries 990 and subsequent releases, single and double length CIPHER and ENCIPHER keys are also supported.

### **text\_length**

Direction: Input/Output

Type: Integer

On entry, the length of the plaintext (*clear\_text* parameter) you supply. The maximum length of text is 2,147,836,47 bytes. A zero value for the *text\_length* parameter is not valid. If the returned enciphered text (*cipher\_text* parameter) is a different length because of the addition of padding bytes, the value is updated to the length of the ciphertext.

**Note:** The MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647).

The application program passes the length of the plaintext to the callable service. The callable service returns the length of the ciphertext to the application program.

### **clear\_text**

Direction: Input

Type: String

The text that is to be enciphered.

### **initialization\_vector**

Direction: Input

Type: String

The 8-byte supplied string for the cipher block chaining. The first 8 bytes (or less) block of the data is exclusive ORed with the ICV and then enciphered. The input block is enciphered and the next ICV is created. You must use the same ICV to decipher the data.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supply in the *rule\_array* parameter. The value must be 1, 2, or 3.

### **rule\_array**

Direction: Input

Type: Character string

An array of 8-byte keywords providing the processing control information. The array is positional. See the keywords in Table 144 on page 384. The first keyword in the array is the processing rule. You choose the processing rule you want the callable service to use for enciphering the data. The second keyword is the ICV selection keyword. The third keyword (or the second if the ICV selection keyword is allowed to default to INITIAL) is the encryption algorithm to use.

The service will fail if keyword DES is specified in the *rule\_array* in a CDMF-only system. The service will likewise fail if the keyword CDMF is specified in the *rule\_array* in a DES-only system.

Table 144. Keywords for the Encipher Rule Array Control Information

Keyword	Meaning
<b>Processing Rule (required)</b>	
Rules CUSP, IPS, X9.23, and 4700-PAD should be specified only when there is one request or on the last request of a sequence of chained requests.	
CBC	Performs ANSI X3.102 cipher block chaining. The data must be a multiple of 8 bytes. An OCV is produced and placed in the <i>chaining_vector</i> parameter. If the ICV selection keyword CONTINUE is specified, the CBC OCV from the previous call is used as the ICV for this call.
CUSP	Performs ciphering that is compatible with IBM's CUSP and PCF products. The data can be of any length and does not need to be in multiples of 8 bytes. The ciphertext will be the same length as the plaintext. The CUSP/PCF OCV is placed in the <i>chaining_vector</i> parameter. If the ICV selection keyword CONTINUE is specified, the CUSP/PCF OCV from the previous call is used as the ICV for this call.
IPS	Performs ciphering that is compatible with IBM's IPS product. The data may be of any length and does not need to be in multiples of 8 bytes. The ciphertext will be the same length as the plaintext. The IPS OCV is placed in the <i>chaining_vector</i> parameter. If the ICV selection keyword CONTINUE is specified, the IPS OCV from the previous call is used as the ICV for this call.
X9.23	Performs cipher block chaining with 1 to 8 bytes of padding. This is compatible with the requirements in ANSI standard X9.23. If the data is not in exact multiples of 8 bytes, X9.23 pads the plaintext so that the ciphertext produced is an exact multiple of 8 bytes. The plaintext is padded to the next multiple 8 bytes, even if this adds 8 bytes. An OCV is produced.
4700-PAD	Performs padding by extending the user's plaintext with the caller's specified pad character, followed by a one-byte binary count field that contains the total number of bytes added to the message. 4700-PAD pads the plaintext so that the ciphertext produced is an exact multiple of 8 bytes. An OCV is produced.
<b>ICV Selection (optional)</b>	
CONTINUE	This specifies taking the initialization vector from the output chaining vector (OCV) contained in the work area to which the <i>chaining_vector</i> parameter points. CONTINUE is valid only for processing rules CBC, IPS, and CUSP.
INITIAL	This specifies taking the initialization vector from the <i>initialization_vector</i> parameter. INITIAL is the default value.
<b>Encryption Algorithm (optional)</b>	
CDMF	This specifies using the Commercial Data Masking Facility and ignoring the token marking. You cannot use double-length or triple-length keys with CDMF. The CDMF keyword, or tokens marked as CDMF, are only supported on an IBM @server zSeries 900.
DES	This specifies using the data encryption standard and ignoring the token marking.

Table 144. Keywords for the Encipher Rule Array Control Information (continued)

Keyword	Meaning
TOKEN	This specifies using the data encryption algorithm in the DATA key token. TOKEN is the default.

These recommendations help the caller determine which encipher processing rule to use:

- If you are exchanging enciphered data with a specific implementation, for example, CUSP or ANSI X9.23, use that processing rule.
- If the ciphertext translate callable service is to be invoked on the enciphered data at an intermediate node, ensure that the ciphertext is a multiple of 8 bytes. Use CBC, X9.23, or 4700-PAD to prevent the creation of ciphertext that is not a multiple of 8 bytes and that cannot be processed by the ciphertext translate callable service.
- If the ciphertext length must be equal to the plaintext length and the plaintext length cannot be a multiple of 8 bytes, use either the IPS or CUSP processing rule.

“Cipher Processing Rules” on page 940 describes the cipher processing rules in detail.

#### **pad\_character**

Direction: Input

Type: Integer

An integer, 0 to 255, that is used as a padding character for the 4700-PAD process rule (*rule\_array* parameter).

#### **chaining\_vector**

Direction: Input/Output

Type: String

An 18-byte field that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector holds the output chaining vector (OCV) from the caller. The OCV is the first 8 bytes in the 18-byte string.

The direction is output if the ICV selection keyword of the *rule\_array* parameter is INITIAL.

The direction is input/output if the ICV selection keyword of the *rule\_array* parameter is CONTINUE.

#### **cipher\_text**

Direction: Output

Type: String

The enciphered text the callable service returns. The length of the ciphertext is returned in the *text\_length* parameter. The *cipher\_text* may be 8 bytes longer than the length of the *clear\_text* field because of the padding that is required for some processing rules.

#### **clear\_text\_id**

Direction: Input

Type: Integer

For CSNBENC1/CSNEENC1 only, the ALET of the clear text to be enciphered.

## Encipher

### cipher\_text\_id

Direction: Input

Type: Integer

For CSNBENC1/CSNEENC1 only, the ALET of the ciphertext that the application supplied.

## Restrictions

The service will fail under these conditions:

- If the keyword DES is specified in the *rule\_array* parameter in a CDMF-only system
- If the keyword CDMF is specified in the *rule\_array* parameter in a DES-only system
- If the key token contains double- or triple-length keys and triple-DES is not enabled.
- If the keyword CDMF is specified on a PCIXCC or CCA Crypto Express coprocessor.
- If a token is marked CDMF on a PCIXCC or CCA Crypto Express coprocessor.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

On a CCF system, only a DATA key token or DATA key label can be used in this service.

Single and double length CIPHER and ENCIPHER keys are supported on a PCIXCC or CCA Crypto Express coprocessor.

## Access Control Point

The **Encipher - DES** access control point controls the function of this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 145. Encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	If keyword CDMF is specified or if the token is marked as CDMF, the service fails.
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	If keyword CDMF is specified or if the token is marked as CDMF, the service fails.

Table 145. Encipher required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor  Crypto Express3 Coprocessor	If keyword CDMF is specified or if the token is marked as CDMF, the service fails.
z196  z114	Crypto Express3 Coprocessor	If keyword CDMF is specified or if the token is marked as CDMF, the service fails.
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	If keyword CDMF is specified or if the token is marked as CDMF, the service fails.

## Related Information

You **cannot** overlap the plaintext and ciphertext fields. For example:

```
pppppp
  cccccc  is not supported.
```

```
ccccc
  pppppp  is not supported.
```

```
ppppppccccc is supported.
```

P represents the plaintext and c represents the ciphertext.

On z990, z890 , z9 EC and z9 BC systems, the PCIXCC/CEX2C will support non destructive overlap. For example:

```
ccccc
  pppppp  is supported.
```

The method used to produce the OCV is the same with the CBC, 4700-PAD, and X9.23 processing rules. However, that method is different from the method used by the CUSP and IPS processing rules.

“Cipher Processing Rules” on page 940 discusses the cipher processing rules.

The Decipher callable services are described under “Decipher (CSNBDEC or CSNBDEC1 and CSNEDEC or CSNEDEC1)” on page 370.

---

## Encode (CSNBECO and CSNEECO)

Use the encode callable service to encipher an 8-byte string using a clear key. The callable service uses the electronic code book (ECB) mode of the DES. (This service is available only on a DES-capable system.)

The callable service name for AMODE(64) invocation is CSNEECO.

## Considerations

If you have only a clear key, you are *not* limited to using just the encode and decode callable services.

## Encode

- You can pass your clear key to the clear key import service, and get back a token that will allow you to use the encipher and decipher callable services.
- On an IBM @server zSeries 990 and subsequent releases, consider using the Symmetric Key Encipher service (“Symmetric Key Encipher (CSNBSYE or CSNBSYE1 and CSNESYE or CSNESYE1)” on page 412).

## Format

```
CALL CSNBECO(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    clear_key,  
    clear_text,  
    cipher_text)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **clear\_key**

Direction: Input

Type: String

The 8-byte clear key value that is used to encode the data.

### **clear\_text**

Direction: Input

Type: String

The plaintext that is to be encoded. Specify 8 bytes of text.

**cipher\_text**

Direction: Output

Type: String

The 8-byte field where the ciphertext is returned by the callable service.

## Restrictions

You cannot use this service on a CDMF-only system.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 146. Encode required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990	CP Assist for Cryptographic Functions	
IBM @server zSeries 890		
IBM System z9 EC	CP Assist for Cryptographic Functions	
IBM System z9 BC		
IBM System z10 EC	CP Assist for Cryptographic Functions	
IBM System z10 BC		
z196	CP Assist for Cryptographic Functions	
z114		
IBM zEnterprise EC12	CP Assist for Cryptographic Functions	

## Symmetric Algorithm Decipher (CSNBSAD or CSNBSAD1 and CSNESAD or CSNESAD1)

The symmetric algorithm decipher callable service deciphers data with the AES algorithm. Data is deciphered that has been enciphered in either CBC mode or ECB mode.

You can specify that the clear text data was padded before encryption using the method described in the PKCS standards. In this case, the callable service will remove the padding bytes and return the unpadded clear text data. PKCS padding is described in “PKCS Padding Method” on page 943.

The callable service names for AMODE(64) invocation are CSNESAD and CSNESAD1.



### Choosing Between CSNBSAD and CSNBSAD1 or CSNESAD and CSNESAD1

CSNBSAD, CSNBSAD1, CSNESAD, and CSNESAD1 provide identical functions. When choosing which service to use, consider this:

- CSNBSAD and CSNESAD require the cipher text and plaintext to reside in the caller's primary address space. Also, a program using CSNBSAD adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- CSNBSAD1 and CSNESAD1 allow the cipher text and plaintext to reside either in the caller's primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using CSNBSAD1 and CSNESAD1 does not adhere to the IBM CCA: Cryptographic API and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSAD1 and CSNESAD1, *cipher\_text\_id* and *clear\_text\_id* are access list entry token (ALET) parameters of the data spaces containing the cipher text and plaintext.

### Format

```
CALL CSNBSAD(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_identifier_length,  
    key_identifier,  
    key_parms_length,  
    key_parms,  
    block_size,  
    initialization_vector_length,  
    initialization_vector,  
    chain_data_length,  
    chain_data,  
    cipher_text_length,  
    cipher_text,  
    clear_text_length,  
    clear_text,  
    optional_data_length,  
    optional_data)
```

```
CALL CSNBSAD1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_length,
    key_identifier,
    key_parms_length,
    key_parms,
    block_size,
    initialization_vector_length,
    initialization_vector,
    chain_data_length,
    chain_data,
    cipher_text_length,
    cipher_text,
    clear_text_length,
    clear_text,
    optional_data_length,
    optional_data,
    cipher_text_id,
    clear_text_id)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **rule\_array\_count**

Direction: Input

Type: Integer

## Symmetric Algorithm Decipher

The number of keywords you supplied in the *rule\_array* parameter. The value may be 2, 3 or 4.

### **rule\_array**

Direction: Input

Type: String

An array of 8-byte keywords providing the processing control information. The keywords must be in contiguous storage, left-justified and padded on the right with blanks.

Table 147. Symmetric Algorithm Decipher Rule Array Keywords

Keyword	Meaning
<i>Algorithm (one keyword, required)</i>	
AES	Specifies that the Advanced Encryption Standard (AES) algorithm is to be used. The block size is 16 bytes. The key length may be 16, 24, or 32 bytes.
<i>Processing Rule (optional - zero or one keyword)</i>	
CBC	Performs encryption in cipher block chaining (CBC) mode. The text length must be a multiple of the AES block size (16-bytes). This is the default value.
ECB	Performs encryption in electronic code book (ECB) mode. The text length must be a multiple of the AES block size (16-bytes).
PKCS-PAD	Deciphers with cipher block chaining and text length reduced to the original value. The ciphertext length must be an exact multiple of 16 bytes. Padding is removed from the plaintext. This rule should be specified only when there is one request or on the last request of a sequence of chained requests.
<i>Key Rule (required)</i>	
KEYIDENT	This indicates that the value in the <i>key_identifier</i> parameter is either an internal key token or the label of a key token in the CKDS. The key must be a secure AES key, that is, enciphered under the current master key.
<i>ICV Selection (optional - zero or one keyword)</i>	
INITIAL	This specifies that this is the first request of a sequence of chained requests, and indicates that the initialization vector should be taken from the <i>initialization_vector</i> parameter. This is the default value.
CONTINUE	This specifies that this request is part of a sequence of chained requests, and is not the first request in that sequence. The initialization vector will be taken from the work area identified in the <i>chain_data</i> parameter. This keyword is only valid for processing rules CBC or PKCS-PAD.

### **key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *key\_identifier* parameter. The length must be 64 bytes for an AES DATA Internal Key Token (version X'04') or a CKDS label, or between the actual length of the token and 725 for an AES CIPHER Internal Key Token (version X'05').

**key\_identifier**

Direction: Input

Type: String

This specifies an internal secure AES token or the labelname of a secure AES token in the CKDS. Normal CKDS labelname syntax is required.

The AES key identifier must be an encrypted key contained in an internal key token, where the key is enciphered under the AES master key. The key can be 128-, 192-, or 256-bits in length.

**key\_parms\_length**

Direction: Input

Type: Integer

The length of the *key\_parms* parameter. This must be 0.

**key\_parms**

Direction: Ignored

Type: String

This parameter is ignored. It is reserved for future use.

**block\_size**

Direction: Input

Type: Integer

The block size for the cryptographic algorithm. AES requires the block size to be 16.

**initialization\_vector\_length**

Direction: Input

Type: Integer

The length of the *initialization\_vector* parameter. The length should be equal to the block length for the algorithm specified. This parameter is ignored if the process rule is ECB.

**initialization\_vector**

Direction: Input

Type: String

This parameter contains the initialization vector (IV) for CBC mode decryption, including CBC mode invoked using the PKCS-PAD keyword. This parameter is ignored if the process rule is ECB. For AES CBC mode decryption, the initialization vector length must be 16 bytes, the length of an AES block. The IV must be the same value used when the data was encrypted.

**chain\_data\_length**

Direction: Input/Output

Type: Integer

## Symmetric Algorithm Decipher

The length of the *chain\_data* parameter. On input it contains the length of the buffer provided with parameter *chain\_data*. On output, it is updated with the length of the data returned in the *chain\_data* parameter.

### **chain\_data**

Direction: Input/Output

Type: String

A buffer that is used as a work area for sequences of chained symmetric algorithm decipher requests. When the keyword INITIAL is used, this is an output parameter and receives data that is needed when deciphering the next part of the input data. When the keyword CONTINUE is used, this is an input/output parameter; the value received as output from the previous call in the sequence is provided as input to this call, and in turn this call will return new *chain\_data* that will be used as input on the next call. When CONTINUE is used, both the data (*chain\_data* parameter) and the length (*chain\_data\_length* parameter) must be the same values that were received in these parameters as output on the preceding call to the service in the chained sequence.

The exact content and layout of *chain\_data* is not described. For AES CBC encryption, the field must be at least 32-bytes in length. For AES ECB encryption the field is not used and any length is acceptable including zero.

### **cipher\_text\_length**

Direction: Input

Type: Integer

The length of the cipher text. The length must be a multiple of the algorithm block size.

### **cipher\_text**

Direction: Input

Type: String

The text to be deciphered.

### **clear\_text\_length**

Direction: Input/Output

Type: Integer

On input, this parameter specifies the size of the storage pointed to by the *clear\_text* parameter. On output, this parameter has the actual length of the text stored in the *clear\_text* parameter.

If process rule PKCS-PAD is used, the clear text length will be less than the cipher text length since padding bytes are removed.

### **clear\_text**

Direction: Output

Type: String

The deciphered text the service returns.

### **optional\_data\_length**

Direction: Input

Type: Integer

The length of the *optional\_data* parameter. This parameter must be 0.

### **optional\_data**

Direction: Ignored

Type: String

Optional data required by a specified algorithm.

**cipher\_text\_id**

Direction: Input

Type: Integer

For CSNBSAD1 and CSNESAD1 only, the ALET of the dataspace in which the *cipher\_text* parameter resides.

**clear\_text\_id**

Direction: Input

Type: Integer

For CSNBSAD1 and CSNESAD1 only, the ALET of the dataspace in which the *clear\_text* parameter resides.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

## Access Control Point

The **Symmetric Algorithm Decipher - secure AES keys** access control point controls the function of this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 148. Symmetric Algorithm Decipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This service is not supported.
IBM @server zSeries 990		This service is not supported.
IBM @server zSeries 890		
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC)
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor  Crypto Express3 Coprocessor	Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC)
z196  z114	Crypto Express3 Coprocessor	AES Variable-length Symmetric Internal Key Tokens require the Sep. 2011 or later licensed internal code (LIC).

|

## Symmetric Algorithm Decipher

Table 148. Symmetric Algorithm Decipher required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

---

## Symmetric Algorithm Encipher (CSNBSAE or CSNBSAE1 and CSNESAE or CSNESAE1)

The symmetric algorithm encipher callable service enciphers data with the AES algorithm. Data is enciphered that has been deciphered in either CBC mode or ECB mode.

The callable service names for AMODE(64) invocation are CSNESAE and CSNESAE1

### Choosing between CSNBSAE and CSNBSAE1 or CSNESAE and CSNESAE1

CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1 provide identical functions. When choosing which service to use, consider this:

- CSNBSAE and CSNESAE require the cipher text and plaintext to reside in the caller's primary address space. Also, a program using CSNBSAE adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- CSNBSAE1 and CSNESAE1 allow the cipher text and plaintext to reside either in the caller's primary address space or in a data space. This can allow you to encipher more data with one call. However, a program using CSNBSAE1 and CSNESAE1 does not adhere to the IBM CCA: Cryptographic API and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSAE1 and CSNESAE1, *cipher\_text\_id* and *clear\_text\_id* are access list entry token (ALET) parameters of the data spaces containing the cipher text and plaintext.

## Format

```
CALL CSNBSAE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    key_parms_length,
    key_parms,
    block_size,
    initialization_vector_length,
    initialization_vector,
    chain_data_length,
    chain_data,
    clear_text_length,
    clear_text,
    cipher_text_length,
    cipher_text,
    optional_data_length,
    optional_data)
```

```
CALL CSNBSAE1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    key_parms_length,
    key_parms,
    block_size,
    initialization_vector_length,
    initialization_vector,
    chain_data_length,
    chain_data,
    clear_text_length,
    clear_text,
    cipher_text_length,
    cipher_text,
    optional_data_length,
    optional_data,
    clear_text_id,
    cipher_text_id)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer



## Symmetric Algorithm Encipher

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value may be 2, 3 or 4.

### **rule\_array**

Direction: Input

Type: String

This keyword provides control information to the callable service. The keywords must be eight bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 149. Symmetric Algorithm Encipher Rule Array Keywords

Keyword	Meaning
<i>Algorithm (one keyword, required)</i>	
AES	Specifies that the Advanced Encryption Standard (AES) algorithm will be used. The block size is 16-bytes, and the key length may be 16-, 24-, or 32-bytes (128-, 192-, 256-bits).
<i>Processing Rule (optional - zero or one keyword)</i>	
CBC	Performs encryption in cipher block chaining (CBC) mode. The text length must be a multiple of the AES block size (16-bytes). This is the default value.
ECB	Performs encryption in electronic code book (ECB) mode. The text length must be a multiple of the AES block size (16-bytes).
PKCS-PAD	Performs encryption in cipher block chaining (CBC) mode, but the data is padded using PKCS padding rules. The length of the clear text data does not have to be a multiple of the cipher block length. The cipher text will be longer than the clear text by at least one byte, and up to 16-bytes. The PKCS padding method is described in "PKCS Padding Method" on page 943. This rule should be specified only when there is one request or on the last request of a sequence of chained requests.
<i>Key Rule (required)</i>	

Table 149. Symmetric Algorithm Encipher Rule Array Keywords (continued)

Keyword	Meaning
KEYIDENT	This indicates that the value in the <i>key_identifier</i> parameter is either an internal key token or the label of a key token in the CKDS. The key must be a secure AES key, that is, enciphered under the current master key.
<i>ICV Selection (optional - zero or one keyword)</i>	
INITIAL	This specifies that this is the first request of a sequence of chained requests, and indicates that the initialization vector should be taken from the <i>initialization_vector</i> parameter. This is the default value.
CONTINUE	This specifies that this request is part of a sequence of chained requests, and is not the first request in that sequence. The initialization vector will be taken from the work area identified in the <i>chain_data</i> parameter. This keyword is only valid for processing rules CBC or PKCS-PAD.

**key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *key\_identifier* parameter. The length must be 64 bytes for an AES DATA Internal Key Token (version X'04') or a CKDS label, or between the actual length of the token and 725 for an AES CIPHER Internal Key Token (version X'05').

**key\_identifier**

Direction: Input

Type: String

This specifies an internal secure AES token or the labelname of a secure AES token in the CKDS. Normal CKDS labelname syntax is required.

The AES key identifier must be an encrypted key contained in an internal key token, where the key is enciphered under the AES master key. The key can be 128-, 192-, or 256-bits in length.

**key\_parms\_length**

Direction: Input

Type: Integer

The length of the *key\_parms* parameter in bytes. It must be set to 0.

**key\_parms**

Direction: Ignored

Type: String

This parameter is ignored. It is reserved for future use.

**block\_size**

Direction: Input

Type: Integer

The block size for the cryptographic algorithm. AES requires the block size to be 16.

## Symmetric Algorithm Encipher

### **initialization\_vector\_length**

Direction: Input

Type: Integer

The length of the *initialization\_vector* parameter in bytes. This parameter is ignored if the process rule is ECB.

### **initialization\_vector**

Direction: Input

Type: String

This parameter contains the initialization vector (IV) for CBC mode encryption, including the CBC mode invoked using the PKCS-PAD keyword. This parameter is ignored if the process rule is ECB. For AES CBC mode encryption, the initialization vector length must be 16 bytes, the length of an AES block. The same IV must be used when decrypting the data.

### **chain\_data\_length**

Direction: Input/Output

Type: Integer

The length in bytes of the *chain\_data* parameter. On input it contains the length of the buffer provided with parameter *chain\_data*. On output, it is updated with the length of the data returned in the *chain\_data* parameter.

### **chain\_data**

Direction: Input/Output

Type: String

A buffer that is used as a work area for sequences of chained symmetric algorithm encipher requests. When the keyword INITIAL is used, this is an output parameter and receives data that is needed when enciphering the next part of the input data. When the keyword CONTINUE is used, this is an input/output parameter; the value received as output from the previous call in the sequence is provided as input to this call, and in turn this call will return new *chain\_data* that will be used as input on the next call. When CONTINUE is used, both the data (*chain\_data* parameter) and the length (*chain\_data\_length* parameter) must be the same values that were received in these parameters as output on the preceding call to the service in the chained sequence.

The exact content and layout of *chain\_data* is not described. For AES CBC encryption, the field must be at least 32-bytes in length. For AES ECB encryption the field is not used and any length is acceptable including zero.

### **clear\_text\_length**

Direction: Input

Type: Integer

The length of the clear text data in the *clear\_text* parameter. Unless process rule PKCS-PAD is used, the length must be a multiple of the algorithm block size. The length must be 1 or greater.

### **clear\_text**

Direction: Input

Type: String

The text to be enciphered.

### **cipher\_text\_length**

Direction: Input/Output

Type: Integer

On input, this parameter specifies the size of the storage pointed to by the *cipher\_text* parameter. On output, this parameter has the actual length of the text stored in the buffer addressed by the *cipher\_text* parameter.

If process rule PKCS-PAD is used, the cipher text length will exceed the clear text length by at least one byte, and up to 16-bytes. For other process rules, the cipher text length will be equal to the clear text length.

### **cipher\_text**

Direction: Output

Type: String

The enciphered text the service returns.

### **optional\_data\_length**

Direction: Input

Type: Integer

The length of the *optional\_data* parameter. This parameter is reserved for future use. It must be set to 0.

### **optional\_data**

Direction: Ignored

Type: String

The optional data used in processing the request. This parameter is ignored.

### **cipher\_text\_id**

Direction: Input

Type: Integer

For CSNBSAE1 and CSNESAE1 only, the ALET of the dataspace in which the *cipher\_text* parameter resides.

### **clear\_text\_id**

Direction: Input

Type: Integer

For CSNBSAE1 and CSNESAE1 only, the ALET of the dataspace in which the *clear\_text* parameter resides.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

## Access Control Point

The **Symmetric Algorithm Encipher - secure AES keys** access control point controls the function of this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

## Symmetric Algorithm Encipher

Table 150. Symmetric Algorithm Encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This service is not supported.
IBM @server zSeries 990		This service is not supported.
IBM @server zSeries 890		
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	Secure AES key support requires the Nov. 2008 or later licensed internal code (LIC).
z196 z114	Crypto Express3 Coprocessor	AES Variable-length Symmetric Internal Key Tokens require the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	

## Symmetric Key Decipher (CSNBSYD or CSNBSYD1 and CSNESYD or CSNESYD1)

Use the symmetric key decipher callable service to decipher data using one of the supported modes. ICSF supports several processing rules to decipher data. You choose the type of processing rule that the Symmetric Key Decipher callable service should use for block chaining. See “Modes of Operation” on page 351 for more information.

### Processing Rule

#### ANSI X9.23

### Purpose

For cipher block chaining. The ciphertext must be an exact multiple of the block size for the specified algorithm (8 bytes for DES). The plaintext will be between 1 and 8 bytes shorter than the ciphertext. This process rule always pads the plaintext during encryption so that ciphertext produced is an exact multiple of the block size, even if the plaintext was already a multiple of the blocksize.

#### CBC

For cipher block chaining. The ciphertext must be an exact multiple of the block size for the specified algorithm (8 bytes for DES, 16 bytes for AES). The plaintext will have the same length as the ciphertext.

<b>CBC-CS</b>	For cipher block chaining. The ciphertext can be any length. The plaintext will have the same length as the ciphertext.
<b>CFB</b>	Performs cipher feedback encryption with the segment size equal to the block size. The ciphertext can be of any length. The plaintext will have the same length as the ciphertext.
<b>CFB-LCFB</b>	Performs cipher feedback encryption with the segment size set by the caller. The ciphertext can be of any length. The plaintext will have the same length as the ciphertext.
<b>CUSP</b>	For cipher block chaining. The ciphertext can be of any length. The plaintext will have the same length as the ciphertext.
<b>ECB</b>	Performs electronic code book encryption. The ciphertext must be an exact multiple of the block size for the specified algorithm (8 bytes for DES, 16 bytes for AES). The plaintext will have the same length as the ciphertext.
<b>GCM</b>	Perform Galois/Counter mode decryption, which provides both confidentiality and authentication for the plaintext and authentication for the additional authenticated data (AAD). The ciphertext can be any length. The plaintext will have the same length as the ciphertext. Additionally, the authentication tag will be verified before any data is returned.
<b>IPS</b>	For cipher block chaining. The ciphertext can be any length. The plaintext will have the same length as the ciphertext.
<b>OFB</b>	Perform output feedback mode encryption. The ciphertext can be any length. The plaintext will have the same length as the ciphertext.
<b>PKCS-PAD</b>	For cipher block chaining. The ciphertext must be an exact multiple of the block size (8 bytes for DES and 16 bytes for AES). The plaintext will be between 1 and the blocksize (8 bytes for DES, 16 bytes for AES) bytes shorter than the ciphertext. This process rule always pads the ciphertext so that ciphertext produced is an exact multiple of the blocksize, even if the plaintext was already a multiple of the blocksize.

The Advanced Encryption Standard (AES) and Data Encryption Standard (DES) are supported. AES encryption uses a 128-, 192-, or 256-bit key. DES encryption uses a 56-, 112-, or 168-bit key. See the processing rule descriptions for limitations. For each algorithm, certain processing rules are not allowed. See the `rule_array` parameter description for more information.

All modes except ECB use an initial chaining vector (ICV) in their processing.

## Symmetric Key Decipher

All modes that utilize chaining produce a resulting chaining value called the output chaining vector (OCV). The application can pass the OCV as the ICV in the next decipher call. This results in record chaining.

The selection between single-DES decryption mode and triple-DES decryption mode is controlled by the length of the key supplied in the *key\_identifier* parameter. If a single-length key is supplied, single-DES decryption is performed. If a double-length or triple-length key is supplied, triple-DES decryption is performed.

The key may be specified as a clear key value, an internal clear key token, or the label name of a clear key or an encrypted key in the CKDS.

### Choosing Between CSNBSYD and CSNBSYD1

CSNBSYD and CSNBSYD1 provide identical functions. When choosing which service to use, consider this:

- **CSNBSYD** requires the ciphertext and plaintext to reside in the caller's primary address space. Also, a program using CSNBSYD adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface. The callable service name for AMODE(64) invocation is CSNESYD.
- **CSNBSYD1** allows the ciphertext and plaintext to reside either in the caller's primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using CSNBSYD1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSYD1, *cipher\_text\_id* and *clear\_text\_id* are access list entry token (ALET) parameters of the data spaces containing the ciphertext and plaintext.

The callable service name for AMODE(64) invocation is CSNESYD1.

### Format

```
CALL CSNBSYD(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_identifier_length,  
    key_identifier,  
    key_parms_length,  
    key_parms,  
    block_size,  
    initialization_vector_length,  
    initialization_vector,  
    chain_data_length,  
    chain_data,  
    cipher_text_length,  
    cipher_text,  
    clear_text_length,  
    clear_text,  
    optional_data_length,  
    optional_data)
```

```
CALL CSNBSYD1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    key_parms_length,
    key_parms,
    block_size,
    initialization_vector_length,
    initialization_vector,
    chain_data_length,
    chain_data,
    cipher_text_length,
    cipher_text,
    clear_text_length,
    clear_text,
    optional_data_length,
    optional_data,
    cipher_text_id,
    clear_text_id)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **rule\_array\_count**

Direction: Input

Type: Integer



## Symmetric Key Decipher

The number of keywords you supplied in the *rule\_array* parameter. The value may be 1, 2, 3 or 4.

### **rule\_array**

Direction: Input

Type: String

An array of 8-byte keywords providing the processing control information. The keywords must be in contiguous storage, left-justified and padded on the right with blanks.

Table 151. Symmetric Key Decipher Rule Array Keywords

Keyword	Meaning
<b>Algorithm (required)</b>	
AES	Specifies that the Advanced Encryption Standard (AES) algorithm is to be used. The block size is 16 bytes. The key length may be 16, 24, or 32 bytes. The <i>chain_data</i> field must be at least 32 bytes in length. The OCV is the first 16 bytes in the <i>chain_data</i> . AES does not support the CUSP, IPS, or X9.23 processing rules.
DES	Specifies that the Data Encryption Standard (DES) algorithm is to be used. The algorithm, DES or TDES, will be determined from the length of the key supplied. The key length may be 8, 16, or 24. The block size is 8 bytes. The <i>chain_data</i> field must be at least 16 bytes in length. The OCV is the first eight bytes in the <i>chain_data</i> . DES does not support the GCM processing rule.
<b>Processing Rule (optional)</b>	
Rules CBC-CS, CUSP, IPS, PKCS-PAD, and X9.23 should be specified only when there is one request or on the last request of a sequence of chained requests.	
CBC	Performs cipher block chaining. The text length must be a multiple of the block size for the specified algorithm. CBC is the default value.
CBC-CS	CBC mode (cipher block chaining) with ciphertext stealing. Input text may be any length.
CFB	CFB mode (cipher feedback) that is compatible with IBM's Encryption Facility product. Input text may be any length.
CFB-LCFB	CFB mode (cipher feedback). This rule allows the value of <i>s</i> (the segment size) to be something other than the block size ( <i>s</i> is set to the block size with the CFB processing rule). <i>key_parms_length</i> and <i>key_parms</i> are used to set the value of <i>s</i> . Input text may be any length.
CUSP	CBC mode (cipher block chaining) that is compatible with IBM's CUSP and PCF products. Input text may be any length.
ECB	Performs electronic code book encryption. The text length must be a multiple of the block size for the specified algorithm.

Table 151. Symmetric Key Decipher Rule Array Keywords (continued)

Keyword	Meaning
GCM	GCM (Galois/Counter Mode). <i>key_parms_length</i> and <i>key_parms</i> are used to indicate the length of the tag (the value <i>t</i> ) on input and contain the tag on output. Additional Authenticated Data (AAD) is contained in <i>optional_data_length</i> and <i>optional_data</i> . Input text may be any length. GCM does not support chaining, so CONTINUE and FINAL are not allowed for the ICV Selection rule.
IPS	CBC mode (cipher block chaining) that is compatible with IBM's IPS product. Input text may be any length.
OFB	OFB mode (output feedback). Input text may be any length.
PKCS-PAD	CBC mode (cipher block chaining) but the ciphertext must be an exact multiple of the block length (8 bytes for DES and 16 bytes for AES). The plaintext will be 1 to 8 bytes shorter for DES and 1 to 16 bytes shorter for AES than the ciphertext.
X9.23	CBC mode (cipher block chaining) for 1 to 8 bytes of padding dropped from the output clear text.
<b>Key Rule (optional)</b>	
KEY-CLR	This specifies that the key parameter contains a clear key value. KEY-CLR is the default value.
KEYIDENT	This specifies that the <i>key_identifier</i> field will be an internal clear token, or the label name of a clear key or encrypted key in the CKDS. Normal CKDS labelname syntax is required.
<b>ICV Selection (optional)</b>	
INITIAL	This specifies taking the initialization vector from the <i>initialization_vector</i> parameter. INITIAL is the default value. INITIAL is not valid with processing rule GCM.
CONTINUE	This specifies taking the initialization vector from the output chaining vector contained in the work area to which the <i>chain_data</i> parameter points. CONTINUE is not valid for processing rules ECB, GCM, or X9.23.
FINAL	This specifies taking the initialization vector from the output chaining vector contained in the work area to which the <i>chain_data</i> parameter points. Using FINAL indicates that this call contains the last portion of data. FINAL is valid for processing rules CBC-CS, CFB, CFB-LCBF, and OFB.
ONLY	This specifies taking the initialization vector from the <i>initialization_vector</i> parameter and that the entirety of the data to be processed is in this single call. ONLY is valid for processing rules CBC-CS, CFB, CFB-LCFB, GCM, and OFB.

**key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *key\_identifier* parameter. For clear keys, the length is in bytes and includes only the value of the key. The maximum size is 256 bytes.

For the KEYIDENT keyword, this parameter value must be 64.

**key\_identifier**

## Symmetric Key Decipher

Direction: Input

Type: String

For the KEY-CLR keyword, this specifies the cipher key. The parameter must be left justified.

For the KEYIDENT keyword, this specifies an internal clear token, or the label name of a clear key or an encrypted key in the CKDS. Normal CKDS labelname syntax is required. KEYIDENT is valid with DES and AES.

### **key\_parms\_length**

Direction: Input

Type: Integer

The length of the *key\_parms* parameter.

- For the CFB-LCFB processing rule, this length must be 1.
- For the GCM processing rule, this is the length in bytes of the authentication tag to be verified. Valid lengths are 4, 8, 12, 13, 14, 15, 16. Using a length of 4 or 8 is stringly discouraged.
- For all other processing rules, this field is ignored.

You must specify the same length used when enciphering the text.

### **key\_parms**

Direction: Input

Type: String

This parameter contains key-related parameters specific to the encryption algorithm and processing mode.

- For the CFB-LCFB processing rule, this 1-byte field specifies the segment size in bytes. Valid values are 1 to the block size, inclusive. The block size is eight for DES and sixteen for AES.
- For the GCM processing rule, this contains the authentication tag for the provided ciphertext (*cipher\_text* parameter) and additional authenticated data (*optional\_data* parameter).
- For all other processing rules, this field is ignored.

For the modes where *key\_parms* is used, you must specify the same *key\_parms* used when enciphering the text using the Symmetric Key Encipher.

### **block\_size**

Direction: Input

Type: Integer

This parameter contains the processing size of the text block in bytes. This value will be algorithm specific. Be sure to specify the same block size as used to encipher the text.

### **initialization\_vector\_length**

Direction: Input

Type: Integer

The length of the *initialization\_vector* parameter. This parameter is ignored for the ECB processing rule. For the GCM processing rule, NIST recommends a length of 12, but tolerates any non-zero length. For all other processing rules, the length should be equal to the block length for the algorithm specified.

### **initialization\_vector**

Direction: Input

Type: String

This initialization chaining value. You must use the same ICV that was used to encipher the data. This parameter is ignored for the ECB processing rule.

**chain\_data\_length**

Direction: Input/Output

Type: Integer

The length of the *chain\_data* parameter. On output, the actual length of the chaining vector will be stored in the parameter. This parameter is ignored if the ICV selection keyword is ONLY.

**chain\_data**

Direction: Input/Output

Type: String

This field is used as a system work area for the chaining vector. Your application program must not change the data in this string. The chaining vector holds the output chaining vector from the caller.

The direction is output if the ICV selection keyword is INITIAL. This parameter is ignored if the ICV selection keyword is ONLY.

The mapping of the *chain\_data* depends on the algorithm specified. For AES, the *chain\_data* field must be at least 32 bytes in length. The OCV is in the first 16 bytes in the *chain\_data*. For DES, *chain\_data* field must be at least 16 bytes in length.

**cipher\_text\_length**

Direction: Input

Type: Integer

The length of the ciphertext. A zero value in the *cipher\_text\_length* parameter is not valid except with the GCM processing rule when performing a GMAC operation. The length must be a multiple of the algorithm block size for the CBC, ECB, and PKCS-PAD processing rules, but may be any length with the other processing rules.

**cipher\_text**

Direction: Input

Type: String

The text to be deciphered.

**clear\_text\_length**

Direction: Input/Output

Type: Integer

On input, this parameter specifies the size of the storage pointed to by the *clear\_text* parameter. On output, this parameter has the actual length of the text stored in the *clear\_text* parameter. The *clear\_text* parameter must be at least the same length as the *cipher\_text* parameter, except for the PKCS-PAD and X9.23 processing rules, where the padding is automatically dropped on output.

**clear\_text**

Direction: Output

Type: String

## Symmetric Key Decipher

The deciphered text the service returns.

### **optional\_data\_length**

Direction: Input

Type: Integer

The length of the *optional\_data* parameter. For the GCM processing rule, this parameter contains the length of the Additional Authenticated Data (AAD). For all other processing rules, this field is ignored.

### **optional\_data**

Direction: Input

Type: String

Optional data required by a specified algorithm or processing mode. For the GCM processing rule, this parameter contains the Additional Authenticated Data (AAD). For all other processing rules, this field is ignored.

You must specify the same *optional\_data* used when enciphering the text using Symmetric Key Encipher.

### **cipher\_text\_id**

Direction: Input

Type: Integer

For CSNBSYD1 only, the ALET of the ciphertext to be deciphered.

### **clear\_text\_id**

Direction: Input

Type: Integer

For CSNBSYD1 only, the ALET of the clear text supplied by the application.

## Usage Notes

- SAF may be invoked to verify the caller is authorized to use the specified key label stored in the CKDS.
- To use a CKDS encrypted key, the ICSF segment of the CSFKEYS class general resource profile associated with the specified key label must contain SYMCPACFWRAP(YES).
- No pre- or post-processing exits are enabled for this service.
- The master keys need to be loaded only when using this service with encrypted key labels.
- The AES algorithm will use hardware if it is available. Otherwise, clear key operations will be performed in software.
- AES has the same availability restrictions as triple-DES.
- This service will fail if execution would cause destructive overlay of the *cipher\_text* field.

## Access Control Points

When the label of an encrypted key is specified for the *key\_identifier* parameter, the appropriate access control point listed below must be enabled.

Table 152. Required access control points for Symmetric Key Decipher

Key algorithm	Access control point
AES	Symmetric Key Encipher/Decipher - Encrypted AES keys
DES	Symmetric Key Encipher/Decipher - Encrypted DES keys

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 153. Symmetric Key Decipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	DES keyword is not supported.  CFB-LCFB, GCM, and OFB processing rules are not supported.
IBM @server zSeries 990 IBM @server zSeries 890	CP Assist for Cryptographic Functions	CFB-LCFB, GCM, and OFB processing rules are not supported.
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	CFB-LCFB, GCM, and OFB processing rules are not supported.
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions  Crypto Express3 Coprocessor	CFB-LCFB, GCM, and OFB processing rules are not supported.  Encrypted keys require CEX3C with the Nov. 2009 or later licensed internal code (LIC).
z196 z114	CP Assist for Cryptographic Functions  Crypto Express3 Coprocessor	Encrypted keys require CEX3C with the Nov. 2009 or later licensed internal code (LIC).
IBM zEnterprise EC12	CP Assist for Cryptographic Functions  Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## Related Information

You **cannot** overlap the plaintext and ciphertext fields. For example:

```
pppppp
cccccc is not supported.
```

## Symmetric Key Decipher

cccccc  
pppppp is not supported.

ppppppcccccc is supported.

P represents the plaintext and c represents the ciphertext.

“Cipher Processing Rules” on page 940 discusses the cipher processing rules.

---

## Symmetric Key Encipher (CSNBSYE or CSNBSYE1 and CSNESYE or CSNESYE1)

Use the symmetric key encipher callable service to encipher data using one of the supported modes. ICSF supports several processing rules to encipher data. You choose the type of processing rule that the Symmetric Key Encipher callable service should use for the block chaining. See “Modes of Operation” on page 351 for more information.

Processing Rule	Purpose
ANSI X9.23	For cipher block chaining. The ciphertext must be an exact multiple of the block size for the specified algorithm (8 bytes for DES). The plaintext will be between 1 and 8 bytes shorter than the ciphertext. This process rule always pads the plaintext during encryption so that ciphertext produced is an exact multiple of the block size, even if the plaintext was already a multiple of the blocksize.
CBC	For cipher block chaining. The ciphertext must be an exact multiple of the block size for the specified algorithm (8 bytes for DES, 16 bytes for AES). The plaintext will have the same length as the ciphertext.
CBC-CS	For cipher block chaining. The ciphertext can be any length. The plaintext will have the same length as the ciphertext.
CFB	Performs cipher feedback encryption with the segment size equal to the block size. The ciphertext can be of any length. The plaintext will have the same length as the ciphertext.
CFB-LCFB	Performs cipher feedback encryption with the segment size set by the caller. The ciphertext can be of any length. The plaintext will have the same length as the ciphertext.
CUSP	For cipher block chaining. The ciphertext can be of any length. The plaintext will have the same length as the ciphertext.
ECB	Performs electronic code book encryption. The ciphertext must be an exact multiple of the block size for the specified algorithm (8 bytes for DES, 16 bytes for AES). The plaintext will have the same length as the ciphertext.
GCM	Perform Galois/Counter mode decryption, which provides both confidentiality and authentication for

	the plaintext and authentication for the additional authenticated data (AAD). The ciphertext can be any length. The plaintext will have the same length as the ciphertext. Additionally, the authentication tag will be verified before any data is returned.
<b>IPS</b>	For cipher block chaining. The ciphertext can be any length. The plaintext will have the same length as the ciphertext.
<b>OFB</b>	Perform output feedback mode encryption. The ciphertext can be any length. The plaintext will have the same length as the ciphertext.
<b>PKCS-PAD</b>	For cipher block chaining. The ciphertext must be an exact multiple of the block size (8 bytes for DES and 16 bytes for AES). The plaintext will be between 1 and the blocksize (8 bytes for DES, 16 bytes for AES) bytes shorter than the ciphertext. This process rule always pads the ciphertext so that ciphertext produced is an exact multiple of the blocksize, even if the plaintext was already a multiple of the blocksize.

The Advanced Encryption Standard (AES) and Data Encryption Standard (DES) are supported. AES encryption uses a 128-, 192-, or 256-bit key. The CBC, CBC-CS, CFB, CFB-LCFB, ECB, GCM, OFB, and XTS-AES modes are supported.

All modes except ECB and XTS-AES use an initial chaining vector (ICV) in their processing.

All modes that tolerate chaining produce a resulting chaining value called the output chaining vector (OCV). The application can pass the OCV as the ICV in the next encipher call. This results in record chaining.

The selection between single-DES decryption mode and triple-DES decryption mode is controlled by the length of the key supplied in the *key\_identifier* parameter. If a single-length key is supplied, single-DES decryption is performed. If a double-length or triple-length key is supplied, triple-DES decryption is performed.

The key may be specified as a clear key value, an internal clear key token, or the label name of a clear key or an encrypted key in the CKDS.

## Choosing between CSNBSYE and CSNBSYE1

CSNBSYE and CSNBSYE1 provide identical functions. When choosing which service to use, consider this:

- **CSNBSYE** requires the cleartext and ciphertext to reside in the caller's primary address space. Also, a program using CSNBSYE adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface. The callable service name for AMODE(64) invocation is CSNESYE.
- **CSNBSYE1** allows the cleartext and ciphertext to reside either in the caller's primary address space or in a data space. This can allow you to encipher more data with one call. However, a program using CSNBSYE1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface.



## Symmetric Key Encipher

For CSNBSYE1, *clear\_text\_id* and *cipher\_text\_id* are access list entry token (ALET) parameters of the data spaces containing the cleartext and ciphertext.  
The callable service name for AMODE(64) invocation is CSNESYE1.

### Format

```
CALL CSNBSYE(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_identifier_length,  
    key_identifier,  
    key_parms_length,  
    key_parms,  
    block_size,  
    initialization_vector_length,  
    initialization_vector,  
    chain_data_length,  
    chain_data,  
    clear_text_length,  
    clear_text,  
    cipher_text_length,  
    cipher_text,  
    optional_data_length,  
    optional_data)
```

```
CALL CSNBSYE1(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_identifier_length,  
    key_identifier,  
    key_parms_length,  
    key_parms,  
    block_size,  
    initialization_vector_length,  
    initialization_vector,  
    chain_data_length,  
    chain_data,  
    clear_text_length,  
    clear_text,  
    cipher_text_length,  
    cipher_text,  
    optional_data_length,  
    optional_data,  
    clear_text_id,  
    cipher_text_id)
```

### Parameters

**return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

**reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

**exit\_data**

Direction: Ignored

Type: String

This field is ignored.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value may be 1, 2, 3 or 4.

**rule\_array**

Direction: Input

Type: String

An array of 8-byte keywords providing the processing control information. The keywords must be in contiguous storage, left-justified and padded on the right with blanks.

Table 154. Symmetric Key Encipher Rule Array Keywords

Keyword	Meaning
<i>Algorithm (required)</i>	
AES	Specifies that the Advanced Encryption Standard (AES) algorithm is to be used. The block size is 16 bytes. The key length may be 16, 24, or 32 bytes. The <i>chain_data</i> field must be at least 32 bytes in length. The OCV is the first 16 bytes in the <i>chain_data</i> . AES does not support the CUSP, IPS, or X9.23 processing rules.
DES	Specifies that the Data Encryption Standard (DES) algorithm is to be used. The algorithm, DES or TDES, will be determined from the length of the key supplied. The key length may be 8, 16, or 24. The block size is 8 bytes. The <i>chain_data</i> field must be at least 16 bytes in length. The OCV is the first eight bytes in the <i>chain_data</i> . DES does not support the GCM processing rule.

## Symmetric Key Encipher

Table 154. Symmetric Key Encipher Rule Array Keywords (continued)

Keyword	Meaning
<b>Processing Rule (optional)</b>	
Rules CBC-CS, CUSP, IPS, PKCS-PAD, and X9.23 should be specified only when there is one request or on the last request of a sequence of chained requests.	
CBC	CBC mode (cipher block chaining). The text length must be a multiple of the block size for the specified algorithm. CBC is the default value.
CBC-CS	CBC mode (cipher block chaining) with ciphertext stealing. Input text may be any length.
CFB	CFB mode (cipher feedback) that is compatible with IBM's Encryption Facility product. Input text may be any length.
CFB-LCFB	CFB mode (cipher feedback). This rule allows the value of <i>s</i> (the segment size) to be something other than the block size ( <i>s</i> is set to the block size with the CFB processing rule). The <i>key_parms_length</i> and <i>key_parms</i> parameters are used to set the value of <i>s</i> . Input text may be any length.
CUSP	CBC mode (cipher block chaining) that is compatible with IBM's CUSP and PCF products. Input text may be any length.
ECB	ECB mode (electronic codebook). The text length must be a multiple of the block size for the specified algorithm.
GCM	GCM mode (Galois/Counter Mode). The <i>key_parms_length</i> and <i>key_parms</i> parameters are used to indicate the length of the tag (the value <i>t</i> ) on input and contain the tag on output. Additional Authenticated Data (AAD) is contained in the <i>optional_data_length</i> and <i>optional_data</i> parameters. Input text may be any length.
IPS	CBC mode (cipher block chaining) that is compatible with IBM's IPS product. Input text may be any length.
OFB	OFB mode (output feedback). Input text may be any length.
PKCS-PAD	CBC mode (cipher block chaining) not necessarily in exact multiples of the block length (8 bytes for DES and 16 bytes for AES). PKCS-PAD always pads the plaintext so that the ciphertext produced is an exact multiple of the block length and longer than the plaintext.
X9.23	CBC mode (cipher block chaining) for 1 to 8 bytes of padding added according to ANSI X9.23. Input text may be any length.
<b>Key Rule (optional)</b>	
KEY-CLR	This specifies that the key parameter contains a clear key value. KEY-CLR is the default.
KEYIDENT	This specifies that the <i>key_identifier</i> field will be an internal clear token, or the label name of a clear key or encrypted key in the CKDS. Normal CKDS labelname syntax is required.
<b>ICV Selection (optional)</b>	
INITIAL	This specifies taking the initialization vector from the <i>initialization_vector</i> parameter. INITIAL is the default value. INITIAL is not valid with processing rule GCM.

Table 154. Symmetric Key Encipher Rule Array Keywords (continued)

Keyword	Meaning
CONTINUE	This specifies taking the initialization vector from the output chaining vector contained in the work area to which the <i>chain_data</i> parameter points. CONTINUE is not valid for processing rules ECB, GCM, or X9.23.
FINAL	This specifies taking the initialization vector from the output chaining vector contained in the work area to which the <i>chain_data</i> parameter points. Using FINAL indicates that this call contains the last portion of data. FINAL is valid for processing rules CBC-CS, CFB, CFB-LCFB, and OFB.
ONLY	This specifies taking the initialization vector from the <i>initialization_vector</i> parameter and that the entirety of the data to be processed is in this single call. ONLY is valid for processing rules CBC-CS, CFB, CFB-LCFB, GCM, and OFB.

**key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *key\_identifier* parameter. For clear keys, the length is in bytes and includes only the value of the key.

For the KEYIDENT keyword, this parameter value must be 64.

**key\_identifier**

Direction: Input

Type: String

For the KEY-CLR keyword, this specifies the cipher key. The parameter must be left justified.

For the KEYIDENT keyword, this specifies a internal clear token, or the label name of a clear key or an encrypted key in the CKDS. Normal CKDS label name syntax is required.

**key\_parms\_length**

Direction: Input

Type: Integer

The length of the *key\_parms* parameter.

- For the CFB-LCFB processing rule, this length must be 1.
- For the GCM processing rule, this is the length in bytes of the authentication tag to be generated. Valid lengths are 4, 8, 12, 13, 14, 15, 16. Using a length of 4 or 8 is strongly discouraged.
- For all other processing rules, this field is ignored.

When deciphering the text, you must specify this same length.

**key\_parms**

Direction: Input/Output

Type: String

This parameter contains key-related parameters specific to the encryption algorithm and processing mode.

## Symmetric Key Encipher

- For the CFB-LCFB processing rule, this 1-byte field specifies the segment size in bytes. Valid values are 1 to the blocksize, inclusive. The blocksize is eight for DES and sixteen for AES.
- For the GCM processing rule, this will contain the generated authentication tag for the provided plaintext (*plain\_text* parameter) and additional authenticated data (*optional\_data* parameter).
- For all other processing rules, this field is ignored.

For the modes where *key\_parms* is used, you must specify the same *key\_parms* when deciphering the text using the Symmetric Key Decipher callable service.

### **block\_size**

Direction: Input

Type: Integer

This parameter contains the processing size of the text block in bytes. This value will be algorithm specific.

### **initialization\_vector\_length**

Direction: Input

Type: Integer

The length of the *initialization\_vector* parameter. This parameter is ignored for the ECB processing rule. For the GCM processing rule, NIST recommends a length of 12, but tolerates any non-zero length. For all other processing rules, the length should be equal to the block length for the algorithm specified.

### **initialization\_vector**

Direction: Input

Type: String

This initialization chaining value. You must use the same ICV to decipher the data. This parameter is ignored for the ECB processing rule.

### **chain\_data\_length**

Direction: Input/Output

Type: Integer

The length of the *chain\_data* parameter. On output, the actual length of the chaining vector will be stored in the parameter. This parameter is ignored if the ICV selection keyword is ONLY.

### **chain\_data**

Direction: Input/Output

Type: String

This field is used as a system work area for the chaining vector. Your application program must not change the data in this string. The chaining vector holds the output chaining vector from the caller.

The direction is output if the ICV selection keyword is INITIAL. This parameter is ignored if the ICV selection keyword is ONLY.

The mapping of the *chain\_data* depends on the algorithm specified. For AES, the *chain\_data* field must be at least 32 bytes in length. The OCV is in the first 16 bytes in the *chain\_data*. For DES, the *chain\_data* field must be at least 16 bytes in length.

### **clear\_text\_length**

Direction: Input

Type: Integer

The length of the cleartext. A zero value in the *clear\_text\_length* parameter is not valid except with the GCM processing rule when performing a GMAC operation. The length must be a multiple of the algorithm block size for the CBC, ECB, and PKCS-PAD processing rules, but may be any length with the other processing rules. For the processing rules that support partial blocks (or segments for CFB-LCFB), it is recommended that is the final block (or segment) be the only one that is partial. Having a partial block in the middle is not a supported operation as defined by the standards documents and may not be portable to other encryption systems.

**clear\_text**

Direction: Input

Type: String

The text to be enciphered.

**cipher\_text\_length**

Direction: Input/Output

Type: Integer

On input, this parameter specifies the size of the storage pointed to by the *cipher\_text* parameter. On output, this parameter has the actual length of the text stored in the buffer addressed by the *cipher\_text* parameter.

**cipher\_text**

Direction: Output

Type: String

The enciphered text the service returns.

**optional\_data\_length**

Direction: Input

Type: Integer

The length of the *optional\_data* parameter. For the GCM processing rule, this parameter contains the length of the Additional Authenticated Data (AAD), and may be any length, including zero. For all other processing rules, this field is ignored.

**optional\_data**

Direction: Input

Type: String

Optional data required by a specified algorithm. Optional data required by a specified algorithm or processing mode. For the GCM processing rule, this parameter contains the Additional Authenticated Data (AAD). For all other processing rules, this field is ignored.

You must specify the same *optional\_data* when deciphering the text using Symmetric Key Decipher.

**clear\_text\_id**

Direction: Input

Type: Integer

For CSNBSYE1 only, the ALET of the clear text to be enciphered.

## Symmetric Key Encipher

**cipher\_text\_id**

Direction: Input

Type: Integer

For CSNBSYE1 only, the ALET of the ciphertext that the application supplied.

### Usage Notes

- SAF may be invoked to verify the caller is authorized to use the specified key label stored in the CKDS.
- To use a CKDS encrypted key, the ICSF segment of the CSFKEYS class general resource profile associated with the specified key label must contain SYMCPACFWRAP(YES).
- No pre- or post-processing exits are enabled for this service.
- The master keys need to be loaded only when using this service with the encrypted key labels.
- The AES algorithm will use hardware if it is available. Otherwise, clear key operations will be performed in software.
- AES has the same availability restrictions as triple-DES.
- This service will fail if execution would cause destructive overlay of the *clear\_text* field.

### Access Control Points

When the label of an encrypted key is specified for the *key\_identifier* parameter, the appropriate access control point listed below must be enabled.

Table 155. Required access control points for Symmetric Key Encipher

Key algorithm	Access control point
AES	Symmetric Key Encipher/Decipher - Encrypted AES keys
DES	Symmetric Key Encipher/Decipher - Encrypted DES keys

### Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 156. Symmetric Key Encipher required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		DES keyword is not supported.  CFB-LCFB, GCM, and OFB processing rules are not supported.
IBM @server zSeries 990 IBM @server zSeries 890	CP Assist for Cryptographic Functions	CFB-LCFB, GCM, and OFB processing rules are not supported.

Table 156. Symmetric Key Encipher required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	CFB-LCFB, GCM, and OFB processing rules are not supported.
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions  Crypto Express3 Coprocessor	CFB-LCFB, GCM, and OFB processing rules are not supported.  Encrypted keys require the CEX3C with the Nov. 2009 or later licensed internal code (LIC).
z196  z114	CP Assist for Cryptographic Functions  Crypto Express3 Coprocessor	CFB-LCFB, GCM, and OFB processing rules are not supported.  Encrypted keys require the CEX3C with the Nov. 2009 or later licensed internal code (LIC).
IBM zEnterprise EC12	CP Assist for Cryptographic Functions  Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	CFB-LCFB, GCM, and OFB processing rules are not supported.

## Related Information

You **cannot** overlap the plaintext and ciphertext fields. For example:

```
pppppp
  ccccc  is not supported.
```

```
cccccc
  pppppp  is not supported.
```

```
ppppppcccccc is supported.
```

P represents the plaintext and c represents the ciphertext.

The method used to produce the OCV is the same with the CBC and X9.23 processing rules. However, that method is different from the method used by the CUSP and IPS processing rules.

“Cipher Processing Rules” on page 940 discusses the cipher processing rules.



## Symmetric Key Encipher

---

## Chapter 7. Verifying Data Integrity and Authenticating Messages

ICSF provides several methods to verify the integrity of transmitted messages and stored data:

- Message authentication code (MAC)
- Hash functions, including modification detection code (MDC) processing and one-way hash generation

**Note:** You can also use digital signatures (see Chapter 9, “Using Digital Signatures,” on page 557) to authenticate messages.

The choice of callable service depends on the security requirements of the environment in which you are operating. If you need to ensure the authenticity of the sender as well as the integrity of the data, and both the sender and receiver can share a secret key, consider message authentication code processing. If you need to ensure the integrity of transmitted data in an environment where it is not possible for the sender and the receiver to share a secret cryptographic key, consider hashing functions, such as the modification detection code process.

The callable services are described in the following topics:

- “HMAC Generate (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1)” on page 425
- “HMAC Verify (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1)” on page 429
- “MAC Generate (CSNBMG or CSNBMG1 and CSNEMG or CSNEMG1)” on page 434
- “MAC Verify (CSNBMG or CSNBMG1 and CSNEMG or CSNEMG1)” on page 440
- “MDC Generate (CSNBMDG or CSNBMDG1 and CSNEMDG or CSNEMDG1)” on page 445
- “One-Way Hash Generate (CSNBOWH or CSNBOWH1 and CSNEOWH or CSNEOWH1)” on page 450
- “Symmetric MAC Generate (CSNBSMG or CSNBSMG1 and CSNESMG or CSNESMG1)” on page 454
- “Symmetric MAC Verify (CSNBSMG or CSNBSMG1 and CSNESMG or CSNESMG1)” on page 459

---

### How MACs are Used

When a message is sent, an application program can generate an authentication code for it using the MAC generation callable service. ICSF supports the ANSI X9.9-1 basic procedure and both the ANSI X9.19 basic procedure and optional double key MAC procedure. The service computes the text of the message authentication code using the algorithm and a key. The ANSI X9.9-1 or ANSI X9.19 basic procedures accept either a single-length MAC generation (MAC) key or a data-encrypting (DATA) key, and the message text. The ANSI X9.19 optional double key MAC procedure accepts a double-length MAC key and the message text. The message text may be in clear or encrypted form. The originator of the message sends the MAC with the message text.

When the receiver gets the message, an application program calls the *MAC verification callable service*. The callable service generates a MAC using the same algorithm as the sender and either the single-length or double-length MAC verification key, the single-length or double-length MAC generation key, or DATA key, and the message text. The MACVER callable service compares the MAC it generates with the one sent with the message and issues a return code that indicates whether the MACs match. If the return code indicates that the MACs match, the receiver can accept the message as genuine and unaltered. If the return code indicates that the MACs do not match, the receiver can assume that the message is either bogus or has been altered. The newly computed MAC is not revealed outside the cryptographic feature.

In a similar manner, MACs can be used to ensure the integrity of data stored on the system or on removable media, such as tape.

Secure use of the MAC generation and MAC verification services requires the use of MAC and MACVER keys in these services, respectively. To accomplish this, the originator of the message generates a MAC/MACVER key pair, uses the MAC key in the MAC generation service, and exports the MACVER key to the receiver. The originator of the message enforces key separation on the link by encrypting the MACVER key under a transport key that is not an NOCV key before exporting the key to the receiver. With this type of key separation enforced, the receiver can only receive a MACVER key and can use only this key in the MAC verification service. This ensures that the receiver cannot alter the message and produce a valid MAC with the altered message. These security features are not present if DATA keys are used in the MAC generation service, or if DATA or MAC keys are used in the MAC verification service.

By using MACs, you get the following benefits:

- **For data transmitted over a network**, you can validate the authenticity of the message as well as ensure that the data has not been altered during transmission. For example, an active eavesdropper can tap into a transmission line, and interject bogus messages or alter sensitive data being transmitted. If the data is accompanied by a MAC, the recipient can use a callable service to detect whether the data has been altered. Since both the sender and receiver share a secret key, the receiver can use a callable service that calculates a MAC on the received message and compares it to the MAC transmitted with the message. If the comparison is equal, the message may be accepted as unaltered. Furthermore, since the shared key is secret, when a MAC is verified it can be assumed that the sender was, in fact, the other person who knew the secret key.
- **For data stored on tape or DASD**, you can ensure that the data read back onto the system was the same as the data written onto the tape or DASD. For example, someone might be able to bypass access controls. Such an access might escape the notice of auditors. However, if a MAC is stored with the data, and verified when the data is read, you can detect alterations to the data.

---

## How Hashing Functions Are Used

Hashing functions include the MDC and one-way hash. You need to hash text before submitting it to digital signature services (see Chapter 9, "Using Digital Signatures," on page 557).

### How MDCs Are Used

When a message is sent, an application program can generate a modification detection code for it using the *MDC generation callable service*. The service computes

the modification detection code, a 128-bit value, using a one-way cryptographic function and the message text (which itself may be in clear or encrypted form). The originator of the message ensures that the MDC is transmitted with integrity to the intended receiver of the message. For example, the MDC could be published in a reliable source of public information.

When the receiver gets the message, an application program calls the *MDC callable service*. The callable service generates an MDC by using the same one-way cryptographic function and the message text. The application program can compare the new MDC with the one generated by the originator of the message. If the MDCs match, the receiver knows that the message was not altered.

In a similar manner, MDCs can be used to ensure the integrity of data stored on the system or on removable media, such as tape.

By using MDCs, you get the following benefits:

- **For data transmitted over a network between locations that do not share a secret key**, you can ensure that the data has not been altered during transmission. It is easy to compute an MDC for specific data, yet hard to find data that will result in a given MDC. In effect, the problem of ensuring the integrity of a large file is reduced to ensuring the integrity of a 128-bit value.
- **For data stored on tape or DASD**, you can ensure that the data read back onto the system was the same as the data written onto the tape or DASD. Once an MDC has been established for a file, the MDC generation callable service can be run at any later time on the file. The resulting MDC can be compared with the stored MDC to detect deliberate or inadvertent modification.

SHA-1 is a FIPS standard required for DSS. MD5 is a hashing algorithm used to derive Message Digests in Digital Signature applications.

---

## HMAC Generate (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1)

Use the HMAC generate callable service to generate a keyed hash message authentication code (MAC) for the text string provided as input.

The callable service names for AMODE(64) are CSNEHMG and CSFEHMG1.

### Choosing Between CSNBHMG and CSNBHMG1

CSNBHMG and CSNBHMG1 provide identical functions. When choosing which service to use, consider the following:

- CSNBHMG requires the application-supplied text to reside in the caller's primary address space.
- CSNBHMG1 allows the application-supplied text to reside either in the caller's primary address space or in a data space. This can allow you to process more data with one call. For CSNBHMG1, *text\_id\_in* is an access list entry token (ALET) parameter of the data space containing the application-supplied text.

## Format

```
CALL CSNBHMG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    text_length,
    text,
    chaining_vector_length,
    chaining_vector,
    mac_length,
    mac )
```

```
CALL CSNBHMG1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    text_length,
    text,
    chaining_vector_length,
    chaining_vector,
    mac_length,
    mac,
    text_id_in )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

**exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value may be 2 or 3.

**rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. The following table lists the keywords. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

*Table 157. Keywords for HMAC Generate Control Information*

Keyword	Meaning
<i>Token algorithm (One required)</i>	
HMAC	Specifies the HMAC algorithm to be used to generate the MAC.
<i>Hash method (One required)</i>	
SHA-1	Specifies the FIPS-198 HMAC procedure using the SHA-1 hash method, a symmetric key and text to produce a 20-byte (160-bit) MAC.
SHA-224	Specifies the FIPS-198 HMAC procedure using the SHA-224 hash method, a symmetric key and text to produce a 28-byte (224-bit) MAC.
SHA-256	Specifies the FIPS-198 HMAC procedure using the SHA-256 hash method, a symmetric key and text to produce a 32-byte (256-bit) MAC.
SHA-384	Specifies the FIPS-198 HMAC procedure using the SHA-384 hash method, a symmetric key and text to produce a 48-byte (384-bit) MAC.
SHA-512	Specifies the FIPS-198 HMAC procedure using the SHA-512 hash method, a symmetric key and text to produce a 64-byte (512-bit) MAC.
<i>Segmenting Control (One optional)</i>	
FIRST	First call, this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.
ONLY	Only call; segmenting is not employed by the application program. This is the default value.

**key\_identifier\_length**

## HMAC Generate

Direction: Input

Type: Integer

The length of the *key\_identifier* parameter. The maximum value is 725.

**key identifier**

Direction: Input/Output

Type: String

The 64-byte label or internal token of an encrypted HMAC key.

**text length**

Direction: Input

Type: Integer

The length of the text you supply in the *text* parameter. The maximum length of *text* is 214783647 bytes. For FIRST and MIDDLE calls, the *text\_length* must be a multiple of 64 for SHA-1, SHA-224 and SHA-256 and a multiple of 128 for SHA-384 and SHA-512 hash methods.

text

Direction: Input

Type: String

The application-supplied text for which the MAC is generated.

**chaining\_vector\_length**

Direction: Input/Output

Type: Integer

The length of the *chaining\_vector* in bytes. The value must be 128 bytes.

## chaining vector

Direction: Input/Output

Type: String

An 128-byte string that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter as binary zeros.

mac length

Direction: Input/Output

Type: Integer

The length of the *mac* parameter in bytes. This parameter is updated to the actual length of the *mac* parameter on output. The minimum value is 4, and the maximum value is 64.

mac

Direction: Output

Type: String

The field in which the callable service returns the MAC value if the segmenting rule is ONLY or LAST.

**text\_id\_in**

Direction: Input

Type: Integer

For CSNBHMG1 only, the ALET of the text for which the MAC is generated.

## Access Control Points

This table lists the access control points in the ICSF role that control the function for this service.

Table 158. HMAC Generate Access Control Points

Hash method	Access control point
SHA-1	HMAC Generate - SHA-1
SHA-224	HMAC Generate - SHA-224
SHA-256	HMAC Generate - SHA-256
SHA-384	HMAC Generate - SHA-384
SHA-512	HMAC Generate - SHA-512

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 159. HMAC generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This service is not supported.
IBM @server zSeries 990		This service is not supported.
IBM @server zSeries 890		
IBM System z9 EC		This service is not supported.
IBM System z9 BC		
IBM System z10 EC	Crypto Express2 Coprocessor	This service is not supported.
IBM System z10 BC	Crypto Express3 Coprocessor	HMAC keys not supported.
z196	Crypto Express3 Coprocessor	HMAC key support requires the Nov. 2010 or later licensed internal code (LIC).
z114		
IBM zEnterprise EC12	Crypto Express3 Coprocessor	
	Crypto Express4 Coprocessor	

## HMAC Verify (CSNBHMOV or CSNBHMOV1 and CSNEHMOV or CSNEHMOV1)

Use the HMAC verify callable service to verify a keyed hash message authentication code (MAC) for the text string provided as input.

The callable service names for AMODE(64) are CSNEHMOV and CSFEHMOV1.



## Choosing Between CSNBHBMV and CSNBHBMV1

CSNBHBMV and CSNBHBMV1 provide identical functions. When choosing which service to use, consider the following:

- CSNBHBMV requires the application-supplied text to reside in the caller's primary address space.
- CSNBHBMV1 allows the application-supplied text to reside either in the caller's primary address space or in a data space. This can allow you to process more data with one call. For CSNBHBMV1, *text\_id\_in* is an access list entry token (ALET) parameter of the data space containing the application-supplied text.

## Format

```
CALL CSNBHBMV(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    text_length,
    text,
    chaining_vector_length,
    chaining_vector,
    mac_length,
    mac )
```

```
CALL CSNBHBMV1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    text_length,
    text,
    chaining_vector_length,
    chaining_vector,
    mac_length,
    mac,
    text_id_in )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value may be 2 or 3.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. The following table lists the keywords. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

*Table 160. Keywords for HMAC Verify Control Information*

Keyword	Meaning
<i>Token algorithm (One required)</i>	
HMAC	Specifies the HMAC algorithm to be used to verify the MAC.
<i>Hash method (One required)</i>	
SHA-1	Specifies the FIPS-198 HMAC procedure using the SHA-1 hash method, a symmetric key and text to produce a 20-byte (160-bit) MAC.
SHA-224	Specifies the FIPS-198 HMAC procedure using the SHA-224 hash method, a symmetric key and text to produce a 28-byte (224-bit) MAC.
SHA-256	Specifies the FIPS-198 HMAC procedure using the SHA-256 hash method, a symmetric key and text to produce a 32-byte (256-bit) MAC.
SHA-384	Specifies the FIPS-198 HMAC procedure using the SHA-384 hash method, a symmetric key and text to produce a 48-byte (384-bit) MAC.
SHA-512	Specifies the FIPS-198 HMAC procedure using the SHA-512 hash method, a symmetric key and text to produce a 64-byte (512-bit) MAC.

Table 160. Keywords for HMAC Verify Control Information (continued)

Keyword	Meaning
<i>Segmenting Control (optional)</i>	
FIRST	First call, this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.
ONLY	Only call; segmenting is not employed by the application program. This is the default value.

**key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *key\_identifier* parameter. The maximum value is 725.

**key\_identifier**

Direction: Input/Output

Type: String

The 64-byte label or internal token of an encrypted HMAC or HMACVER key.

**text\_length**

Direction: Input

Type: Integer

The length of the text you supply in the *text* parameter. The maximum length of *text* is 214783647 bytes. For FIRST and MIDDLE calls, the *text\_length* must be a multiple of 64 for SHA-1, SHA-224 and SHA-256 and a multiple of 128 for SHA-384 and SHA-512 hash methods.

**text**

Direction: Input

Type: String

The application-supplied text for which the MAC is generated.

**chaining\_vector\_length**

Direction: Input/Output

Type: Integer

The length of the *chaining\_vector* in bytes. The value must be 128 bytes.

**chaining\_vector**

Direction: Input/Output

Type: String

An 128-byte string that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter as binary zeros.

**mac\_length**

Direction: Input

Type: Integer

The length of the *mac* parameter in bytes. The maximum value is 64.

**mac**

Direction: Input

Type: String

The field that contains the MAC value you want to verify.

**text\_id\_in**

Direction: Input

Type: Integer

For CSNBHVM1 only, the ALET of the text for which the MAC is generated.

## Access Control Points

This table lists the access control points in the ICSF role that control the function for this service.

Table 161. HMAC Verify Access Control Points

Hash method	Access control point
SHA-1	HMAC Generate - SHA-1
SHA-224	HMAC Generate - SHA-224
SHA-256	HMAC Generate - SHA-256
SHA-384	HMAC Generate - SHA-384
SHA-512	HMAC Generate - SHA-512

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 162. HMAC generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This service is not supported.
IBM @server zSeries 990		This service is not supported.
IBM @server zSeries 890		
IBM System z9 EC		This service is not supported.
IBM System z9 BC		
IBM System z10 EC	Crypto Express2 Coprocessor	This service is not supported.
IBM System z10 BC	Crypto Express3 Coprocessor	HMAC keys not supported.
z196	Crypto Express3 Coprocessor	HMAC key support requires the Nov. 2010 or later licensed internal code (LIC).
z114		

Table 162. HMAC generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM zEnterprise EC12	Crypto Express3 Coprocessor	
	Crypto Express4 Coprocessor	

## MAC Generate (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)

Use the MAC generate callable service to generate a 4-, 6-, or 8-byte message authentication code (MAC) for an application-supplied text string. You can specify that the callable service uses either the ANSI X9.9-1 procedure or the ANSI X9.19 optional double key MAC procedure to compute the MAC. For the ANSI X9.9-1 procedure you identify either a MAC generate key or a DATA key, and the message text. For the ANSI X9.19 optional double key MAC procedure, you identify a double-length MAC key and the message text.

The MAC generate callable service also supports the padding rules specified in the EMV Specification and ISO 16609. For the EMV MAC procedure, you identify a single- or double-length MAC key and the message text. For the ISO 16609 procedure you identify a double-length MAC or DATA key and the message text.

### Choosing Between CSNBMGN and CSNBMGN1

CSNBMGN and CSNBMGN1 provide identical functions. When choosing which service to use, consider the following:

- **CSNBMGN** requires the application-supplied text to reside in the caller's primary address space. Also, a program using CSNBMGN adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

The callable service name for AMODE(64) invocation is CSNEMGN.

- **CSNBMGN1** allows the application-supplied text to reside either in the caller's primary address space or in a data space. This can allow you to process more data with one call. However, a program using CSNBMGN1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified before it can run with other cryptographic products that follow this programming interface.

The callable service name for AMODE(64) invocation is CSNEMGN1.

For CSNBMGN1, *text\_id\_in* is an access list entry token (ALET) parameter of the data space containing the application-supplied text.

## Format

```
CALL CSNBMGN(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector,
    mac )
```

```
CALL CSNBMGN1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector,
    mac,
    text_id_in )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

## MAC Generate

### key\_identifier

Direction: Input/Output

Type: String

The 64-byte key label or internal key token that identifies a single or double-length MAC generate key, a DATAM key, or a single-length DATA key. The type of key depends on the MAC process rule in the *rule\_array* parameter.

### text\_length

Direction: Input

Type: Integer

The length of the text you supply in the *text* parameter. The maximum length of text is 214783647 bytes. If the *text\_length* is not a multiple of 8 bytes and if the ONLY or LAST keyword of the *rule\_array* parameter is called, the text is padded in accordance with the processing rule specified.

**Note:** The MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced.

### text

Direction: Input

Type: String

The application-supplied text for which the MAC is generated.

### rule\_array\_count

Direction: Input

Type: Integer

The number of keywords specified in the *rule\_array* parameter. The value can be 0, 1, 2, or 3.

### rule\_array

Direction: Input

Type: Character string

Zero to three keywords that provide control information to the callable service. The keywords are shown in Table 163. The keywords must be in 24 bytes of contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. For example,

'X9.9-1 MIDDLE MACLEN4 '

The order of the *rule\_array* keywords is not fixed.

You can specify one of the MAC processing rules and then choose one of the segmenting control keywords and one of the MAC length keywords.

Table 163. Keywords for MAC generate Control Information

Keyword	Meaning
<i>MAC Process Rules (optional)</i>	
EMVMAC	EMV padding rule with a single-length MAC key. The <i>key_identifier</i> parameter must identify a single-length MAC or a single-length DATA key. The text is always padded with 1 to 8 bytes so that the resulting text length is a multiple of 8 bytes. The first pad character is X'80'. The remaining 0 to 7 pad characters are X'00'.

Table 163. Keywords for MAC generate Control Information (continued)

Keyword	Meaning
EMVMACD	EMV padding rule with a double-length MAC key. The <i>key_identifier</i> parameter must identify a double-length MAC key. The padding rules are the same as for EMVMAC.
X9.19OPT	ANSI X9.19 optional double key MAC procedure. The <i>key_identifier</i> parameter must identify a double-length MAC key. The padding rules are the same as for X9.9-1.
X9.9-1	ANSI X9.9-1 and X9.19 basic procedure. The <i>key_identifier</i> parameter must identify a single-length MAC or a single-length DATA key. X9.9-1 causes the MAC to be computed from all of the data. The text is padded only if the text length is not a multiple of 8 bytes. If padding is required, the pad character 'X'00' is used. This is the default value.
TDES-MAC	ISO 16609 procedure. The <i>key_identifier</i> must identify a double-length MAC or a double-length DATA key. The text is padded only if the text length is not a multiple of 8 bytes.
<b>Segmenting Control (optional)</b>	
FIRST	First call, this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.
ONLY	Only call; segmenting is not employed by the application program. This is the default value.
<b>MAC Length and Presentation (optional)</b>	
HEX-8	Generates a 4-byte MAC value and presents it as 8 hexadecimal characters.
HEX-9	Generates a 4-byte MAC value and presents it as 2 groups of 4 hexadecimal characters with a space between the groups.
MACLEN4	Generates a 4-byte MAC value. This is the default value.
MACLEN6	Generates a 6-byte MAC value.
MACLEN8	Generates an 8-byte MAC value.

**chaining\_vector**

Direction: Input/Output

Type: String

An 18-byte string that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter as binary zeros.

**mac**

Direction: Output

Type: String

The 8-byte or 9-byte field in which the callable service returns the MAC value if the segmenting rule is ONLY or LAST. Allocate an 8-byte field for MAC



## MAC Generate

values of 4 bytes, 6 bytes, 8 bytes, or HEX-8. Allocate a 9-byte MAC field if you specify HEX-9 in the *rule\_array* parameter.

**text\_id\_in**

Direction: Input

Type: Integer

For CSNBMGN1/CSNEMGN1 only, the ALET of the text for which the MAC is generated.

### Usage Notes — General

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

### Usage Notes — CCF Systems

To use a DATA key, the NOCV-enablement keys must be present in the CKDS. Using a DATA key instead of a MAC generate key in this service substantially increases the path length for generating the MAC.

To calculate a MAC in one call, specify the ONLY keyword for segmenting control for the *rule\_array* parameter. For two or more calls, specify the FIRST keyword for the first input block, the MIDDLE keyword for intermediate blocks (if any), and the LAST keyword for the last block.

For a given text string, the resulting MAC is the same whether the text is segmented or not.

### Access Control Point

The **MAC Generate** access control point controls the function of this service.

### Required Hardware

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 164. MAC generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	<p>ICSF routes the request to a PCI Cryptographic Coprocessor if the control vector in the supplied key identifier cannot be processed on the Cryptographic Coprocessor Feature. If no PCI Cryptographic Coprocessor is online in this case, the request fails. The request must meet the following restrictions:</p> <ul style="list-style-type: none"> <li>• The MAC Process Rule is X9.19OPT or EMVMACD.</li> <li>• The MAC key is a valid double-length MAC generate key.</li> <li>• The <i>text_length</i> must be less than or equal to 4K bytes for the FIRST and MIDDLE keywords, and the text length must be a multiple of 8 bytes.</li> <li>• The <i>text_length</i> on the final call (ONLY or LAST) can not be greater than 4K including padding.</li> </ul> <p>TDES-MAC not supported.</p>
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	TDES-MAC not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor  Crypto Express3 Coprocessor	
z196  z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## Related Information

For more information about MAC processing rules and segmenting control, refer to IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference.

The MAC verification callable service is described in “MAC Verify (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)” on page 440.

## MAC Verify (CSNBMVR or CSNBMVR1 and CSNEMVR or CSNEMVR1)

Use the MAC verify callable service to verify a 4-, 6-, or 8-byte message authentication code (MAC) for an application-supplied text string. You can specify that the callable service uses either the ANSI X9.9-1 procedure or the ANSI X9.19 optional double key MAC procedure to compute the MAC. For the ANSI X9.9-1 procedure you identify either a MAC verify key, a MAC generation key, or a DATA key, and the message text. For the ANSI X9.19 optional double key MAC procedure, you identify either a double-length MAC verify key or a double-length MAC generation key and the message text. The cryptographic feature compares the generated MAC with the one sent with the message. A return code indicates whether the MACs are the same. If the MACs are the same, the receiver knows the message was not altered. The generated MAC never appears in storage is not revealed outside the cryptographic feature.

The MAC verify callable service also supports the padding rules specified in the EMV Specification and ISO 16609. For the EMV MAC procedure, you identify a single- or double-length MAC key and the message text. For the ISO 16609 procedure you identify a double-length MAC or DATA key and the message text.

### Choosing Between CSNBMVR and CSNBMVR1

CSNBMVR and CSNBMVR1 provide identical functions. When choosing which service to use, consider the following:

- **CSNBMVR** requires the application-supplied text to reside in the caller's primary address space. Also, a program using CSNBMVR adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

The callable service name for AMODE(64) invocation is CSNEMVR.

- **CSNBMVR1** allows the application-supplied text to reside either in the caller's primary address space or in a data space. This can allow you to verify more data with one call. However, a program using CSNBMVR1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified before it can run with other cryptographic products that follow this programming interface.

The callable service name for AMODE(64) invocation is CSNEMVR1.

For CSNBMVR1, *text\_id\_in* is an access list entry token (ALET) parameter of the data space containing the application-supplied text.

### Format

```
CALL CSNBMVR(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector,
    mac )
```

```
CALL CSNBMVR1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector,
    mac,
    text_id_in )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **key\_identifier**

Direction: Input/Output

Type: String

The 64-byte key label or internal key token that identifies a single or double-length MAC verify key, a single or double-length MAC verify key, a single or double length MAC generation key, a DATAM or DATAMV key, or a single-length DATA key. The type of key depends on the MAC process rule in the *rule\_array* parameter.

### **text\_length**

Direction: Input

Type: Integer

The length of the text you supply in the *text* parameter. The maximum length of text is 214783647 bytes. If the *text\_length* parameter is not a multiple of 8 bytes and if the ONLY or LAST keyword of the *rule\_array* parameter is called, the text is padded in accordance with the processing rule specified.

**Note:** The MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647).

#### **text**

Direction: Input

Type: String

The application-supplied text for which the MAC is generated.

#### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords specified in the *rule\_array* parameter. The value can be 0, 1, 2, or 3.

#### **rule\_array**

Direction: Input

Type: String

Zero to three keywords that provide control information to the callable service. The keywords are shown in Table 165. The keywords must be in 24 bytes of contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. For example,

```
'X9.9-1 MIDDLE MACLEN4 '
```

The order of the *rule\_array* keywords is not fixed.

You can specify one of the MAC processing rules and then choose one of the segmenting control keywords and one of the MAC length keywords.

*Table 165. Keywords for MAC verify Control Information*

Keyword	Meaning
<b>MAC Process Rules (optional)</b>	
EMVMAC	EMV padding rule with a single-length MAC key. The <i>key_identifier</i> parameter must identify a single-length MAC, MACVER, or DATA key. The text is always padded with 1 to 8 bytes so that the resulting text length is a multiple of 8 bytes. The first pad character is X'80'. The remaining 0 to 7 pad characters are X'00'.
EMVMACD	EMV padding rule with a double-length MAC key. The <i>key_identifier</i> parameter must identify a double-length MAC or MACVER key. The padding rules are the same as for EMVMAC.
X9.19OPT	ANSI X9.9-1 and X9.19 basic procedure. The <i>key_identifier</i> parameter must identify a single-length MAC, MACVER, or DATA key. X9.9-1 causes the MAC to be computed from all of the data. The text is padded only if the text length is not a multiple of 8 bytes. If padding is required, the pad character X'00' is used. This is the default value.

Table 165. Keywords for MAC verify Control Information (continued)

Keyword	Meaning
X9.9-1	ANSI X9.9-1 and X9.19 basic procedure. The <i>key_identifier</i> parameter must identify a single-length MAC, or single-length DATA key. X9.9-1 causes the MAC to be computed from all of the data. The text is padded only if the text length is not a multiple of 8 bytes. If padding is required, the pad character X'00' is used. This is the default value.
TDES-MAC	ISO 16609 procedure. The <i>key_identifier</i> must identify a double-length MAC or a double-length DATA key. The text is padded only if the text length is not a multiple of 8 bytes.
<b>Segmenting Control (optional)</b>	
FIRST	First call; this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.
ONLY	Only call; the application program does not employ segmenting. This is the default value.
<b>MAC Length and Presentation (optional)</b>	
HEX-8	Verifies a 4-byte MAC value that is represented as 8 hexadecimal characters.
HEX-9	Verifies a 4-byte MAC value that is represented as 2 groups of 4 hexadecimal characters with a space character between the groups.
MACLEN4	Verifies a 4-byte MAC value. This is the default value.
MACLEN6	Verifies a 6-byte MAC value.
MACLEN8	Verifies an 8-byte MAC value.

**chaining\_vector**

Direction: Input/Output

Type: String

An 18-byte string that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter to binary zeros.

**mac**

Direction: Output

Type: String

The 8- or 9-byte field that contains the MAC value you want to verify. The value in the field must be left-justified and padded with zeros. If you specified the X'09' keyword in the *rule\_array* parameter, the input MAC is 9 bytes.

**text\_id\_in**

Direction: Input

Type: Integer

## MAC Verify

For CSNBMVR1/CSNEMVR1 only, the ALET of the text for which the MAC is to be verified.

### Usage Notes — General

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

To verify a MAC in one call, specify the **ONLY** keyword on the segmenting rule keyword for the *rule\_array* parameter. For two or more calls, specify the **FIRST** keyword for the first input block, **MIDDLE** for intermediate blocks (if any), and **LAST** for the last block.

For a given text string, the MAC resulting from the verification process is the same regardless of how the text is segmented, or how it was segmented when the original MAC was generated.

### Usage Notes — CCF Systems

To use a MAC generation key or a DATA key, the NOCV enablement keys must be present in the CKDS. Using either a MAC generation key or a DATA key instead of a MAC verify key in this service substantially increases the path length for verifying the MAC.

### Access Control Point

The **MAC Verify** access control point controls the function of this service.

### Required Hardware

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 166. MAC verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	<p>ICSF routes the request to a PCI Cryptographic Coprocessor if the control vector in the supplied key identifier cannot be processed on the Cryptographic Coprocessor Feature. The request must meet the following restrictions:</p> <ul style="list-style-type: none"><li>• The MAC Process Rule is X9.19OPT or EMVMACD.</li><li>• The MAC key is a valid double-length MAC generate key.</li><li>• The <i>text_length</i> on the final call (<b>ONLY</b> or <b>LAST</b>) can not be greater than 4K including padding.</li><li>• The <i>text_length</i> must be less than or equal to 4K bytes for the <b>FIRST</b> and <b>MIDDLE</b> keywords, and the text length must be a multiple of 8 bytes.</li></ul> <p>TDES-MAC not supported.</p>

Table 166. MAC verify required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	TDES-MAC not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor  Crypto Express3 Coprocessor	
z196  z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## Related Information

For more information about MAC processing rules and segmenting control, refer to IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference.

The MAC generation callable service is described in “MAC Generate (CSNBMGN or CSNBMGN1 and CSNEMGN or CSNEMGN1)” on page 434.

---

## MDC Generate (CSNBMDG or CSNBMDG1 and CSNEMDG or CSNEMDG1)

A modification detection code (MDC) can be used to provide a form of support for data integrity.

Use the MDC generate callable service to generate a 128-bit modification detection code (MDC) for an application-supplied text string.

The returned MDC value should be securely stored and/or sent to another user. To validate the integrity of the text string at a later time, the MDC generate callable service is again used to generate a 128-bit MDC. The new MDC value is compared with the original MDC value. If the values are equal, the text is accepted as unchanged.

## Choosing Between CSNBMDG and CSNBMDG1

CSNBMDG and CSNBMDG1 provide identical functions. When choosing which service to use, consider the following:



- **CSNBMDG** requires the application-supplied text to reside in the caller's primary address space. Also, a program using CSNBMDG adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

The callable service name for AMODE(64) invocation is CSNEMDG.

- **CSNBMDG1** allows the application-supplied text to reside either in the caller's primary address space or in a data space. This can allow you to process more data with one call. However, a program using CSNBMDG1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface and may need to be modified before it can run with other cryptographic products that follow this programming interface.

The callable service name for AMODE(64) invocation is CSNEMDG1.

For CSNBMDG1, *text\_id\_in* parameter specifies the access list entry token (ALET) for the data space containing the application-supplied text.

## Format

```
CALL CSNBMDG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector,
    mdc )
```

```
CALL CSNBMDG1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector,
    mdc,
    text_id_in )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes," on page 781 lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that

indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes," on page 781 lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **text\_length**

Direction: Input

Type: Integer

The length of the text you supply in the *text* parameter. The maximum length of text is 214783647 bytes.

**Note:** The MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced (214783647).

Additional restrictions on length of the text depend on whether padding of the text is requested, and on the segmenting control used.

- When padding is requested (by specifying a process rule of PADMDC-2 or PADMDC-4 in the *rule\_array* parameter), a text length of 0 is valid for any segment control specified in the *rule\_array* parameter (FIRST, MIDDLE, LAST, or ONLY). When LAST or ONLY is specified, the supplied text will be padded with X'FF's and a padding count in the last byte to bring the total text length to the next multiple of 8 that is greater than or equal to 16,
- When no padding is requested (by specifying a process rule of MDC-2 or MDC-4), the total length of the text provided (over a single or segmented calls) must be at least 16 bytes, and a multiple of 8.

For segmented calls with no padding, text length of 0 is valid on any of the calls provided the total length over the segmented calls is at least 16 and a multiple of 8.

For a single call (that is, segment control is ONLY) with no padding, the length the text provided must be at least 16, and a multiple of 8.

### **text**

Direction: Input

Type: String

The application-supplied text for which the MDC is generated.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords specified in the *rule\_array* parameter. This value must be 2.

### **rule\_array**

## MDC Generate

Direction: Input

Type: Character string

The two keywords that provide control information to the callable service are shown in Table 167. The two keywords must be in 16 bytes of contiguous storage with each of the two keywords left-justified in its own 8-byte location and padded on the right with blanks. For example,

```
'MDC-2    FIRST    '
```

Choose one of the MDC process rule control keywords and one of the segmenting control keywords from the following table.

*Table 167. Keywords for MDC Generate Control Information*

Keyword	Meaning
<b>MDC Process Rules (required)</b>	
MDC-2	MDC-2 specifies two encipherments per 8 bytes of input text and no padding of the input text.
MDC-4	MDC-4 specifies four encipherments per 8 bytes of input text and no padding of the input text.
PADMDC-2	PADMDC-2 specifies two encipherments per 8 bytes of input text and padding of the input text.  When the segment rule specifies ONLY or LAST, the input text is padded with X'FF's and a padding count in the last byte to bring the total text length to the next even multiple of 8 that is greater than, or equal to, 16.
PADMDC-4	PADMDC-4 specifies four encipherments per 8 bytes of input text and padding of the input text.  When the segment rule specifies ONLY or LAST, the input text is padded with X'FF's and a padding count in the last byte to bring the total text length to the next even multiple of 8 that is greater than, or equal to, 16.
<b>Segmenting Control (required)</b>	
FIRST	First call; this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.
ONLY	Only call; segmenting is not employed by the application program.

### chaining\_vector

Direction: Input/Output

Type: String

An 18-byte string that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter as binary zeros.

### mdc

Direction: Input/Output

Type: String

A 16-byte field in which the callable service returns the MDC value when the

segmenting rule is ONLY or LAST. When the segmenting rule is FIRST or MIDDLE, the value returned in this field is an intermediate MDC value that will be used as input for a subsequent call and must not be changed by the application program.

**text\_id\_in**

Direction: Input

Type: Integer

For CSNBMDG1/CSNEMDG1 only, the ALET for the data space containing the text for which the MDC is to be generated.

## Usage Notes

To calculate an MDC in one call, specify the ONLY keyword for segmenting control in the *rule\_array* parameter. For more than one call, specify the FIRST keyword for the first input block, the MIDDLE keyword for any intermediate blocks, and the LAST keyword for the last block. For a given text string, the resulting MDC is the same whether the text is segmented or not.

The two versions of MDC calculation (with two or four encipherments per 8 bytes of input text) allow the caller to trade a performance improvement for a decrease in security. Since 2 encipherments create results different from the results of 4 encipherments, ensure that you use the same number of encipherments to verify the MDC value.

## Required Hardware

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 168. MDC generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990 IBM @server zSeries 890	CP Assist for Cryptographic Functions	
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions	
z196 z114	CP Assist for Cryptographic Functions	
IBM zEnterprise EC12	CP Assist for Cryptographic Functions	

### One-Way Hash Generate (CSNBOWH or CSNBOWH1 and CSNEOWH or CSNEOWH1)

Use the one-way hash generate callable service to generate a one-way hash on specified text. This service supports the following methods:

- MD5 - software only
- SHA-1
- RIPEMD-160 - software only
- SHA-224
- SHA-256
- SHA-384
- SHA-512

The callable service names for AMODE(64) invocation are CSNEOWH and CSNEOWH1.

### Format

```
CALL CSNBOWH(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    text_length,  
    text,  
    chaining_vector_length,  
    chaining_vector,  
    hash_length,  
    hash)
```

```
CALL CSNBOWH1(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    text_length,  
    text,  
    chaining_vector_length,  
    chaining_vector,  
    hash_length,  
    hash,  
    text_id_in)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

**exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The value must be 1 or 2.

**rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service are listed in Table 169. The optional chaining flag keyword indicates whether calls to this service are chained together logically to overcome buffer size limitations. Each keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

*Table 169. Keywords for One-Way Hash Generate Rule Array Control Information*

Keyword	Meaning
<i>Hash Method (required)</i>	
MD5	Hash algorithm is MD5 algorithm. Use this hash method for PKCS-1.0 and PKCS-1.1. Length of hash generated is 16 bytes.
MD5-LG	Hash algorithm is similar to the MD5 algorithm. Use this hash method for PKCS-1.0 and PKCS-1.1. Length of hash generated is 16 bytes. Legacy hash values from release HCR7751 and lower prior to APAR OA33657 will be generated for verification purposes with previously archived hash values.
RPMD-LG	Hash algorithm is similar to the RIPEMD-160. Length of hash generated is 20 bytes. Legacy hash values from release HCR7751 and lower prior to APAR OA33657 will be generated for verification purposes with previously archived hash values.
RPMD-160	Hash algorithm is RIPEMD-160. Length of hash generated is 20 bytes.

## One-Way Hash Generate

Table 169. Keywords for One-Way Hash Generate Rule Array Control Information (continued)

Keyword	Meaning
SHA-1	Hash algorithm is SHA-1 algorithm. Use this hash method for DSS. Length of hash generated is 20 bytes.
SHA-224	Hash algorithm is SHA-256 algorithm. Length of hash generated is 28 bytes.
SHA-256	Hash algorithm is SHA-256 algorithm. Length of hash generated is 32 bytes.
SHA-384	Hash algorithm is SHA-384 algorithm. Length of hash generated is 48 bytes.
SHA-512	Hash algorithm is SHA-512 algorithm. Length of hash generated is 64 bytes.
<b>Chaining Flag (optional)</b>	
FIRST	Specifies this is the first call in a series of chained calls. Intermediate results are stored in the <i>hash</i> field.
LAST	Specifies this is the last call in a series of chained calls.
MIDDLE	Specifies this is a middle call in a series of chained calls. Intermediate results are stored in the <i>hash</i> field.
ONLY	Specifies this is the only call and the call is not chained. This is the default.

### text\_length

Direction: Input

Type: Integer

The length of the *text* parameter in bytes.

**Note:** If you specify the FIRST or MIDDLE keyword, then the text length must be a multiple of the blocksize of the hash method. For MD5, RPMD-160, SHA-1, SHA-224 and SHA-256, this is a multiple of 64 bytes. For SHA-384 and SHA-512, this is a multiple of 128 bytes.

For ONLY and LAST, this service performs the required padding according to the algorithm specified.

### text

Direction: Input

Type: String

The application-supplied text on which this service performs the hash.

### chaining\_vector\_length

Direction: Input

Type: Integer

The byte length of the *chaining\_vector* parameter. This must be 128 bytes.

### chaining\_vector

Direction: Input/Output

Type: String

This field is a 128-byte work area. Your application must not change the data in this string. The chaining vector permits chaining data from one call to another.

### hash\_length

Direction: Input

Type: Integer

The length of the supplied *hash* field in bytes.

**Note:** For SHA-1 and RPMD-160 this must be at least 20 bytes; for MD5 this must be at least 16 bytes. For SHA-224 and SHA-256, the length must be at least 32 bytes long. Even though the length of the SHA-224 hash is less than SHA-256, the extra bytes are used as a work area during the generation of the hash value. The SHA-224 value is left-justified and padded with zeroes.

For SHA-384 and SHA-512, the length must be at least 64 bytes long. Even though the length of the SHA-384 hash is less than SHA-512, the extra bytes are used as a work area during the generation of the hash value. The SHA-384 value is left-justified and padded with zeroes.

### hash

Direction: Input/Output

Type: String

This field contains the hash, left-justified. The processing of the rest of the field depends on the implementation. If you specify the FIRST or MIDDLE keyword, this field contains the intermediate hash value. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message.

### text\_id\_in

Direction: Input

Type: Integer

For CSNBOWH1 only, the ALET for the data space containing the text for which to generate the hash.

## Usage Notes

Although MD5, SHA-1 and SHA-256 allow it, bit length text is not supported for any hashing method.

## Required Hardware

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.



## One-Way Hash Generate

Table 170. One-way hash generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	SHA-1 requires CCF SHA-224 keyword not supported SHA-256 keyword not supported SHA-384 keyword not supported SHA-512 keyword not supported
IBM @server zSeries 990 IBM @server zSeries 890	CP Assist for Cryptographic Functions	SHA-1 requires CPACF SHA-224 keyword not supported SHA-256 keyword not supported SHA-384 keyword not supported SHA-512 keyword not supported
IBM System z9 EC IBM System z9 BC	CP Assist for Cryptographic Functions	SHA-384 keyword not supported SHA-512 keyword not supported
IBM System z10 EC IBM System z10 BC	CP Assist for Cryptographic Functions	
z196 z114	CP Assist for Cryptographic Functions	
IBM zEnterprise EC12	CP Assist for Cryptographic Functions	

## Symmetric MAC Generate (CSNBSMG or CSNBSMG1 and CSNESMG or CSNESMG1)

Use the symmetric MAC generate callable service to generate a 96- or 128-bit message authentication code (MAC) for an application-supplied text string using an AES key.

The callable service names for AMODE(64) invocation are CSNESMG and CSNESMG1.

### Choosing Between CSNBSMG and CSNBSMG1 or CSNESMG and CSNESMG1

CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1 provide identical functions. When choosing which service to use, consider this:

- CSNBSMG and CSNESMG require the text to reside in the caller's primary address space. Also, a program using CSNBSMG adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- CSNBSMG1 and CSNESMG1 allow the text to reside either in the caller's primary address space or in a data space. This can allow you to decipher more

data with one call. However, a program using CSNBSMG1 and CSNESMG1 do not adhere to the IBM CCA: Cryptographic API and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSMG1 and CSNESMG1, *text\_id\_in* is an access list entry token (ALET) parameter of the data spaces containing the text.

## Format

```
CALL CSNBSMG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier_length,
    key_identifier,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector_length,
    chaining_vector,
    reserved_data_length,
    reserved_data
    mac_length
    mac )
```

```
CALL CSNBSMG1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier_length,
    key_identifier,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector_length,
    chaining_vector,
    reserved_data_length,
    reserved_data
    mac_length
    mac
    text_id_in)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

## Symmetric MAC Generate

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **key\_identifier\_length**

Direction: Input

Type: String

The length of the *key\_identifier* parameter. For the KEY-CLR keyword, the length is in bytes and includes only the value of the key length. The key length value can be 16, 24, or 32. For the KEYIDENT keyword, the length must be 64.

### **key\_identifier**

Direction: Input

Type: String

For the KEY-CLR keyword, this specifies the clear AES key. The parameter must be left justified. For the KEYIDENT keyword, this specifies an internal clear AES token or the label name of a clear AES key in the CKDS. Normal CKDS label name syntax is required.

### **text\_length**

Direction: Input

Type: Integer

The length of the text you supply in the *text* parameter. The maximum length of text is 2147483647 bytes. If the *text\_length* is not a multiple of 8 bytes and if the ONLY or LAST keyword of the *rule\_array* parameter is called, the text is padded in accordance with the processing rule specified.

### **text**

Direction: Input

Type: String

The application-supplied text for which the MAC is generated.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords specified in the *rule\_array* parameter. The value can be 1, 2, 3 or 4.

### **rule\_array**

Direction: Input

Type: Character string

This keyword provides control information to the callable service. The keywords must be eight bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

You can specify one of the MAC processing rules and then choose one of the segmenting control keywords and one of the MAC length keywords.

Table 171. Keywords for symmetric MAC generate control information

Keyword	Meaning
<i>Algorithm (required)</i>	
AES	Specifies that the Advanced Encryption Standard (AES) algorithm is to be used.
<i>MAC processing rule (optional)</i>	
CBC-MAC	CBC MAC with padding for any key length. This is the default value.
XCBC-MAC	AES-XCBC-MAC-96 and AES-XCBC-PRF-128 MAC generation with padding for 128-bit keys.
<i>Key rule (optional)</i>	
KEY-CLR	This specifies that the key parameter contains a clear key value. This is the default value.
KEYIDENT	This specifies that the key_identifier field will be an internal clear token or the label name of a clear key in the CKDS. Normal CKDS label name syntax is required.
<i>Segmenting Control (optional)</i>	
FIRST	First call, this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.
ONLY	Only call; segmenting is not employed by the application program. This is the default value.

### chaining\_vector\_length

Direction: Input/Output

Type: Integer

The length of the *chaining\_vector* parameter. On output, the actual length of the chaining vector will be stored in the parameter.

### chaining\_vector

Direction: Input/Output

Type: String

This field is used as a system work area for the chaining vector. Your application program must not change the data in this string. The chaining vector holds the output chaining vector from the caller.

The mapping of the *chaining\_vector* depends on the algorithm specified. For AES, the *chaining\_vector* field must be at least 36 bytes in length.

### reserved\_data\_length

## Symmetric MAC Generate

Direction: Input

Type: Integer

Reserved for future use. Value must be zero.

### **reserved\_data**

Direction: Ignored

Type: String

Reserved for future use.

### **mac\_length**

Direction: Input

Type: Integer

The length in bytes of the MAC to be returned in the mac field. The allowable values are 12 and 16 bytes.

### **mac**

Direction: Output

Type: String

The 12-byte or 16-byte field in which the callable service returns the MAC value if the segmenting rule is ONLY or LAST.

### **text\_id\_in**

Direction: Input

Type: Integer

For CSNBSMG1 and CSNESMG1 only, the ALET of the text for which the MAC is generated.

## Required Hardware

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 172. Symmetric MAC generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990	CPACF	
IBM @server zSeries 890		
IBM System z9 EC	CPACF	
IBM System z9 BC		
IBM System z10 EC	CPACF	
IBM System z10 BC		
z196	CPACF	
z114		
IBM zEnterprise EC12	CPACF	

## Symmetric MAC Verify (CSNBSMV or CSNBSMV1 and CSNESMV or CSNESMV1)

Use the symmetric MAC verify callable service to verify a 96- or 128-bit message authentication code (MAC) for an application-supplied text string using an AES key.

The callable service names for AMODE(64) invocation are CSNESMV and CSNESMV1.

### Choosing Between CSNBSMV and CSNBSMV1 or CSNESMV and CSNESMV1

CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1 provide identical functions. When choosing which service to use, consider this:

- CSNBSMV and CSNESMV require the text to reside in the caller's primary address space. Also, a program using CSNBSMV adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- CSNBSMV1 and CSNESMV1 allow the text to reside either in the caller's primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using CSNBSMV1 and CSNESMV1 do not adhere to the IBM CCA: Cryptographic API and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSMV1 and CSNESMV1, *text\_id\_in* is an access list entry token (ALET) parameter of the data spaces containing the text.

### Format

```
CALL CSNBSMV(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier_length,
    key_identifier,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector_length,
    chaining_vector,
    reserved_data_length,
    reserved_data,
    mac_length,
    mac )
```

## Symmetric MAC Verify

```
CALL CSNBSMV1(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    key_identifier_length,  
    key_identifier,  
    text_length,  
    text,  
    rule_array_count,  
    rule_array,  
    chaining_vector_length,  
    chaining_vector,  
    reserved_data_length,  
    reserved_data  
    mac_length  
    mac  
    text_id_in )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

#### **key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *key\_identifier* parameter. For the KEY-CLR keyword, the length is in bytes and includes only the value of the key length. The key length value can be 16, 24, or 32. For the KEYIDENT keyword, the length must be 64.

#### **key\_identifier**

Direction: Input

Type: String

For the KEY-CLR keyword, this specifies the clear AES key. The parameter must be left justified. For the KEYIDENT keyword, this specifies an internal clear AES token or the label name of a clear AES key in the CKDS. Normal CKDS label name syntax is required.

**text\_length**

Direction: Input

Type: Integer

The length of the text you supply in the *text* parameter. The maximum length of text is 2147483647 bytes. If the *text\_length* parameter is not a multiple of 8 bytes and if the ONLY or LAST keyword of the *rule\_array* parameter is called, the text is padded in accordance with the processing rule specified.

**text**

Direction: Input

Type: String

The application-supplied text for which the MAC is verified.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords specified in the *rule\_array* parameter. The value can be 1, 2, 3 or 4.

**rule\_array**

Direction: Input

Type: String

This keyword provides control information to the callable service. The keywords must be eight bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The order of the *rule\_array* keywords is not fixed.

You can specify one of the MAC processing rules and then choose one of the segmenting control keywords and one of the MAC length keywords.

Table 173. Keywords for symmetric MAC verify control information

Keyword	Meaning
<i>Algorithm (required)</i>	
AES	Specifies that the Advanced Encryption Standard (AES) algorithm is to be used.
<i>MAC processing rule (optional)</i>	
CBC-MAC	CBC MAC with padding for any key length. This is the default value.
XCBC-MAC	AES-XCBC-MAC-96 and AES-XCBC-PRF-128 MAC generation with padding for 128-bit keys.
<i>Key rule (optional)</i>	
KEY-CLR	This specifies that the key parameter contains a clear key value. This is the default value.



## Symmetric MAC Verify

Table 173. Keywords for symmetric MAC verify control information (continued)

Keyword	Meaning
KEYIDENT	This specifies that the <code>key_identifier</code> field will be an internal clear token or the label name of a clear key in the CKDS. Normal CKDS label name syntax is required.
<i>Segmenting Control (optional)</i>	
FIRST	First call, this is the first segment of data from the application program.
LAST	Last call; this is the last data segment.
MIDDLE	Middle call; this is an intermediate data segment.
ONLY	Only call; segmenting is not employed by the application program. This is the default value.

### **chaining\_vector\_length**

Direction: Input/Output

Type: String

The length of the *chaining\_vector* parameter. On output, the actual length of the chaining vector will be stored in the parameter.

### **chaining\_vector**

Direction: Input/Output

Type: String

This field is used as a system work area for the chaining vector. Your application program must not change the data in this string. The chaining vector holds the output chaining vector from the caller.

The mapping of the *chaining\_vector* depends on the algorithm specified. For AES, the *chaining\_vector* field must be at least 36 bytes in length.

### **reserved\_data\_length**

Direction: Input

Type: Integer

Reserved for future use. Value must be zero.

### **reserved\_data**

Direction: Ignored

Type: String

Reserved for future use.

### **mac\_length**

Direction: Input

Type: Integer

The length in bytes of the MAC to be verified the *mac* field. The allowable values are 12 and 16 bytes.

### **mac**

Direction: Input

Type: String

The 12-byte or 16-byte field that contains the MAC value you want to verify. The value must be left-justified and padded with zeros.

**text\_id\_in**

Direction: Input

Type: Integer

For CSNBSMV1 and CSNESMV1 only, the ALET of the text for which the MAC is to be verified.

## Required Hardware

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 174. Symmetric MAC verify required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990	CPACF	
IBM @server zSeries 890		
IBM System z9 EC IBM System z9 BC	CPACF	
IBM System z10 EC IBM System z10 BC	CPACF	
z196 z114	CPACF	
IBM zEnterprise EC12		

## Symmetric MAC Verify

---

## Chapter 8. Financial Services

The process of validating personal identities in a financial transaction system is called *personal authentication*. The personal identification number (PIN) is the basis for verifying the identity of a customer across financial industry networks. ICSF provides callable services to translate, verify, and generate PINs. You can use the callable services to prevent unauthorized disclosures when organizations handle PINs.

These callable services are described in these topics:

- “Clear PIN Encrypt (CSNBCPE and CSNECPE)” on page 476
- “Clear PIN Generate (CSNBPGN and CSNEPGN)” on page 480
- “Clear PIN Generate Alternate (CSNBCPA and CSNECPA)” on page 485
- “CVV Key Combine (CSNBCKC and CSNECKC)” on page 491
- “Encrypted PIN Generate (CSNBEPG and CSNEEPG)” on page 497
- “Encrypted PIN Translate (CSNBPTR and CSNEPTR)” on page 502
- “Encrypted PIN Verify (CSNBPVR and CSNEPVR)” on page 509
- “PIN Change/Unblock (CSNBPCU and CSNEPCU)” on page 516
- “Secure Messaging for Keys (CSNBSKY and CSNESKY)” on page 523
- “Secure Messaging for PINs (CSNBSPN and CSNESPN)” on page 526
- “SET Block Compose (CSNDSBC and CSNFSBC)” on page 531
- “SET Block Decompose (CSNDSBD and CSNFSBD)” on page 536
- “Transaction Validation (CSNBTRV and CSNETRV)” on page 542
- “VISA CVV Service Generate (CSNBCSG and CSNECSG)” on page 546
- “VISA CVV Service Verify (CSNBCSV and CSNECSV)” on page 551

---

### How Personal Identification Numbers (PINs) are Used

Many people are familiar with PINs, which allow them to use an automated teller machine (ATM). From the system point of view, PINs are used primarily in financial networks to authenticate users — typically, a user is assigned a PIN, and enters the PIN at automated teller machines (ATMs) to gain access to his or her accounts. It is extremely important that the PIN be kept private, so that no one other than the account owner can use it. ICSF allows your applications to generate PINs, to verify supplied PINs, and to translate PINs from one format to another.

---

### How VISA Card Verification Values Are Used

The Visa International Service Association (VISA) and MasterCard International, Incorporated have specified a cryptographic method to calculate a value that relates to the personal account number (PAN), the card expiration date, and the service code. The VISA card-verification value (CVV) and the MasterCard card-verification code (CVC) can be encoded on either track 1 or track 2 of a magnetic striped card and are used to detect forged cards. Because most online transactions use track-2, the ICSF callable services generate and verify the CVV<sup>4</sup> by the track-2 method.

The VISA CVV service generate callable service calculates a 1- to 5-byte value through the DES-encryption of the PAN, the card expiration date, and the service code using two data-encrypting keys or two MAC keys. The VISA CVV service

---

4. The VISA CVV and the MasterCard CVC refer to the same value. CVV is used here to mean both CVV and CVC.

verify callable service calculates the CVV by the same method, compares it to the CVV supplied by the application (which reads the credit card's magnetic stripe) in the *CVV\_value*, and issues a return code that indicates whether the card is authentic.

---

## Translating Data and PINs in Networks

More and more data is being transmitted across networks where, for various reasons, the keys used on one network cannot be used on another network. Encrypted data and PINs that are transmitted across these boundaries must be “translated” securely from encryption under one key to encryption under another key. For example, a traveler visiting a foreign city might wish to use an ATM to access an account at home. The PIN entered at the ATM might need to be encrypted at the ATM and sent over one or more financial networks to the traveler's home bank. At the home bank, the PIN must be verified prior to access being allowed. On intermediate systems (between networks), applications can use the Encrypted PIN translate callable service to re-encrypt a PIN block from one key to another. Running on ICSF, such applications can ensure that PINs never appear in the clear and that the PIN-encrypting keys are isolated on their own networks.

---

## Working with Europay–MasterCard–Visa smart cards

There are several services you can use in secure communications with EMV smart cards. The processing capabilities are consistent with the specifications provided in these documents:

- *EMV 2000 Integrated Circuit Card Specification for Payment Systems Version 4.0 (EMV4.0) Book 2*
- *Design Visa Integrated Circuit Card Specification Manual*
- *Integrated Circuit Card Specification (VIS) 1.4.0 Corrections*

EMV smart cards include the following processing capabilities:

- The diversified key generate (CSNBDBG and CSNEDKG) callable service with rule-array options **TDES-XOR**, **TDESEMV2**, and **TDESEMV4** enables you to derive a key used to cipher and authenticate messages, and more particularly message parts, for exchange with an EMV smart card. You use the derived key with services such as encipher, decipher, MAC generate, MAC verify, secure messaging for keys, and secure messaging for PINs. These message parts can be combined with message parts created using the secure messaging for keys and secure messaging for PINs services.
- The secure messaging for keys (CSNBSKY and CSNESKY) service enables you to securely incorporate a key into a message part (generally the value portion of a TLV component of a secure message for a card). Similarly, the secure messaging for PINs (CSNBSPN and CSNESPEN) service enables secure incorporation of a PIN block into a message part.
- The PIN change/unblock (CSNBPCU and CSNEPCU) service enables you to encrypt a new PIN to send to a new EMV card, or to update the PIN value on an initialized EMV card. This verb generates both the required session key (from the master encryption key) and the required authentication code (from the master authentication key).
- The **ZERO-PAD** option of the PKA encrypt (CSNDPKE) service enables you to validate a digital signature created according to ISO 9796-2 standard by encrypting information you format, including a hash value of the message to be validated. You compare the resulting enciphered data to the digital signature accompanying the message to be validated.

- The MAC generate and MAC verify services post-pad a X'80'...X'00' string to a message as required for authenticating messages exchanged with EMV smart cards.

---

## PIN Callable Services

You use the PIN callable services to generate, verify, and translate PINs. This topic discusses the PIN callable services, as well as the various PIN algorithms and PIN block formats supported by ICSF. It also explains the use of PIN-encrypting keys.

### Generating a PIN

To generate personal identification numbers, call the Clear PIN Generate or Encrypted PIN Generate callable service. Using a PIN generation algorithm, data used in the algorithm, and the PIN generation key, the Clear PIN generate callable service generates a clear PIN and a PIN verification value, or offset. The Clear PIN Generate callable service can only execute in special secure mode. For a description of this mode, see “Special Secure Mode” on page 10. Using a PIN generation algorithm, data used in the algorithm, the PIN generation key, and an outbound PIN encrypting key, the encrypted PIN generate callable service generates and formats a PIN and encrypts the PIN block.

### Encrypting a PIN

To format a PIN into a supported PIN block format and encrypt the PIN block, call the Clear PIN encrypt callable service.

### Generating a PIN Validation Value from an Encrypted PIN Block

To generate a clear VISA PIN validation value (PVV) from an encrypted PIN block, call the *clear PIN generate alternate* callable service. The PIN block can be encrypted under an input PIN-encrypting key (IPINENC) or an output PIN encrypting key (OPINENC). Using an IPINENC key requires that NOCV keys are enabled in the CKDS.

### Verifying a PIN

To verify a supplied PIN, call the *Encrypted PIN verify* callable service. You supply the enciphered PIN, the PIN-encrypting key that enciphers the PIN, and other data. You must also specify the PIN verification key and PIN verification algorithm. The callable service generates a verification PIN. The service compares the two personal identification numbers and if they are the same, it verifies the supplied PIN.

### Translating a PIN

To translate a PIN block format from one PIN-encrypting key to another or from one PIN block format to another, call the *Encrypted PIN translate* callable service. You must identify the input PIN-encrypting key that originally enciphered the PIN. You also need to specify the output PIN-encrypting key that you want the callable service to use to encipher the PIN. If you want to change the PIN block format, specify a different output PIN block format from the input PIN block format.

## Algorithms for Generating and Verifying a PIN

ICSF supports these algorithms for generating and verifying personal identification numbers:

- IBM 3624 institution-assigned PIN
- IBM 3624 customer-selected PIN (through a PIN offset)
- IBM German Bank Pool PIN (verify through an institution key)
- IBM German Bank Pool PIN (verify through a pool key and a PIN offset). This algorithm is supported when the service using the PIN is processed on the Cryptographic Coprocessor Feature. **Restriction:** This algorithm is not supported on a z990, z890, z9 EC or z9 BC.
- VISA PIN through a VISA PIN validation value
- Interbank PIN

The algorithms are discussed in detail in “PIN Formats and Algorithms” on page 929.

## Using PINs on Different Systems

ICSF allows you to translate different PIN block formats, which lets you use personal identification numbers on different systems. ICSF supports these formats:

- IBM 3624
- IBM 3621 (same as IBM 5906)
- IBM 4704 encrypting PINPAD format
- ISO 0 (same as ANSI 9.8, VISA 1, and ECI 1)
- ISO 1 (same as ECI 4)
- ISO 2
- ISO 3
- VISA 2
- VISA 3
- VISA 4
- ECI 2
- ECI 3

The formats are discussed in “PIN Formats and Algorithms” on page 929.

## PIN-Encrypting Keys

A unique master key variant enciphers each type of key. For further key separation, an installation can choose to have each PIN block format enciphered under a different PIN-encrypting key. The PIN-encrypting keys can have a 16-byte PIN block variant constant exclusive ORed on them prior to using to translate or verify PIN blocks. This is specified in the format control field in the Encrypted PIN translate and Encrypted PIN verify callable services.

You should only use PIN block variant constants when you are communicating with another host processor with the Integrated Cryptographic Service Facility.

### Derived Unique Key Per Transaction Algorithms

ICSF supports ANSI X9.24 derived unique key per transaction algorithms to generate PIN-encrypting keys from user data. ICSF supports both single- and double-length key generation. Keywords for single- and double-length key generation can not be mixed. A PCICCC, PCIXCC, or CCA Crypto Express coprocessor is required for this support. Double-length key generation is only supported on z990 with the May 2004 LIC or higher, z890, and later systems.

### Encrypted PIN Translate

The UKPTIPIN, IPKTOPIN and UKPTBOTH keywords will cause the service to generate single-length keys. DUKPT-IP, DKPT-OP and DUKPT-BH are the

respective keywords to generate double-length keys. The *input\_PIN\_profile* and *output\_PIN\_profile* must supply the current key serial number when these keywords are specified.

### Encrypted PIN Verify

The UKPTIPIN keyword will cause the service to generate single-length keys. DUKPT-IP is the keyword for double-length key generation. The *input\_PIN\_profile* must supply the current key serial number when these keywords are specified.

For more information about PIN-encrypting keys, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

---

## ANSI X9.8 PIN Restrictions

Access control points (ACP) in the ICSF role control PIN block processing restrictions from the X9.8 standard. These access control points are available on the z196 and z114 with the CEX3C, or the IBM zEnterprise EC12 with the CEX3C or CEX4C. These callable services are affected by these access control points. These access control points are disabled in the default role. A TKE Workstation is required to enable these ACPs.

- Clear PIN Generate Alternate (CSNBCPA and CSNECPA)
- Encrypted PIN Generate (CSNBEPG and CSNEEPG)
- Encrypted PIN Translate (CSNBPTR and CSNEPTR)
- Encrypted PIN Verify (CSNBPVR and CSNEPVR)
- Secure Messaging for PINs (CSNBSPN and CSNESP)

There are four access control points:

- ANSI X9.8 PIN - Enforce PIN block restrictions
- ANSI X9.8 PIN - Allow modification of PAN
- ANSI X9.8 PIN - Allow only ANSI PIN blocks
- ANSI X9.8 PIN - Use stored decimalization tables only

PIN decimalization tables can be stored in the CEX3C or CEX4C for use by callable services. Only tables that have been activated can be used. A TKE Workstation is required to manage the tables in the coprocessors.

**Note:** ICSF routes work to all active coprocessors based on work load. All coprocessors must have the same set of active decimalization tables for the **ANSI X9.8 PIN - Use stored decimalization tables only** access control point to be effective.

### ANSI X9.8 PIN - Enforce PIN block restrictions

When **ANSI X9.8 PIN - Enforce PIN block restrictions** access control point is enable, the following restrictions will be enforced.

- CSNBPTR and CSNBSPN will not accept IBM 3624 PIN format in the output profile parameter when the input profile parameter is not IBM 3624.
- CSNBPTR will not accept ISO-0 or ISO-3 formats in the input PIN profile unless ISO-0 or ISO-3 is in the output PIN profile.
- CSNBPTR and CSNBSPN will not accept ISO-1 or ISO-2 formats in the output profile parameter when the input profile parameter contains ISO-0, ISO-3, or VISA4



- When the input profile parameter of CSNBPTR or CSNBSPN contains either ISO-0 or ISO-3 formats, the decrypted PIN block will be examined to ensure that the PAN within the PIN block is the same as the PAN which was supplied as the input PAN parameter, and that this is the same as the PAN which was supplied as the output PAN parameter.
- The input PAN and output PAN parameters of CSNBPTR or CSNBSPN must be equivalent.
- When the rule array for CSNBCPA contains VISA-PVV, the input PIN profile must contain ISO-0 or ISO-3 formats.

## ANSI X9.8 PIN - Allow modification of PAN

In order to enable the **ANSI X9.8 PIN - Allow modification of PAN** access control point, the **ANSI X9.8 PIN - Enforce PIN block restrictions** must also be enabled. The **ANSI X9.8 PIN - Allow modification of PAN** access control point cannot be enabled by itself.

When the **ANSI X9.8 PIN - Allow modification of PAN** access control point is enabled, the input PAN and output PAN parameters will be tested in CSNBPTR or CSNBSPN. The input PAN will be compared to the portions of the PAN which are recoverable from the decrypted PIN block. If the PANs compare, then the account number will be changed in the output PIN block.

## ANSI X9.8 PIN - Allow only ANSI PIN blocks

In order to enable the **ANSI X9.8 PIN - Allow only ANSI PIN blocks** access control point, the **ANSI X9.8 PIN - Enforce PIN block restrictions** must also be enabled. The **ANSI X9.8 PIN - Allow only ANSI PIN blocks** access control point cannot be enabled by itself.

When this access control point is enable, CSNBPTR will allow reformatting of the PIN block as shown in the following table.

Table 175. ANSI X9.8 PIN - Allow only ANSI PIN blocks

Reformat To:	ISO Format 0	ISO Format 1	ISO Format 3
From:			
ISO Format 0	Reformat permitted Change of PAN not permitted	Not permitted	Reformat permitted Change of PAN not permitted
ISO Format 1	Reformat permitted	Reformat permitted	Reformat permitted
ISO Format 3	Reformat permitted Change of PAN not permitted	Not permitted	Reformat permitted Change of PAN not permitted

## ANSI X9.8 PIN – Use stored decimalization tables only

The **ANSI X9.8 PIN – Use stored decimalization tables only** access control point may be enabled by itself.

When this access control point is enabled, CSNBPGN, CSNBCPA, CSNBEPG, and CSNBPVR services must supply a decimalization table that matches the active decimalization tables stored in the coprocessors. The decimalization table in the *data\_array* parameter will be compared against the active decimalization tables in

the coprocessor and if the supplied table matches a stored table, the request will be processed. If the supplied table doesn't match any of the stored tables or there are no stored tables, the request will fail.

PIN decimalization tables can be stored in the CEX3C coprocessors for use by callable services. Only tables that have been activated can be used. A TKE Workstation is required to manage the tables in the coprocessors.

**Note:** ICSF routes work to all active coprocessors based on work load. All coprocessors must have the same set of decimalization tables for the decimalization table access control point to be effective.

---

## The PIN Profile

The PIN profile consists of:

- PIN block format (see "PIN Block Format")
- Format control (see "Format Control" on page 474)
- Pad digit (see "Pad Digit" on page 475)
- Current Key Serial Number (for UKPT and DUKPT – see "Current Key Serial Number" on page 476)

Table 176 shows the format of a PIN profile.

*Table 176. Format of a PIN Profile*

Bytes	Description
0–7	PIN block format
8–15	Format control
16–23	Pad digit
24–47	Current Key Serial Number (for UKPT and DUKPT )

## PIN Block Format

This keyword specifies the format of the PIN block. The 8-byte value must be left-justified and padded with blanks. Refer to Table 177 for a list of valid values.

*Table 177. Format Values of PIN Blocks*

Format Value	Description
ECI-2	Eurocheque International format 2
ECI-3	Eurocheque International format 3
ISO-0	ISO format 0, ANSI X9.8, VISA 1, and ECI 1
ISO-1	ISO format 1 and ECI 4
ISO-2	ISO format 2
ISO-3	ISO format 3
VISA-2	VISA format 2
VISA-3	VISA format 3
VISA-4	VISA format 4
3621	IBM 3621 and 5906
3624	IBM 3624
4704-EPP	IBM 4704 with encrypting PIN pad

## PIN Block Format and PIN Extraction Method Keywords

In the Clear PIN Generate Alternate, Encrypted PIN Translate and Encrypted PIN Verify callable services, you may specify a PIN extraction keyword for a given PIN block format. In this table, the allowable PIN extraction methods are listed for each PIN block format. The first PIN extraction method keyword listed for a PIN block format is the default. If you specify a PIN extraction method keyword that is not the default, the request will be routed to the PCI Cryptographic Coprocessor on the z900 server.

Table 178. PIN Block Format and PIN Extraction Method Keywords

PIN Block Format	PIN Extraction Method Keywords	Description
ECI-2	PINLEN04	The PIN extraction method keywords specify a PIN extraction method for a PINLEN04 format.
ECI-3	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
ISO-0	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
ISO-1	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
ISO-2	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
ISO-3	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
VISA-2	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
VISA-3	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.
VISA-4	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.

Table 178. PIN Block Format and PIN Extraction Method Keywords (continued)

PIN Block Format	PIN Extraction Method Keywords	Description
3621	PADDIGIT, HEXDIGIT, PINLEN04 to PINLEN12, PADEXIST	The PIN extraction method keywords specify a PIN extraction method for an IBM 3621 PIN block format. The first keyword, PADDIGIT, is the default PIN extraction method for the PIN block format.
3624	PADDIGIT, HEXDIGIT, PINLEN04 to PINLEN16, PADEXIST	The PIN extraction method keywords specify a PIN extraction method for an IBM 3624 PIN block format. The first keyword, PADDIGIT, is the default PIN extraction method for the PIN block format.
4704-EPP	PINBLOCK	The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.

The PIN extraction methods operate as follows:

#### **PINBLOCK**

Specifies that the service use one of these:

- the PIN length, if the PIN block contains a PIN length field
- the PIN delimiter character, if the PIN block contains a PIN delimiter character.

#### **PADDIGIT**

Specifies that the service use the pad value in the PIN profile to identify the end of the PIN.

#### **HEXDIGIT**

Specifies that the service use the first occurrence of a digit in the range from X'A' to X'F' as the pad value to determine the PIN length.

#### **PINLEN<sub>xx</sub>**

Specifies that the service use the length specified in the keyword, where xx can range from 4 to 16 digits, to identify the PIN.

#### **PADEXIST**

Specifies that the service use the character in the 16th position of the PIN block as the value of the pad value.

## **Enhanced PIN Security Mode**

An Enhanced PIN Security Mode is available. This optional mode is selected by enabling the PTR Enhanced PIN Security access control point in the PCICC, PCIXCC, or CCA Crypto Express coprocessor default role. When active, this control point affects all PIN callable services that extract or format a PIN using a PIN-block format of 3621 or 3624 with a PIN-extraction method of PADDIGIT.

Table 179 summarizes the callable services affected by the Enhanced PIN Security Mode and describes the effect that the mode has when the access control point is enabled.

*Table 179. Callable Services Affected by Enhanced PIN Security Mode*

PIN-block format and PIN-extraction method	Callable Services Affected	PIN processing changes when Enhanced PIN Security Mode enabled
ECI-2, 3621, or 3624 formats AND PINLENxx	PIN-block format and PIN-extraction method  Clear_PIN_Generate_Alternate  Encrypted_PIN_Translate  Encrypted_PIN_Verify	The PINLENxx keyword in rule_array parameter for PIN extraction method is not allowed if the Enhanced PIN Security Mode is enabled. <b>Note:</b> The services will fail with return code 8 reason code '7E0'x.
3621 or 3624 format and PADDIGIT	Clear_PIN_Generate_Alternate  Encrypted_PIN_Translate  Encrypted_PIN_Verify  PIN Change/Unblock	PIN extraction determines the PIN length by scanning from right to left until a digit, not equal to the pad digit, is found. The minimum PIN length is set at four digits, so scanning ceases one digit past the position of the 4th PIN digit in the block.
3621 or 3624 format and PADDIGIT	Clear_PIN_Encrypt  Encrypted_PIN_Generate  Encrypted_PIN_Translate	PIN formatting does not examine the PIN, in the output PIN block, to see if it contains the pad digit.
3621 or 3624 format and PADDIGIT	Encrypted_PIN_Translate	Restricted to non-decimal digit for PAD digit.

## Format Control

This keyword specifies whether there is any control on the user-supplied PIN format. The 8-byte value must be left-justified and padded with blanks. Specify one of these values:

### NONE

No format control.

**PBVC** A PIN block variant constant (PBVC) enforces format control. Use the PBVC value only if you have coded PBVC in the encrypted PIN translate callable service. For the PBVC, the clear PIN key-encrypting key has been exclusive ORed with one of the PIN block formats. The cryptographic feature removes the pattern from the clear PIN key-encrypting key prior to it decrypting the PIN block.

**Restriction:** PBVC is not supported on IBM @server zSeries 990 and subsequent releases.

### Notes:

1. Only control vectors and extraction methods valid for the Cryptographic Coprocessor Feature may be used if the PBVC format control is desired.

2. PBVC is supported for compatibility with prior releases of OS/390 ICSF and existing ICSF applications. It is recommended that a format control of NONE be specified for maximum flexibility to run on PCI Cryptographic Coprocessors.

If you do not specify a value for the format control parameter, ICSF uses hexadecimal zeros.

Table 194 on page 490 lists the PIN block variant constants.

## Pad Digit

Some PIN formats require this parameter. If the PIN format does not need a pad digit, the callable service ignores this parameter. Table 180 shows the format of a pad digit. The PIN profile pad digit must be specified in upper case.

*Table 180. Format of a Pad Digit*

Bytes	Description
16–22	Seven space characters
23	Character representation of a hexadecimal pad digit or a space if a pad digit is not needed. Characters must be one of these: 0–9, A–F, or a blank.

Each PIN format supports only a pad digit in a certain range. This table lists the valid pad digits for each PIN block format.

*Table 181. Pad Digits for PIN Block Formats*

PIN Block Format	Output PIN Profile	Input PIN Profile
ECI-2	Pad digit is not used	Pad digit is not used
ECI-3	Pad digit is not used	Pad digit is not used
ISO-0	F	Pad digit is not used
ISO-1	Pad digit is not used	Pad digit is not used
ISO-2	Pad digit is not used	Pad digit is not used
ISO-3	Pad digit is not used	Pad digit is not used
VISA-2	0 through 9	Pad digit is not used
VISA-3	0 through F	Pad digit is not used
VISA-4	F	Pad digit is not used
3621	0 through F	0 through F
3624	0 through F	0 through F
4704-EPP	F	Pad digit is not used

The callable service returns an error indicating that the PAD digit is not valid if all of these conditions are met:

1. The PTR Enhanced Security access control point is enabled in the active role
2. The output PIN profile specifies 3621 or 3624 as the PIN-block format
3. The output PIN profile specifies a decimal digit (0-9) as the PAD digit

## Recommendations for the Pad Digit

IBM recommends that you use a nondecimal pad digit in the range of A through F when processing IBM 3624 and IBM 3621 PIN blocks. If you use a decimal pad digit, the creator of the PIN block must ensure that the calculated PIN does not contain the pad digit, or unpredictable results may occur.

For example, you can exclude a specific decimal digit from being in any calculated PIN by using the IBM 3624 calculation procedure and by specifying a decimalization table that does not contain the desired decimal pad digit.

## Current Key Serial Number

The current key serial number is the concatenation of the initial key serial number (a 59-bit value) and the encryption counter (a 21-bit value). The concatenation is an 80-bit (10-byte) value. Table 182 shows the format of the current key serial number.

When UKPT or DUKPT is specified, the PIN profile parameter is extended to a 48-byte field and must contain the current key serial number.

*Table 182. Format of the Current Key Serial Number Field*

Bytes	Description
24–47	Character representation of the current key serial number used to derive the initial PIN encrypting key. It is left justified and padded with 4 blanks.

## Decimalization Tables

Decimalization tables can be loaded in the coprocessors to restrict attacks using modified tables. The management of the tables requires a TKE Workstation.

Clear PIN Generate (CSNBPGN and CSNEPGN), Clear PIN Generate Alternate (CSNBCPA and CSNECPA), Encrypted PIN Generate (CSNBEPG and CSNEEPG), and Encrypted PIN Verify (CSNBPVR and CSNEPVR) callable services will make use of the stored decimalization tables.

The **ANSI X9.8 PIN – Use stored decimalization tables only** access control point is used to restrict the use of tables. When the access control point is enabled, the table supplied by the callable service will be compared against the active tables stored in the coprocessor. If the supplied table doesn't match any of the active tables, the request will fail.

A TKE workstation (Version 7.1 or later) is required to manage the PIN decimalization tables. The tables must be loaded and then activated. Only active tables are checked when the access control point is enabled.

**Note:** ICSF routes work to all active coprocessors based on work load. All coprocessors must have the same set of decimalization tables for the decimalization table access control point to be effective.

---

## Clear PIN Encrypt (CSNBCPE and CSNECPE)

The Clear PIN Encrypt callable service formats a PIN into one of these PIN block formats and encrypts the results. You can use this service to create an encrypted PIN block for transmission. With the RANDOM keyword, you can have the service generate random PIN numbers.

**Note:** A clear PIN is a sensitive piece of information. Ensure that your application program and system design provide adequate protection for any clear PIN value.

- IBM 3621 format
- IBM 3624 format
- ISO-0 format (same as the ANSI X9.8, VISA-1, and ECI formats)
- ISO-1 format (same as the ECI-4 format)
- ISO-2 format
- ISO-3 format
- IBM 4704 encrypting PINPAD (4704-EPP) format
- VISA 2 format
- VISA 3 format
- VISA 4 format
- ECI2 format
- ECI3 format

An enhanced PIN security mode, on PCICCC, PCIXCC, and CCA Crypto Express coprocessors, is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. To do this, you must enable the PTR Enhanced PIN Security access control point in the default role. When activated, this mode limits checking of the PIN to decimal digits. No other PIN block consistency checking will occur.

The callable service name for AMODE(64) invocation is CSNECPE.

## Format

```
CALL CSNBCPE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    PIN_encrypting_key_identifier,
    rule_array_count,
    rule_array,
    clear_PIN,
    PIN_profile,
    PAN_data,
    sequence_number,
    encrypted_PIN_block )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer



## Clear PIN Encrypt

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is defined in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **PIN\_encrypting\_key\_identifier**

Direction: Input/Output

Type: String

The 64-byte string containing an internal key token or a key label of an internal key token. The internal key token contains the key that encrypts the PIN block. The control vector in the internal key token must specify an OPINENC key type and have the CPINENC usage bit set to 1.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. Valid values are 0 and 1.

### **rule\_array**

Direction: Input

Type: Character string

Keywords that provide control information to the callable service. The keyword is left-justified in an 8-byte field, and padded on the right with blanks. All keywords must be in contiguous storage. The rule array keywords are shown as follows:

*Table 183. Process Rules for the Clear PIN Encryption Callable Service*

Process Rule	Description
ENCRYPT	This is the default. Use of this keyword is optional.
RANDOM	Causes the service to generate a random PIN value. The length of the PIN is based on the value in the <i>clear_PIN</i> variable. Set the value of the clear PIN to zero and use as many digits as the desired random PIN; pad the remainder of the clear PIN variable with space characters.

### **clear\_PIN**

Direction: Input

Type: String

A 16-character string with the clear PIN. The value in this variable must be left-justified and padded on the right with space characters.

#### **PIN\_profile**

Direction: Input

Type: String

A 24-byte string containing three 8-byte elements with a PIN block format keyword, the format control keyword, NONE, and a pad digit as required by certain formats. See “The PIN Profile” on page 471 for additional information.

#### **PAN\_data**

Direction: Input

Type: String

A 12-byte PAN in character format. The service uses this parameter if the PIN profile specifies the ISO-0 or VISA-4 keyword for the PIN block format. Otherwise, ensure that this parameter is a 12-byte variable in application storage. The information in this variable will be ignored, but the variable must be specified.

**Note:** When using the ISO-0 keyword, use the 12 rightmost digits of the PAN data, excluding the check digit. When using the VISA-4 keyword, use the 12 leftmost digits of the PAN data, excluding the check digit.

#### **sequence\_number**

Direction: Input

Type: Integer

The 4-byte character integer. The service currently ignores the value in this variable. For future compatibility, the suggested value is 99999.

#### **encrypted\_PIN\_block**

Direction: Output

Type: String

The field that receives the 8-byte encrypted PIN block.

## **Restrictions**

The format control specified in the PIN profile must be NONE. If PBVC is specified as the format control, the service will fail.

## **Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

SAF will be invoked to check authorization to use the Clear PIN encrypt service and the label of the *PIN\_encrypting\_key\_identifier*.

## **Access Control Point**

The **Clear PIN Encrypt** access control point controls the function of this service.

## **Required Hardware**

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

## Clear PIN Encrypt

Table 184. Clear PIN encrypt required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	ISO-3 PIN block format is not supported.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	ISO-3 PIN block format is not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	

## Clear PIN Generate (CSNBPGN and CSNEPGN)

Use the Clear PIN generate callable service to generate a clear PIN, a PIN validation value (PVV), or an offset according to an algorithm. You supply the algorithm or process rule using the *rule\_array* parameter.

- IBM 3624 (IBM-PIN or IBM-PINO)
- IBM German Bank Pool (GBP-PINO) - not supported on an IBM @server zSeries 990 and subsequent releases.
- VISA PIN validation value (VISA-PVV)
- Interbank PIN (INBK-PIN)

The callable service can execute only when ICSF is in special secure mode. This mode is described in “Special Secure Mode” on page 10.

For guidance information about VISA, see their appropriate publications. The Interbank PIN algorithm is available only on S/390 Enterprise Servers, the S/390 Multiprise, and the IBM @server Zseries.

The callable service name for AMODE(64) invocation is CSNEPGN.

## Format

```
CALL CSNBPGN(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    PIN_generating_key_identifier,  
    rule_array_count,  
    rule_array,  
    PIN_length,  
    PIN_check_length,  
    data_array,  
    returned_result )
```

## Parameters

**return code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

## reason code

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

## exit\_data\_length

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is defined in the *exit\_data* parameter.

**exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

## PIN\_generating\_key\_identifier

Direction: Input/Output

Type: Character string

The 64-byte key label or internal key token that identifies the PIN generation (PINGEN) key. If the *PIN\_generating\_key\_identifier* identifies a key which does not have the default PIN generation key control vector, the request will be routed to a PCI Cryptographic Coprocessor.

**rule\_array\_count**

Direction: Input

Type: Integer

## Clear PIN Generate

The number of process rules specified in the *rule\_array* parameter. The value must be 1.

### **rule\_array**

Direction: Input

Type: Character string

The process rule provides control information to the callable service. Specify one of the values in Table 185. The keyword is left-justified in an 8-byte field, and padded on the right with blanks.

*Table 185. Process Rules for the Clear PIN Generate Callable Service*

Process Rule	Description
GBP-PIN	The IBM German Bank Pool PIN, which uses the institution PINGEN key to generate an institution PIN (IPIN).
GBP-PINO	The IBM German Bank Pool PIN offset, which uses the pool PINGEN key to generate a pool PIN (PPIN). It uses the institution PIN (IPIN) as input and calculates the PIN offset, which is the output. GBP-PINO is not supported on an IBM @server zSeries 990 and subsequent releases.
IBM-PIN	The IBM 3624 PIN, which is an institution-assigned PIN. It does not calculate the PIN offset.
IBM-PINO	The IBM 3624 PIN offset, which is a customer-selected PIN and calculates the PIN offset (the output).
INBK-PIN	The Interbank PIN is generated.
VISA-PVV	The VISA PIN validation value. Input is the customer PIN.

### **PIN\_length**

Direction: Input

Type: Integer

The length of the PIN used for the IBM algorithms only, IBM-PIN or IBM-PINO. Otherwise, this parameter is ignored. Specify an integer from 4 through 16. If the length is greater than 12, the request will be routed to the PCI Cryptographic Coprocessor.

### **PIN\_check\_length**

Direction: Input

Type: Integer

The length of the PIN offset used for the IBM-PINO process rule only. Otherwise, this parameter is ignored. Specify an integer from 4 through 16.

**Note:** The PIN check length must be less than or equal to the integer specified in the *PIN\_length* parameter.

### **data\_array**

Direction: Input

Type: String

Three 16-byte data elements required by the corresponding *rule\_array* parameter. The data array consists of three 16-byte fields or elements whose specification depends on the process rule. If a process rule only requires one or two 16-byte fields, then the rest of the data array is ignored by the callable service. Table 186 describes the array elements.

Table 186. Array Elements for the Clear PIN Generate Callable Service

Array Element	Description
Clear_PIN	Clear user selected PIN of 4 to 12 digits of 0 through 9. Left-justified and padded with spaces. For IBM-PINO, this is the clear customer PIN (CSPIN). For GBP-PINO, this is the institution PIN. For IBM-PIN and GBP-PIN, this field is ignored.
Decimalization_table	Decimalization table for IBM and GBP only. Sixteen digits of 0 through 9. <b>Note:</b> If the <b>ANSI X9.8 PIN – Use stored decimalization tables only</b> access control point is enabled in the ICSF role, this table must match one of the active decimalization tables in the coprocessors.
Trans_sec_parm	For VISA only, the leftmost sixteen digits. Eleven digits of the personal account number (PAN). One digit key index. Four digits of customer selected PIN.  For Interbank only, sixteen digits. Eleven right-most digits of the personal account number (PAN). A constant of 6. One digit key selector index. Three digits of PIN validation data.
Validation_data	Validation data for IBM and IBM German Bank Pool padded to 16 bytes. One to sixteen characters of hexadecimal account data left-justified and padded on the right with blanks.

Table 187 lists the data array elements required by the process rule (*rule\_array* parameter). The numbers refer to the process rule's position within the array.

Table 187. Array Elements Required by the Process Rule

Process Rule	IBM-PIN	IBM-PINO	GBP-PIN	GBP-PINO	VISA-PVV	INBK-PIN
Decimalization_table	1	1	1	1		
Validation_data	2	2	2	2		
Clear_PIN		3		3		
Trans_sec_parm					1	1

**Note:** Generate offset for GBP algorithm is equivalent to IBM offset generation with *PIN\_check\_length* of 4 and *PIN\_length* of 6.

#### returned\_result

Direction: Output

Type: Character string

The 16-byte generated output, left-justified and padded on the right with blanks.

## Restrictions

PIN lengths of 13-16 require the optional PCI Cryptographic Coprocessor.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

If you are using the IBM 3624 PIN and IBM German Bank Pool PIN algorithms, you can supply an unencrypted customer selected PIN to generate a PIN offset.

## Access Control Points

This table shows the access control points in the ICSF role that control the function of this service.

Table 188. Required access control points for Clear PIN Generate

Rule array keywords	Access control point
IBM-PIN IBM-PINO	Clear PIN Generate - 3624
GBP-PIN	Clear PIN Generate - GBP
VISA-PVV	Clear PIN Generate - VISA PVV
INBK-PIN	Clear PIN Generate - Interbank

If the **ANSI X9.8 PIN – Use stored decimalization tables only** access control point is enabled in the ICSF role, any decimalization table specified must match one of the active decimalization tables in the coprocessors.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 189. Clear PIN generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	ICSF routes this service to a PCI Cryptographic Coprocessor if the control vector of the PIN generating key cannot be processed on the Cryptographic Coprocessor Feature.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	<i>Rule_array</i> keyword GBP-PINO is not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	<i>Rule_array</i> keyword GBP-PINO is not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor  Crypto Express3 Coprocessor	<i>Rule_array</i> keyword GBP-PINO is not supported.

Table 189. Clear PIN generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
z196 z114	Crypto Express3 Coprocessor	<i>Rule_array</i> keyword GBP-PINO is not supported.
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	<i>Rule_array</i> keyword GBP-PINO is not supported.

## Related Information

PIN algorithms are shown in PIN Formats and Algorithms.

## Clear PIN Generate Alternate (CSNBCPA and CSNECPA)

Use the clear PIN generate alternate service to generate a clear VISA PVV (PIN validation value) from an input encrypted PIN block, or to produce a 3624 offset from a customer-selected encrypted PIN. The PIN block can be encrypted under either an input PIN-encrypting key (IPINENC) or an output PIN-encrypting key (OPINENC).

An enhanced PIN security mode, on PCICCC, PCIXCC, and CCA Crypto Express coprocessors, is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. To do this, you must enable the PTR Enhanced PIN Security access control point in the default role. When activated, this mode limits checking of the PIN to decimal digits and a PIN length minimum of 4 is enforced. No other PIN-block consistency checking will occur.

An enhanced PIN security mode on the CEX3C and CEX4C is available to implement restrictions required by the ANSI X9.8 PIN standard. To enforce these restrictions, you must enable the following control points in the default role.

- ANSI X9.8 PIN - Enforce PIN block restrictions
- ANSI X9.8 PIN - Allow modification of PAN
- ANSI X9.8 PIN - Allow only ANSI PIN blocks

The callable service name for AMODE(64) invocation is CSNECPA.



## Format

```
CALL CSNBPA(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    PIN_encryption_key_identifier,
    PIN_generation_key_identifier,
    PIN_profile,
    PAN_data,
    encrypted_PIN_block,
    rule_array_count,
    rule_array,
    PIN_check_length,
    data_array,
    returned_PVV)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **PIN\_encryption\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string consisting of an internal token that contains an IPINENC or OPINENC key or the label of an IPINENC or OPINENC key that is used to encrypt the PIN block. If you specify a label, it must resolve uniquely to either an IPINENC or OPINENC key. If the *PIN\_encryption\_key\_identifier* identifies a

key which does not have the default PIN encrypting control vector (either IPINENC or OPINENC), the request will be routed to the PCI Cryptographic Coprocessor for processing.

#### **PIN\_generation\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte string that consists of an internal token that contains a PIN generation (PINGEN) key or the label of a PINGEN key. If the *PIN\_generation\_key\_identifier* identifies a key which does not have the default PIN generating control vector, the request will be routed to the PCI Cryptographic Coprocessor for processing.

#### **PIN\_profile**

Direction: Input

Type: Character string

The three 8-byte character elements that contain information necessary to extract a PIN from a formatted PIN block. The pad digit is needed to extract the PIN from a 3624 or 3621 PIN block in the clear PIN generate alternate service. See “The PIN Profile” on page 471 for additional information.

#### **PAN\_data**

Direction: Input

Type: String

A 12-byte field that contains 12 characters of PAN data. The personal account number recovers the PIN from the PIN block if the PIN profile specifies ISO-0 or VISA-4 block formats. Otherwise it is ignored, but you must specify this parameter.

For ISO-0, use the rightmost 12 digits of the PAN, excluding the check digit.  
For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

#### **encrypted\_PIN\_block**

Direction: Input

Type: String

An 8-byte field that contains the encrypted PIN that is input to the VISA PVV generation algorithm. The service uses the IPINENC or OPINENC key that is specified in the *PIN\_encryption\_key\_identifier* parameter to encrypt the block.

#### **rule\_array\_count**

Direction: Input

Type: Integer

The number of process rules specified in the *rule\_array* parameter. The value may be 1, 2, or 3.

#### **rule\_array**

Direction: Input

Type: Character string

The process rule for the PIN generation algorithm. Specify IBM-PINO or “VISA-PVV” (the VISA PIN verification value) in an 8-byte field, left-justified, and padded with blanks. The *rule\_array* points to an array of one or two 8-byte elements as follows:

## Clear PIN Generate Alternate

Table 190. Rule Array Elements for the Clear PIN Generate Alternate Service

Rule Array Element	Function of Rule Array keyword
1	PIN calculation method
2	PIN extraction method

The first element in the rule array must specify one of the keywords that indicate the PIN calculation method as shown:

Table 191. Rule Array Keywords (First Element) for the Clear PIN Generate Alternate Service

PIN Calculation Method Keyword	Meaning
IBM-PINO	This keyword specifies use of the IBM 3624 PIN Offset calculation method.
VISA-PVV	This keyword specifies use of the VISA PVV calculation method.

If the second element in the rule array is provided, one of the PIN extraction method keywords shown in Table 178 on page 472 may be specified for the given PIN block format. See “PIN Block Format and PIN Extraction Method Keywords” on page 472 for additional information. If the default extraction method for a PIN block format is desired, you may code the rule array count value as 1.

The PIN extraction methods operate as follows:

### **PINBLOCK**

Specifies that the service use one of these:

- the PIN length, if the PIN block contains a PIN length field
- the PIN delimiter character, if the PIN block contains a PIN delimiter character.

### **PADDIGIT**

Specifies that the service use the pad value in the PIN profile to identify the end of the PIN.

### **HEXDIGIT**

Specifies that the service use the first occurrence of a digit in the range from X'A' to X'F' as the pad value to determine the PIN length.

### **PINLENxx**

Specifies that the service use the length specified in the keyword, where xx can range from 4 to 16 digits, to identify the PIN.

### **PADEXIST**

Specifies that the service use the character in the 16th position of the PIN block as the value of the pad value.

### **PIN\_check\_length**

Direction: Input

Type: Integer

The length of the PIN offset used for the IBM-PINO process rule only. Otherwise, this parameter is ignored. Specify an integer from 4 through 16.

**Note:** The PIN check length must be less than or equal to the integer specified in the *PIN\_length* parameter. If the *PIN\_check\_length* variable is greater than the PIN length, the *PIN\_check\_length* variable will be set to the PIN length.

### data\_array

Direction: Input

Type: String

Three 16-byte elements. Table 192 describes the format when IBM-PINO is specified. Table 193 describes the format when VISA-PVV is specified.

*Table 192. Data Array Elements for the Clear PIN Generate Alternate Service (IBM-PINO)*

Array Element	Description
decimalization_table	This element contains the decimalization table of 16 characters (0 to 9) that are used to convert hexadecimal digits (X'0' to X'F') of the enciphered validation data to the decimal digits X'0' to X'9'). <b>Note:</b> If the <b>ANSI X9.8 PIN – Use stored decimalization tables only</b> access control point is enabled in the ICSF role, this table must match one of the active decimalization tables in the coprocessors.
validation_data	This element contains one to 16 characters of account data. The data must be left justified and padded on the right with space characters.
Reserved-3	This field is ignored, but you must specify it.

When using the VISA-PVV keyword, identify these elements in the data array.

*Table 193. Data Array Elements for the Clear PIN Generate Alternate Service (VISA-PVV)*

Array Element	Description
Trans_sec_parm	For VISA-PVV only, the leftmost twelve digits. Eleven digits of the personal account number (PAN). One digit key index. The rest of the field is ignored.
Reserved-2	This field is ignored, but you must specify it.
Reserved-3	This field is ignored, but you must specify it.

### returned\_PVV

Direction: Output

Type: Character

A 16-byte area that contains the 4-byte PVV left-justified and padded with blanks.

## Restrictions

The IBM-PINO PIN calculation method requires a PCICC, PCIXCC, or CCA Crypto Express coprocessor.

On CCF systems, to use an IPINENC key, you must install the NOCV-enablement keys in the CKDS.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Use of the Visa-PVV PIN-calculation method will always output four digits rather than padding the output with binary zeros to the length of the PIN.

On CCF systems, to use an IPINENC key, you must install the NOCV-enablement keys in the CKDS.

This table lists the PIN block variant constants (PBVC) to use.

**Note:** PBVC is supported for compatibility with prior releases of OS/390 ICSF and existing ICSF applications. If PBVC is specified in the format control parameter of the PIN profile, the Clear PIN Generate Alternate service will not be routed to a PCI Cryptographic Coprocessor for processing. This means that only control vectors and extraction methods valid for the Cryptographic Coprocessor Feature may be used if PBVC formatting is desired. It is recommended that a format control of NONE be used for maximum flexibility.

**Restriction:** PBVC is supported only on an IBM zSeries 900.

Table 194. PIN Block Variant Constants (PBVCs)

PIN Format Name	PIN Block Variant Constant (PBVC)
ECI-2	X'000000000000093000000000000009300'
ECI-3	X'000000000000095000000000000009500'
ISO-0	X'000000000000088000000000000008800'
ISO-1	X'00000000000008B000000000000008B00'
VISA-2	X'00000000000008D000000000000008D00'
VISA-3	X'00000000000008E000000000000008E00'
VISA-4	X'000000000000090000000000000009000'
3621	X'000000000000084000000000000008400'
3624	X'000000000000082000000000000008200'
4704-EPP	X'000000000000087000000000000008700'

## Access Control Points

This table shows the access control points in the ICSF role that control the function of this service.

Table 195. Required access control points for Clear PIN Generate Alternate

Rule array keywords	Access control point
IBM-PINO	Clear PIN Generate Alternate - 3624 Offset
VISA-PVV	Clear PIN Generate Alternate - VISA PVV

If the **ANSI X9.8 PIN – Use stored decimalization tables only** access control point is enabled in the ICSF role, any decimalization table specified must match one of the active decimalization tables in the coprocessors.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 196. Clear PIN generate alternate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 990	Cryptographic Coprocessor Feature	<p>If PBVC is specified for format control, the request will be routed to a Cryptographic Coprocessor Feature.</p> <p>ICSF routes the request to a PCI Cryptographic Coprocessor if:</p> <ul style="list-style-type: none"> <li>The <i>PIN_encryption_key_identifier</i> identifies a key which does not have the default PIN encrypting control vector (either IPINENC or OPINENC).</li> <li>IBM-PINO PIN calculation method is specified.</li> <li>Anything is specified other than the default in the PIN extraction method keyword for the given PIN block format in <i>rule_array</i>.</li> </ul>
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	Format control in the PIN profile parameter must specify NONE.
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Format control in the PIN profile parameter must specify NONE.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	Format control in the PIN profile parameter must specify NONE.
z196 z114	Crypto Express3 Coprocessor	Format control in the PIN profile parameter must specify NONE.
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	Format control in the PIN profile parameter must specify NONE.

## CVV Key Combine (CSNBCKC and CSNECKC)

Use this callable service to combine 2 single length CCA internal key tokens into 1 double-length CCA key token containing a CVVKEY-A key type for use with the VISA CVV Service Generate or VISA CVV Service Verify callable services. This combined double-length key satisfies current VISA requirements and eases translation between TR-31 and CCA formats for CVV keys.

The callable service name for AMODE(64) is CSNECKC.

## Format

```
CALL CSNBCKC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_a_identifier_length,
    key_a_identifier,
    key_b_identifier_length,
    key_b_identifier,
    output_key_identifier_length,
    output_key_identifier )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The *rule\_array\_count* parameter must be 0, 1, or 2.

### **rule\_array**

Direction: Input

Type: String

The *rule\_array* contains keywords that provide control information to the callable service. The keywords are 8 bytes in length and must be left-aligned and padded on the right with space characters. The *rule\_array* keywords for this callable service are shown in the following table.

Table 197. Keywords for CVV Key Combine Rule Array Control Information

Keyword	Meaning
<b>Key Wrapping Method (One Optional)</b>	
USECONFIG	Specifies that the configuration setting for the default wrapping method is to be used to wrap the key. This is the default.
WRAP-ENH	Specifies that the new enhanced wrapping method is to be used to wrap the key.
WRAP-ECB	Specifies that the original wrapping method is to be used.
<b>Translation Control (One Optional)</b>	
ENH-ONLY	Specify this keyword to indicate that the key once wrapped with the enhanced method cannot be wrapped with the original method. This restricts translation to the original method. If the keyword is not specified translation to the original method will be allowed. This turns on bit 56 in the control vector. This keyword is not valid if processing a zero CV data key. <b>Note:</b> If the default wrapping method is ECB mode, but the enhanced mode and the ENH-ONLY restriction are desired for a particular key token, combine the ENH-ONLY keyword with the WRAP-ENH keyword. If this is not done, then an error will be returned because ENH-ONLY will conflict with the default wrapping mode if the default wrapping method is ECB mode.

#### **key\_a\_identifier\_length**

Direction: Input

Type: Integer

This parameter specifies the length of the *key\_a\_identifier* parameter, in bytes. The value must be 64.

#### **key\_a\_identifier**

Direction: Input

Type: String

This parameter contains a 64-byte internal key token or a label of a single-length zero CV DATA key, a DATA key with the MAC gen and/or verify bits on, or a CVVKEY-A key. The internal key token contains the key-A key that encrypts information in the CVV process.

#### **key\_b\_identifier\_length**

Direction: Input

Type: Integer

This parameter specifies the length of the *key\_b\_identifier* parameter, in bytes. The value in this parameter must be 64.

#### **key\_b\_identifier**

Direction: Input

Type: String

This parameter contains a 64-byte internal key token or a label of a single-length zero CV DATA key, a DATA key with the MAC gen and/or



## CVV Key Combine

verify bits on, or a CVVKEY-B key. The internal key token contains the key-B key that decrypts information in the CVV process.

### **output\_key\_identifier\_length**

Direction: Input

Type: Integer

This parameter specifies the length of the *output\_key\_identifier* parameter, in bytes. The value in this parameter must be 64.

### **output\_key\_identifier**

Direction: Output

Type: String

This parameter contains the output key token. It is either a double-length DATA key or a MAC key with CV bits 0-3 set to 0010 to indicate a CVVKEY-A key.

## Restrictions

None.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS.

If key-A and key-B have different CV values for either the Export bit (CV bit 17) or the TR-31 Export bit (CV bit 57), then the keys cannot be combined and an error is returned (8 / 39).

Both key-A and key-B must be usable in the same role for either the CVV Generate or CVV Verify service, otherwise an error occurs.

Both key-A and key-B must be usable for the same service (CVV Generate or CVV Verify). It is not acceptable to combine a Generate and a Verify key.

If key-A or key-B is a Generate-Only key and the pair pass all criteria to be combined as a single output key, the resulting CV in the output token will indicate a double-length Generate-Only key capability.

The key types of the *key\_a\_identifier* and *key\_b\_identifier* must be the same unless the **CVV Key Combine – Permit mixed key types** access control point is enable. This means both key identifiers must be DATA keys or both must be MAC keys when the access control point is disabled. When enabled, DATA keys can be used with MAC keys.

This following table shows the action taken by the service for different combinations of input key types.

Table 198. Key type combinations for the CVV key combine callable service

Action taken based on key types of the 2 input keys		8-byte input key provided as right-half (key-B) of 16 B CVV key			
		CVVKEY-A	CVVKEY-B	DATA key	ANY-MAC key
8-byte input key provided as left-half (key-A) of 16 B CVV key	CVVKEY-A	Always reject	Always allow	Conditional allow*	Conditional allow*
	CVVKEY-B	Always reject	Always reject	Always reject	Always reject
	DATA key	Always reject	Conditional allow*	Always allow	Conditional allow*
	ANY-MAC key	Always reject	Conditional allow*	Conditional allow*	Always allow
* – Requires Access Control Point “CVV Key Combine – Permit mixed key types” enabled					

There are restrictions on the available wrapping methods for the output key derived from the wrapping methods employed and CV restrictions of the input keys. These are detailed in the following table.

Table 199. Wrapping combinations for the CVV Combine Callable Service

key-A OR key-B uses WRAP-ENH wrapping method	key-A OR key-B has enhanced-only bit (CV bit 56) set to 1 (implies WRAP-ENH for that token)	WRAP-ENH keyword passed or WRAP-ENH is default wrapping method	ENH-ONLY keyword passed	Outcome (form of output key or error)
no	no	no to both	no	output is ECB wrapped
yes	no	no to both	no	error 8 / 2161
no	no	yes to either	no	output is ENH wrapped, bit 56 not set
yes	no	yes to either	no	output is ENH wrapped, bit 56 not set
no	no	yes to either	yes	output is ENH wrapped, bit 56 is set
yes	no	yes to either	yes	output is ENH wrapped, bit 56 is set
yes	yes	yes to either	no	output is ENH wrapped, bit 56 is set
yes	yes	yes to either	yes	output is ENH wrapped, bit 56 is set
no	no	no to both	yes	error 8 / 2111
yes	no	no to both	yes	error 8 / 2111
yes	yes	no to both	no	error 8 / 2111

Table 199. Wrapping combinations for the CVV Combine Callable Service (continued)

key-A OR key-B uses WRAP-ENH wrapping method	key-A OR key-B has enhanced-only bit (CV bit 56) set to 1 (implies WRAP-ENH for that token)	WRAP-ENH keyword passed or WRAP-ENH is default wrapping method	ENH-ONLY keyword passed	Outcome (form of output key or error)
yes	yes	no to both	yes	error 8 / 2111

## Access Control Points

The access control points in the ICSF role that control the function of this service are:

- CVV Key Combine
- CVV Key Combine – Allow wrapping override keywords
- CVV Key Combine – Permit mixed key types

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 200. TR-31 export required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900		This service is not supported.
IBM @server zSeries 990		This service is not supported.
IBM @server zSeries 890		
IBM System z9 EC		This service is not supported.
IBM System z9 BC		
IBM System z10 EC		This service is not supported.
IBM System z10 BC		
z196 z114	Crypto Express3 Coprocessor	This service requires the Sep. 2011 or later LIC.
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## Encrypted PIN Generate (CSNBEPG and CSNEEPG)

The Encrypted PIN Generate callable service formats a PIN and encrypts the PIN block. To generate the PIN, the service uses one of these PIN calculation methods:

- IBM 3624 PIN
- IBM German Bank Pool Institution PIN
- Interbank PIN

To format the PIN, the service uses one of these PIN block formats:

- IBM 3621 format
- IBM 3624 format
- ISO-0 format (same as the ANSI X9.8, VISA-1, and ECI-1 formats)
- ISO-1 format (same as the ECI-4 format)
- ISO-2 format
- ISO-3 format
- IBM 4704 encrypting PINPAD (4704-EPP) format
- VISA 2 format
- VISA 3 format
- VISA 4 format
- ECI-2 format
- ECI-3 format

An enhanced PIN security mode, on PCICCC, PCIXCC, CEX2C, and CEX3C, is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. To do this, you must enable the PTR Enhanced PIN Security access control point in the default role. When activated, this mode limits checking of the PIN to decimal digits. No other PIN block consistency checking will occur.

The callable service name for AMODE(64) invocation is CSNEEPG.

### Format

```
CALL CSNBEPG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    PIN_generating_key_identifier,
    outbound_PIN_encrypting_key_identifier,
    rule_array_count,
    rule_array,
    PIN_length,
    data_array,
    PIN_profile,
    PAN_data,
    sequence_number,
    encrypted_PIN_block )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

## Encrypted PIN Generate

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFFFF' (2 gigabytes). The data is defined in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **PIN\_generating\_key\_identifier**

Direction: Input/Output

Type: String

The 64-byte internal key token or a key label of an internal key token in the CKDS. The internal key token contains the PIN-generating key. The control vector must specify the PINGEN key type and have the EPINGEN usage bit set to 1.

### **outbound\_PIN\_encrypting\_key\_identifier**

Direction: Input

Type: String

A 64-byte internal key token or a key label of an internal key token in the CKDS. The internal key token contains the key to be used to encrypt the formatted PIN and must contain a control vector that specifies the OPINENC key type and has the EPINGEN usage bit set to 1.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The value must be 1.

### **rule\_array**

Direction: Input

Type: Character string

Keywords that provide control information to the callable service. Each keyword is left-justified in an 8-byte field, and padded on the right with blanks. All keywords must be in contiguous storage. The rule array keywords are shown as follows:

Table 201. Process Rules for the Encrypted PIN Generate Callable Service

Process Rule	Description
GBP-PIN	This keyword specifies the IBM German Bank Pool Institution PIN calculation method is to be used to generate a PIN.
IBM-PIN	This keyword specifies the IBM 3624 PIN calculation method is to be used to generate a PIN.
INBK-PIN	This keyword specifies the Interbank PIN calculation method is to be used to generate a PIN.

## PIN\_length

Direction: Input

Type: Integer

A integer defining the PIN length for those PIN calculation methods with variable length PINs; otherwise, the variable should be set to zero.

## data\_array

Direction: Input

Type: String

Three 16-byte character strings, which are equivalent to a single 48-byte string. The values in the data array depend on the keyword for the PIN calculation method. Each element is not always used, but you must always declare a complete data array. The numeric characters in each 16-byte string must be from 1 to 16 bytes in length, uppercase, left-justified, and padded on the right with space characters. Table 202 describes the array elements.

Table 202. Array Elements for the Encrypted PIN Generate Callable Service

Array Element	Description
Decimalization_table	Decimalization table for IBM and GBP only. Sixteen characters that are used to map the hexadecimal digits (X'0' to X'F') of the encrypted validation data to decimal digits (X'0' to X'9'). <b>Note:</b> If the <b>ANSI X9.8 PIN – Use stored decimalization tables only</b> access control point is enabled in the ICSF role, this table must match one of the active decimalization tables in the coprocessors.
Trans_sec_parm	For Interbank only, sixteen digits. Eleven right-most digits of the personal account number (PAN). A constant of 6. One digit key selector index. Three digits of PIN validation data.
Validation_data	Validation data for IBM and IBM German Bank Pool padded to 16 bytes. One to sixteen characters of hexadecimal account data left-justified and padded on the right with blanks.

Table 203 on page 500 lists the data array elements required by the process rule (*rule\_array* parameter). The numbers refer to the process rule's position within the array.

## Encrypted PIN Generate

Table 203. Array Elements Required by the Process Rule

Process Rule	IBM-PIN	GBP-PIN	INBK-PIN
Decimalization_table	1	1	
Validation_data	2	2	
Trans_sec_parm			1

### PIN\_profile

Direction: Input

Type: String array

A 24-byte string containing the PIN profile including the PIN block format. See “The PIN Profile” on page 471 for additional information.

### PAN\_data

Direction: Input

Type: String

A 12-byte string that contains 12 digits of Personal Account Number (PAN) data. The service uses this parameter if the PIN profile specifies the ISO-0 or VISA-4 keyword for the PIN block format. Otherwise, ensure that this parameter is a 4-byte variable in application storage. The information in this variable will be ignored, but the variable must be specified.

**Note:** When using the ISO-0 keyword, use the 12 rightmost digit of the PAN data, excluding the check digit. When using the VISA-4 keyword, use the 12 leftmost digits of the PAN data, excluding the check digit.

### sequence\_number

Direction: Input

Type: Integer

The 4-byte string that contains the sequence number used by certain PIN block formats. The service uses this parameter if the PIN profile specifies the 3621 or 4704-EPP keyword for the PIN block format. Otherwise, ensure that this parameter is a 4-byte variable in application data storage. The information in the variable will be ignored, but the variable must be declared. To enter a sequence number, do this:

- Enter 99999 to use a random sequence number that the service generates.
- For the 3621 PIN block format, enter a value in the range from 0 to 65535.
- For the 4704-EPP PIN block format, enter a value in the range from 0 to 255.

### encrypted\_PIN\_block

Direction: Output

Type: String

The field where the service returns the 8-byte encrypted PIN.

## Restrictions

The format control specified in the PIN profile must be NONE. If PBVC is specified as the format control, the service will fail.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

SAF will be invoked to check authorization to use the Encrypted PIN Generate service and any key labels specified as input.

## Access Control Points

This table shows the access control points in the ICSF role that control the function of this service.

Table 204. Required access control points for Encrypted PIN Generate

Rule array keywords	Access control point
IBM-PIN	Encrypted PIN Generate - 3624
GBP-PIN	Encrypted PIN Generate - GBP
INBK-PIN	Encrypted PIN Generate - Interbank

If the **ANSI X9.8 PIN – Use stored decimalization tables only** access control point is enabled in the ICSF role, any decimalization table specified must match one of the active decimalization tables in the coprocessors.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 205. Encrypted PIN generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	ISO-3 PIN block format is not supported.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	ISO-3 PIN block format is not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	



### Encrypted PIN Translate (CSNBPTR and CSNEPTR)

Use the encrypted PIN translate callable service to reencipher a PIN block from one PIN-encrypting key to another and, optionally, to change the PIN block format, such as the pad digit or sequence number.

The unique-key-per-transaction key derivation for single and double-length keys is available for the encrypted PIN translate service. This support is available for the *input\_PIN\_encrypting\_key\_identifier* and the *output\_PIN\_encrypting\_key\_identifier* parameters for both REFORMAT and TRANSLAT process rules. The *rule\_array* keyword determines which PIN key(s) are derived key(s).

The encrypted PIN translate service can be used for unique-key-per-transaction key derivation.

An enhanced PIN security mode, on PCICCC, PCIXCC, and CCA Crypto Express coprocessors, is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. To do this, you must enable the PTR Enhanced PIN Security access control point in the default role. When activated, this mode limits checking of the PIN to decimal digits. No other PIN block consistency checking will occur.

The enhanced PIN security mode also extracts PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. You must enable the Enhanced PIN Security access control point in the default role. When activated, this mode limits checking of the PIN to decimal digits and a PIN length minimum of 4 is enforced. As with formatting an encrypted PIN block, no other PIN-block consistency checking will occur.

An enhanced PIN security mode on the CEX3C and CEX4C is available to implement restrictions required by the ANSI X9.8 PIN standard. To enforce these restrictions, you must enable the following control points in the default role.

- ANSI X9.8 PIN - Enforce PIN block restrictions
- ANSI X9.8 PIN - Allow modification of PAN
- ANSI X9.8 PIN - Allow only ANSI PIN blocks

The callable service name for AMODE(64) invocation is CSNEPTR.

### Format

```
CALL CSNBPTR(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    input_PIN_encrypting_key_identifier,  
    output_PIN_encrypting_key_identifier,  
    input_PIN_profile,  
    PAN_data_in,  
    PIN_block_in,  
    rule_array_count,  
    rule_array,  
    output_PIN_profile,  
    PAN_data_out,  
    sequence_number,  
    PIN_block_out )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **input\_PIN\_encrypting\_key\_identifier**

Direction: Input/Output

Type: String

The input PIN-encrypting key (IPINENC) for the *PIN\_block\_in* parameter specified as a 64-byte internal key token or a key label. If keyword UKPTIPIN, UKPTBOTH, DUKPT-IP or DUKPT-BH is specified in the *rule\_array*, then the *input\_PIN\_encrypting\_key\_identifier* must specify a key token or key label of a KEYGENKY with the UKPT usage bit enabled.

### **output\_PIN\_encrypting\_key\_identifier**

Direction: Input/Output

Type: String

The output PIN-encrypting key (OPINENC) for the *PIN\_block\_out* parameter specified as a 64-byte internal key token or a key label. If keyword UKPTOPIN, UKPTBOTH, DUKPT-OP or DUKPT-BH is specified in the *rule\_array*, then the *output\_PIN\_encrypting\_key\_identifier* must specify a key token or key label of a KEYGENKY with the UKPT usage bit enabled.

### **input\_PIN\_profile**

Direction: Input

Type: Character string

The three 8-byte character elements that contain information necessary to either create a formatted PIN block or extract a PIN from a formatted PIN block. A particular PIN profile can be either an input PIN profile or an output PIN

## Encrypted PIN Translate

profile depending on whether the PIN block is being enciphered or deciphered by the callable service. See “The PIN Profile” on page 471 for additional information.

If you choose the TRANSLAT processing rule (this is not enforced on the PCIXCC or CCA Crypto Express coprocessor) in the *rule\_array* parameter, the *input\_PIN\_profile* and the *output\_PIN\_profile* must specify the same PIN block format. If you choose the REFORMAT processing rule in the *rule\_array* parameter, the input PIN profile and output PIN profile can have different PIN block formats. If you specify UKPTIPIN/DUKPT-IP or UKPTBOTH/DUKPT-BH in the *rule\_array* parameter, then the *input\_PIN\_profile* is extended to a 48-byte field and must contain the current key serial number. See “The PIN Profile” on page 471 for additional information.

The pad digit is needed to extract the PIN from a 3624 or 3621 PIN block in the Encrypted PIN translate callable service with a process rule (*rule\_array* parameter) of REFORMAT. If the process rule is TRANSLAT, the pad digit is ignored.

### PAN\_data\_in

Direction: Input

Type: Character string

The personal account number (PAN) if the process rule (*rule\_array* parameter) is REFORMAT and the input PIN format is ISO-0 or VISA-4 only. Otherwise, this parameter is ignored. Specify 12 digits of account data in character format.

For ISO-0, use the rightmost 12 digits of the PAN, excluding the check digit.

For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

### PIN\_block\_in

Direction: Input

Type: String

The 8-byte enciphered PIN block that contains the PIN to be translated.

### rule\_array\_count

Direction: Input

Type: Integer

The number of process rules specified in the *rule\_array* parameter. The value may be 1, 2 or 3.

### rule\_array

Direction: Input

Type: Character string

The process rule for the callable service.

Table 206. Keywords for Encrypted PIN Translate

Keyword	Meaning
<i>Processing Rules (required)</i>	
REFORMAT	Changes the PIN format, the contents of the PIN block, and the PIN-encrypting key.
TRANSLAT	Changes the PIN-encrypting key only. It does not change the PIN format and the contents of the PIN block.

Table 206. Keywords for Encrypted PIN Translate (continued)

Keyword	Meaning
<i>PIN Block Format and PIN Extraction Method (optional)</i>	See “PIN Block Format and PIN Extraction Method Keywords” on page 472 for additional information and a list of PIN block formats and PIN extraction method keywords. <b>Note:</b> If a PIN extraction method is not specified, the first one listed in Table 178 on page 472 for the PIN block format will be the default.
<b>DUKPT Keywords - Single length key derivation (optional)</b>	
UKPTIPIN	The <i>input_PIN_encrypting_key_identifier</i> is derived as a single length key. The <i>input_PIN_encrypting_key_identifier</i> must be a KEYGENKY key with the UKPT usage bit enabled. The <i>input_PIN_profile</i> must be 48 bytes and contain the key serial number.
UKPTOPIN	The <i>output_PIN_encrypting_key_identifier</i> is derived as a single length key. The <i>output_PIN_encrypting_key_identifier</i> must be a KEYGENKY key with the UKPT usage bit enabled. The <i>output_PIN_profile</i> must be 48 bytes and contain the key serial number.
UKPTBOTH	Both the <i>input_PIN_encrypting_key_identifier</i> and the <i>output_PIN_encrypting_key_identifier</i> are derived as a single length key. Both the <i>input_PIN_encrypting_key_identifier</i> and the <i>output_PIN_encrypting_key_identifier</i> must be KEYGENKY keys with the UKPT usage bit enabled. Both the <i>input_PIN_profile</i> and the <i>output_PIN_profile</i> must be 48 bytes and contain the respective key serial number.
<b>DUKPT Keywords - double length key derivation (optional) - requires May 2004 or later version of Licensed Internal Code (LIC)</b>	
DUKPT-IP	The <i>input_PIN_encrypting_key_identifier</i> is derived as a double length key. The <i>input_PIN_encrypting_key_identifier</i> must be a KEYGENKY key with the UKPT usage bit enabled. The <i>input_PIN_profile</i> must be 48 bytes and contain the key serial number.
DUKPT-OP	The <i>output_PIN_encrypting_key_identifier</i> is derived as a double length key. The <i>output_PIN_encrypting_key_identifier</i> must be a KEYGENKY key with the UKPT usage bit enabled. The <i>output_PIN_profile</i> must be 48 bytes and contain the key serial number.
DUKPT-BH	Both the <i>input_PIN_encrypting_key_identifier</i> and the <i>output_PIN_encrypting_key_identifier</i> are derived as a double length key. Both the <i>input_PIN_encrypting_key_identifier</i> and the <i>output_PIN_encrypting_key_identifier</i> must be KEYGENKY keys with the UKPT usage bit enabled. Both the <i>input_PIN_profile</i> and the <i>output_PIN_profile</i> must be 48 bytes and contain the respective key serial number.

**output\_PIN\_profile**

Direction: Input

Type: Character string

## Encrypted PIN Translate

The three 8-byte character elements that contain information necessary to either create a formatted PIN block or extract a PIN from a formatted PIN block. A particular PIN profile can be either an input PIN profile or an output PIN profile, depending on whether the PIN block is being enciphered or deciphered by the callable service.

- If you choose the TRANSLAT processing rule in the *rule\_array* parameter, the *input\_PIN\_profile* and the *output\_PIN\_profile* must specify the same PIN block format, except on a PCIXCC or CCA Crypto Express coprocessor.
- If you choose the REFORMAT processing rule in the *rule\_array* parameter, the input PIN profile and output PIN profile can have different PIN block formats.
- If you specify UKPTOPIN or UKPTBOTH in the *rule\_array* parameter, then the *output\_PIN\_profile* is extended to a 48-byte field and must contain the current key serial number. See “The PIN Profile” on page 471 for additional information.
- If you specify DUKPT-OP or DUKPT-BH in the *rule\_array* parameter, then the *output\_PIN\_profile* is extended to a 48-byte field and must contain the current key serial number. See “The PIN Profile” on page 471 for additional information.

### PAN\_data\_out

Direction: Input

Type: Character string

The personal account number (PAN) if the process rule (*rule\_array* parameter) is REFORMAT and the output PIN format is ISO-0 or VISA-4 only. Otherwise, this parameter is ignored. Specify 12 digits of account data in character format.

For ISO-0, use the rightmost 12 digits of the PAN, excluding the check digit.

For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

### sequence\_number

Direction: Input

Type: Integer

The sequence number if the process rule (*rule\_array* parameter) is REFORMAT and the output PIN block format is 3621 or 4704-EPP only. Specify the integer value 99999. Otherwise, this parameter is ignored.

### PIN\_block\_out

Direction: Output

Type: String

The 8-byte output PIN block that is reenciphered.

## Restrictions

Use of the ISO-2 PIN block format requires a PCICC, PCIXCC, or CCA Crypto Express coprocessor.

Use of the UKPT keywords require a PCICC, PCIXCC, or CCA Crypto Express coprocessor. Use of the DUKPT keywords require a PCIXCC or CCA Crypto Express coprocessor.

PAD digit restricted to non-decimal digit when Enhanced PIN Security access control point is enabled and if the output PIN profile specifies 3624 or 3621 as the PIN-block format.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

PIN block formats are more rigorously validated on the IBM @server zSeries 990 and subsequent releases than on CCF systems.

Some PIN block formats are known by several names. This table shows the additional names.

*Table 207. Additional Names for PIN Formats*

PIN Format	Additional Name
ISO-0	ANSI X9.8, VISA format 1, ECI format 1
ISO-1	ECI format 4

This table lists the PIN block variant constants (PBVC) to be used.

**Note:** PBVC is NOT supported on the IBM @server zSeries 990 and subsequent releases. If PBVC is specified in the format control parameter of the PIN profile, the Encrypted PIN translate service will not be routed to a PCI Cryptographic Coprocessor for processing. This means that only control vectors and extraction methods valid for the Cryptographic Coprocessor Feature may be used if PBVC formatting is desired. It is recommended that a format control of NONE be used for maximum flexibility.

*Table 208. PIN Block Variant Constants (PBVCs)*

PIN Format Name	PIN Block Variant Constant (PBVC)
ECI-2	X'0000000000009300000000000009300'
ECI-3	X'0000000000009500000000000009500'
ISO-0	X'0000000000008800000000000008800'
ISO-1	X'0000000000008B00000000000008B00'
VISA-2	X'0000000000008D00000000000008D00'
VISA-3	X'0000000000008E00000000000008E00'
VISA-4	X'0000000000009000000000000009000'
3621	X'0000000000008400000000000008400'
3624	X'0000000000008200000000000008200'
4704-EPP	X'0000000000008700000000000008700'

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

*Table 209. Required access control points for Encrypted PIN Translate*

Processing rule	Access control point
TRANSLAT	Encrypted PIN Translate - Translate
REFORMAT	Encrypted PIN Translate - Reformat

## Encrypted PIN Translate

If any of the Unique Key per Transaction rule array keywords are specified, the UKPT - PIN Verify, PIN Translate access control point must be enabled.

### Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 210. Encrypted PIN translate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	If PBVC is specified for format control, the request will be routed to the Cryptographic Coprocessor Feature.  ISO-3 PIN block format is not supported.
	PCI Cryptographic Coprocessor	ICSF routes this service to a PCI Cryptographic Coprocessor if: <ul style="list-style-type: none"><li>• The control vector in a supplied PIN encrypting key cannot be processed on the Cryptographic Coprocessor Feature.</li><li>• UKPT support is requested.</li><li>• The PIN profile specifies the ISO-2 PIN block format.</li><li>• if the <i>input_PIN_encrypting_key_identifier</i> identifies a key which does not have the default input PIN encrypting key control vector (IPINENC)</li><li>• if the <i>output_PIN_encrypting_key_identifier</i> identifies a key which does not have the default output PIN encrypting key control vector (OPINENC)</li><li>• if anything is specified other than the default in the PIN extraction method keyword for the given PIN block format in <i>rule_array</i></li></ul> DUKPT-IP, DUKPT-OP and DUKPT-BH keywords are not supported.  ISO-3 PIN block format is not supported.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	Format control in the PIN profile parameter must specify NONE.
IBM @server zSeries 890	Crypto Express2 Coprocessor	ISO-3 PIN block format is not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Format control in the PIN profile parameter must specify NONE.  ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	Format control in the PIN profile parameter must specify NONE.
	Crypto Express3 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).



Table 210. Encrypted PIN translate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
z196 z114	Crypto Express3 Coprocessor	Format control in the PIN profile parameter must specify NONE.
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	Format control in the PIN profile parameter must specify NONE.

## Encrypted PIN Verify (CSNBPVR and CSNEPVR)

Use the Encrypted PIN verify callable service to verify that one of these customer selected trial PINs is valid:

- IBM 3624 (IBM-PIN)
- IBM 3624 PIN offset (IBM-PINO)
- IBM German Bank Pool (GBP-PIN)
- IBM German Bank Pool PIN offset (GBP-PINO) - not supported on the IBM @server zSeries 990
- VISA PIN validation value (VISA-PVV)
- VISA PIN validation value (VISAPVV4)
- Interbank PIN (INBK-PIN)

The unique-key-par-transaction key derivation for single and double-length keys is available for the *input\_PIN\_encrypting\_key\_identifier* parameter.

An enhanced PIN security mode, on PCICCC, PCIXCC, and CCA Crypto Express coprocessors, is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. To do this, you must enable the PTR Enhanced PIN Security access control point in the default role. When activated, this mode limits checking of the PIN to decimal digits and a PIN length minimum of 4 is enforced. No other PIN-block consistency checking will occur.

The callable service name for AMODE(64) invocation is CSNEPVR.

## Format

```
CALL CSNBPVR(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    input_PIN_encrypting_key_identifier,
    PIN_verifying_key_identifier,
    input_PIN_profile,
    PAN_data,
    encrypted_PIN_block,
    rule_array_count,
    rule_array,
    PIN_check_length,
    data_array )
```



### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

#### **input\_PIN\_encrypting\_key\_identifier**

Direction: Input/Output

Type: String

The 64-byte key label or internal key token containing the PIN-encrypting key (IPINENC) that enciphers the PIN block. If keyword UKPTIPIN or DUKPT-IP is specified in the *rule\_array*, then the *input\_PIN\_encrypting\_key\_identifier* must specify a key token or key label of a KEYGENKY with the UKPT usage bit enabled.

#### **PIN\_verifying\_key\_identifier**

Direction: Input/Output

Type: String

The 64-byte key label or internal key token that identifies the PIN verify (PINVER) key.

#### **input\_PIN\_profile**

Direction: Input

Type: Character string

The three 8-byte character elements that contain information necessary to either create a formatted PIN block or extract a PIN from a formatted PIN block. A particular PIN profile can be either an input PIN profile or an output PIN profile depending on whether the PIN block is being enciphered or deciphered by the callable service. If you specify UKPTIPIN in the *rule\_array* parameter,

then the *input\_PIN\_profile* is extended to a 48-byte field and must contain the current key serial number. See “The PIN Profile” on page 471 for additional information.

If you specify DUKPT-IP in the *rule\_array* parameter, then the *input\_PIN\_profile* is extended to a 48-byte field and must contain the current key serial number. See “The PIN Profile” on page 471 for additional information.

The pad digit is needed to extract the PIN from a 3624 or 3621 PIN block in the encrypted PIN verify callable service.

## PAN\_data

Direction: Input

Type: Character string

The personal account number (PAN) is required for ISO-0 and VISA-4 only. Otherwise, this parameter is ignored. Specify 12 digits of account data in character format.

For ISO-0, use the rightmost 12 digits of the PAN, excluding the check digit.

For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

## encrypted\_PIN\_block

Direction: Input

Type: String

The 8-byte enciphered PIN block that contains the PIN to be verified.

## rule\_array\_count

Direction: Input

Type: Integer

The number of process rules specified in the *rule\_array* parameter. The value may be 1, 2 or 3.

## rule\_array

Direction: Input

Type: Character string

The process rule for the PIN verify algorithm.

Table 211. Keywords for Encrypted PIN Verify

Keyword	Meaning
<i>Algorithm Value (required)</i>	
GBP-PIN	The IBM German Bank Pool PIN. It verifies the PIN entered by the customer and compares that PIN with the institution generated PIN by using an institution key.
GBP-PINO	The IBM German Bank Pool PIN offset. It verifies the PIN entered by the customer by comparing with the calculated institution PIN (IPIN) and adding the specified offset to the pool PIN (PPIN) generated by using a pool key.  GBP-PINO is not supported on the IBM @server zSeries 990.
IBM-PIN	The IBM 3624 PIN, which is an institution-assigned PIN. It does not calculate the PIN offset.

## Encrypted PIN Verify

Table 211. Keywords for Encrypted PIN Verify (continued)

Keyword	Meaning
IBM-PINO	The IBM 3624 PIN offset, which is a customer-selected PIN and calculates the PIN offset.
INBK-PIN	The Interbank PIN verify algorithm.
VISA-PVV	The VISA PIN verify value.
VISAPVV4	The VISA PIN verify value. If the length is 4 digits, normal processing for VISA-PVV will occur. The VISAPVV4 requires a PCICC, PCIXCC, or CCA Crypto Express coprocessor. If one is not available, the service will fail. If the length is greater than 4 digits, the service will fail.
<i>PIN Block Format and PIN Extraction Method (optional)</i>	See “PIN Block Format and PIN Extraction Method Keywords” on page 472 for additional information and a list of PIN block formats and PIN extraction method keywords. <b>Note:</b> If a PIN extraction method is not specified, the first one listed in Table 178 on page 472 for the PIN block format will be the default.
<i>DUKPT Rule (one optional)</i>	
UKPTIPIN	The <i>input_PIN_encrypting_key_identifier</i> is derived as a single length key.. The <i>input_PIN_encrypting_key_identifier</i> must be a KEYGENKY key with the UKPT usage bit enabled. The <i>input_PIN_profile</i> must be 48 bytes and contain the key serial number.
DUKPT-IP	The <i>input_PIN_encrypting_key_identifier</i> is to be derived using the DUKPT algorithm. The <i>input_PIN_encrypting_key_identifier</i> must be a KEYGENKY key with the DUKPT usage bit enabled. The <i>input_PIN_profile</i> must be 48 bytes and contain the key serial number.

### **PIN\_check\_length**

Direction: Input

Type: Integer

The PIN check length for the IBM-PIN or IBM-PINO process rules only. Otherwise, it is ignored. Specify the rightmost digits, 4 through 16, for the PIN to be verified.

### **data\_array**

Direction: Input

Type: String

Three 16-byte elements required by the corresponding *rule\_array* parameter. The data array consists of three 16-byte fields whose specification depend on the process rule. If a process rule only requires one or two 16-byte fields, then the rest of the data array is ignored by the callable service. Table 212 on page 513 describes the array elements.

Table 212. Array Elements for the Encrypted PIN Verify Callable Service

Array Element	Description
Decimalization_table	Decimalization table for IBM and GBP only. Sixteen decimal digits of 0 through 9. <b>Note:</b> If the <b>ANSI X9.8 PIN – Use stored decimalization tables only</b> access control point is enabled in the ICSF role, this table must match one of the active decimalization tables in the coprocessors.
PIN_offset	Offset data for IBM-PINO and GBP-PINO. One to twelve numeric characters, 0 through 9, left-justified and padded on the right with blanks. For IBM-PINO, the PIN offset length is specified in the <i>PIN_check_length</i> parameter. For GBP-PINO, the PIN offset is always 4 digits. For IBM-PIN and GBP-PIN, the field is ignored.
trans_sec_parm	For VISA, only the leftmost twelve digits of the 16-byte field are used. These consist of the rightmost eleven digits of the personal account number (PAN) and a one-digit key index. The remaining four characters are ignored.  For Interbank only, all 16 bytes are used. These consist of the rightmost eleven digits of the PAN, a constant of X'6', a one-digit key index, and three numeric digits of PIN validation data.
RPVV	For VISA-PVV only, referenced PVV (4 bytes) that is left-justified. The rest of the field is ignored.
Validation_data	Validation data for IBM and GBP padded to 16 bytes. One to sixteen characters of hexadecimal account data left-justified and padded on the right with blanks.

Table 213 lists the data array elements required by the process rule (*rule\_array* parameter). The numbers refer to the process rule's position within the array.

Table 213. Array Elements Required by the Process Rule

Process Rule	IBM-PIN	IBM-PINO	GBP-PIN	GBP-PINO	VISA-PVV	INBK-PIN
Decimalization_table	1	1	1	1		
Validation_data	2	2	2	2		
PIN_offset	3	3	3	3		
Trans_sec_parm					1	1
RPVV					2	

## Restrictions

GBP-PINO is only supported if the Encrypted PIN Verify service is processed on the Cryptographic Coprocessor Feature. If the service is routed to a PCI Cryptographic Coprocessor, the service request will fail if the GBP-PINO calculation method is specified. GBP-PINO is not supported on the IBM @server zSeries 990, IBM @server zSeries 890, z9 EC or z9 BC.

Use of the ISO-2 PIN block format requires a PCICC, PCIXCC, or CCA Crypto Express coprocessor.

Use of the UKPTIPIN keyword requires a PCICC, PCIXCC, or CCA Crypto Express coprocessor.

## Encrypted PIN Verify

Use of the VISAPVV4 keyword requires a PCICC, PCIXCC, or CCA Crypto Express coprocessor.

Use of the DUKPT-IP keyword requires a PCIXCC or CCA Crypto Express coprocessor.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

PIN block formats are more rigorously validated on the IBM @server zSeries 990 than on CCF systems.

This table lists the PIN block variant constants (PBVC) to be used.

**Restriction:** PBVC is not supported on an IBM @server zSeries 990. If PBVC is specified in the format control parameter of the PIN profile, the Encrypted PIN Verify service will not be routed to a PCI Cryptographic Coprocessor for processing. This means that only control vectors and extraction methods valid for the Cryptographic Coprocessor Feature may be used if PBVC formatting is desired. It is recommended that a format control of NONE be used for maximum flexibility.

Table 214. PIN Block Variant Constants (PBVCs)

PIN Format Name	PIN Block Variant Constant (PBVC)
ECI-2	X'0000000000009300000000000009300'
ECI-3	X'0000000000009500000000000009500'
ISO-0	X'00000000000088000000000000008800'
ISO-1	X'0000000000008B000000000000008B00'
VISA-2	X'0000000000008D000000000000008D00'
VISA-3	X'0000000000008E000000000000008E00'
VISA-4	X'00000000000090000000000000009000'
3621	X'00000000000084000000000000008400'
3624	X'00000000000082000000000000008200'
4704-EPP	X'00000000000087000000000000008700'

## Access Control Points

This table shows the access control points in the ICSF role that control the function of this service.

Table 215. Required access control points for Encrypted PIN Verify

Process rule	Access control point
IBM-PIN IBM-PINO	Encrypted PIN Verify - 3624
GBP-PIN GBP-PINO	Encrypted PIN Verify - GBP
VISA-PVV	Encrypted PIN Verify - VISA PVV
INBK-PIN	Encrypted PIN Verify - Interbank

If any of the Unique Key per Transaction rule array keywords, the **UKPT - PIN Verify, PIN Translate** access control point must be enabled.

If the **ANSI X9.8 PIN – Use stored decimalization tables only** access control point is enabled in the ICSF role, any decimalization table specified must match one of the active decimalization tables in the coprocessors.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 216. Encrypted PIN verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	If PBVC is specified for format control, the request will be routed to the Cryptographic Coprocessor Feature.  ISO-3 PIN block format is not supported.
	PCI Cryptographic Coprocessor	ICSF routes the request to a PCI Cryptographic Coprocessor if: <ul style="list-style-type: none"> <li>• The PIN profile specifies the ISO-2 PIN block format.</li> <li>• Anything is specified other than the default in the PIN extraction method keyword for the given PIN block format in <i>rule_array</i>.</li> <li>• The <i>input_PIN_encrypting_key_identifier</i> identifies a key which does not have the default PIN encrypting key control vector (IPINENC).</li> <li>• The <i>PIN_verifying_key_identifier</i> identifies a key which does not have the default PIN verify key control vector.</li> <li>• The VISAPVV4 rule array keyword is specified.</li> <li>• You request UKPT support.</li> </ul> The DUKPT-IP keyword is not supported.  ISO-3 PIN block format is not supported.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	Format control in the PIN profile parameter must specify NONE. GBP-PINO rule array parameter is not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	ISO-3 PIN block format is not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Format control in the PIN profile parameter must specify NONE. GBP-PINO rule array parameter is not supported.  ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).

Table 216. Encrypted PIN verify required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor  Crypto Express3 Coprocessor	Format control in the PIN profile parameter must specify NONE. GBP-PINO rule array parameter is not supported.  ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
z196 z114	Crypto Express3 Coprocessor	Format control in the PIN profile parameter must specify NONE. GBP-PINO rule array parameter is not supported.
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	Format control in the PIN profile parameter must specify NONE. GBP-PINO rule array parameter is not supported.

## Related Information

PIN Formats and Algorithms discusses the PIN algorithms in detail.

## PIN Change/Unblock (CSNBPCU and CSNEPCU)

The PIN Change/Unblock callable service is used to generate a special PIN block to change the PIN accepted by an integrated circuit card (smartcard). The special PIN block is based on the new PIN and the card-specific diversified key and, optionally, on the current PIN of the smartcard. The new PIN block is encrypted with a session key. The session key is derived in a two-step process. First, the card-specific diversified key (ICC Master Key) is derived using the TDES-ENC algorithm of the diversified key generation callable service. The session key is then generated according to the rule array algorithm:

- TDES-XOR - XOR ICC Master Key with the Application Transaction Counter (ATC)
- TDESEMV2 - use the EMV2000 algorithm with a branch factor of 2
- TDESEMV4 - use the EMV2000 algorithm with a branch factor of 4

The generating DKYGENKY cannot have replicated halves. The *encryption\_issuer\_master\_key\_identifier* is a DKYGENKY that permits generation of a SMPIN key. The *authentication\_issuer\_master\_key\_identifier* is also a DKYGENKY that permits generation of a double length MAC key.

The PIN block format is specified by the VISA ICC Card specification: mutually exclusive rule array keywords, AMEXPCU1, AMEXPCU2, VISAPCU1 and VISAPCU2. They refer to whether the current PIN is used in the generation of the new PIN.

- VISAPCU1 would create a new PIN for a card without a PIN in an encrypted PIN-block in the new\_reference\_PIN\_block variable. The contents of the five current\_reference\_PIN\_ variables are ignored.
- VISAPCU2 would provide the existing PIN for a card with a current PIN in an encrypted PIN-block in the current\_reference\_PIN\_block variable, and supply the new PIN-value in an encrypted PIN-block in the new\_reference\_PIN\_block variable.

- AMEXPCU1 would create the output PIN from the new-reference PIN, the smart card-unique, intermediate key, and the current-reference PIN.
- AMEXPCU2 would create the output PIN from the new-reference PIN and the smart-card-unique, intermediate key.

An enhanced PIN security mode, is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. To do this, you must enable the PTR Enhanced PIN Security access control point in the default role. When activated, this mode limits checking of the PIN to decimal digits and a PIN length minimum of 4 is enforced. No other PIN-block consistency checking will occur.

The callable service name for AMODE(64) invocation is CSNEPCU.

## Format

```
CALL CSNBPCU(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    authentication_issuer_master_key_length,
    authentication_issuer_master_key_identifier,
    encryption_issuer_master_key_length,
    encryption_issuer_master_key_identifier,
    key_generation_data_length,
    key_generation_data,
    new_reference_PIN_key_length,
    new_reference_PIN_key_identifier,
    new_reference_PIN_block,
    new_reference_PIN_profile,
    new_reference_PIN_PAN_data,
    current_reference_PIN_key_length,
    current_reference_PIN_key_identifier,
    current_reference_PIN_block,
    current_reference_PIN_profile,
    current_reference_PIN_PAN_data,
    output_PIN_data_length,
    output_PIN_data,
    output_PIN_profile,
    output_PIN_message_length,
    output_PIN_message )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned



## PIN Change/Unblock

to it that indicates specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The valid values are 1 and 2.

### **rule\_array**

Direction: Input

Type: String

Keywords that provides control information to the callable service. The keywords are left-justified in an 8-byte field and padded on the right with blanks. The keywords must be in contiguous storage. Specify one or two of these options:

Table 217. Rule Array Keywords for PIN Change/Unblock

Keyword	Meaning
<i>Algorithm (optional)</i>	
TDES-XOR	TDES encipher clear data to generate the intermediate (card-unique) key, followed by XOR of the final 2 bytes of each key with the ATC counter. This is the default.
TDESEMV2	Same processing as in the diversified key generate service.
TDESEMV4	Same processing as in the diversified key generate service.
<i>PIN processing method (required)</i>	
VISAPCU1	Form the new PIN from the new reference PIN and the smart-card-unique, intermediate key.
VISAPCU2	Form the new PIN from the new reference PIN and the smart-card-unique, the intermediate (card-unique) key and the current reference PIN.
AMEXPCU1	Form the new PIN from the new reference PIN, the smart-card-unique, intermediate key, and the current reference PIN.
AMEXPCU2	Form the new PIN from the new reference PIN and the smart-card-unique, intermediate key.

### **authentication\_issuer\_master\_key\_length**

Direction: Input Type: Integer

| The length of the *authentication\_issuer\_master\_key\_identifier* parameter. The value  
| must be 64.

#### **authentication\_issuer\_master\_key\_identifier**

Direction: Input/Output Type: String

The label name or internal token of a DKYGENKY key type that is to be used to generate the card-unique diversified key. The control vector of this key must be a DKYL0 key that permits the generation of a double-length MAC key (DMAC). This DKYGENKY may not have replicated key halves.

#### **encryption\_issuer\_master\_key\_length**

Direction: Input Type: Integer

| The length of the *encryption\_issuer\_master\_key\_identifier* parameter. The value  
| must be 64.

#### **encryption\_issuer\_master\_key\_identifier**

Direction: Input/Output Type: String

The label name or internal token of a DKYGENKY key type that is to be used to generate the card-unique diversified key and the secure messaging session key for the protection of the output PIN block. The control vector of this key must be a DKYL0 key that permits the generation of a SMPIN key type. This DKYGENKY may not have replicated key halves.

#### **key\_generation\_data\_length**

Direction: Input Type: Integer

The length of the *key\_generation\_data* parameter. This value must be 10, 18, 26 or 34 bytes.

#### **key\_generation\_data**

Direction: Input Type: String

The data provided to generate the card-unique session key. For TDES-XOR, this consists of 8 or 16 bytes of data to be processed by TDES to generate the card-unique diversified key followed by a 16 bit ATC counter to offset the card-unique diversified key to form the session key. For TDESEMV2 and TDESEMV4, this may be 10, 18, 26 or 34 bytes. See “Diversified Key Generate (CSNBDBG and CSNEDKG)” on page 122 for more information.

#### **new\_reference\_PIN\_key\_length**

Direction: Input Type: Integer

| The length of the *new\_reference\_PIN\_key\_identifier* parameter. The value must be  
| 64.

#### **new\_reference\_PIN\_key\_identifier**

## PIN Change/Unblock

Direction: Input/Output                      Type: String

The label name or internal token of a PIN encrypting key that is to be used to decrypt the *new\_reference\_PIN\_block*. This must be an IPINENC or OPINENC key. If the label name is supplied, the name must be unique in the CKDS.

### **new\_reference\_PIN\_block**

Direction: Input                                      Type: String

This is an 8-byte field that contains the enciphered PIN block of the new PIN.

### **new\_reference\_PIN\_profile**

Direction: Input                                      Type: String

This is a 24-byte field that contains three 8-byte elements with a PIN block format keyword, a format control keyword (NONE) and a pad digit as required by certain formats.

### **new\_reference\_PIN\_PAN\_data**

Direction: Input                                      Type: String

This is a 12-byte field containing PAN in character format. This data may be needed to recover the new reference PIN if the format is ISO-0 or VISA-4. If neither is used, this parameter may be blanks.

### **current\_reference\_PIN\_key\_length**

Direction: Input                                      Type: Integer

|                      The length of the *current\_reference\_PIN\_key\_identifier* parameter. The value must  
|                      be 64. If the *rule\_array* contains VISAPCU1 or AMEXPCU2, this value must be  
|                      0.

### **current\_reference\_PIN\_key\_identifier**

Direction: Input/Output                      Type: String

|                      The label name or internal token of a PIN encrypting key that is to be used to  
|                      decrypt the *current\_reference\_PIN\_block*. This must be an IPINENC or  
|                      OPINENC key. If the labelname is supplied, the name must be unique on the  
|                      CKDS. If the *rule\_array* contains VISAPCU1 or AMEXPCU2, this value is  
|                      ignored.

### **current\_reference\_PIN\_block**

Direction: Input                                      Type: String

|                      This is an 8-byte field that contains the enciphered PIN block of the new PIN.  
|                      If the *rule\_array* contains VISAPCU1 or AMEXPCU2, this value is ignored.

### **current\_reference\_PIN\_profile**

Direction: Input                                      Type: String

This is a 24-byte field that contains three 8-byte elements with a PIN block format keyword, a format control keyword (NONE) and a pad digit as required by certain formats. If the *rule\_array* contains VISAPCU1 or AMEXPCU2, this value is ignored.

#### **current\_reference\_PIN\_PAN\_data**

Direction: Input

Type: String

This is a 12-byte field containing PAN in character format. If the VISAPCU2 or the AMEXPCU1 *rule\_array* keyword is present and the PIN-profile specifies an ISO-0 or ISO-3 PIN-block format, the variable contains the PAN data. PAN data is used to recover a PIN from an ISO-0 or ISO-3 PIN block.

#### **output\_PIN\_data\_length**

Direction: Input

Type: Integer

The value of this parameter should be 0.

#### **output\_PIN\_data**

Direction: Input

Type: String

This field is reserved.

#### **output\_PIN\_profile**

Direction: Input

Type: String

This is a 24-byte field that contains three 8-byte elements with a PIN block format keyword (VISAPCU1, VISAPCU2, AMEXPCU1 or AMEXPCU2), a format control keyword, NONE, (left aligned and padded on the right with space characters) and 8 byte spaces.

#### **output\_PIN\_message\_length**

Direction: Input/Output

Type: Integer

The length of the *output\_PIN\_message* field. The value must be at least 16. VISAPCU1 and VISAPCU2 and at least 8 for AMEXPCU1 and AMEXPCU2.

#### **output\_PIN\_message**

Direction: Output

Type: String

The reformatted PIN block with the new reference PIN enciphered under the SMPIN session key.

## **Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

## **Access Control Points**

The following table shows the access control points in the ICSF role that control the function of this service.

## PIN Change/Unblock

Table 218. Required access control points for PIN Change/Unblock

PIN-block encrypting key-type	Access control point
OPINENC	PIN Change/Unblock - change EMV PIN with OPINENC
IPINENC	PIN Change/Unblock - change EMV PIN with IPINENC

When the *authentication\_key\_identifier* or *encryption\_key\_identifier* is specified with control vector bits (19 – 22) of B'1111', the **Diversified Key Generate - DKYGENKY – DALL** access control point must also be enabled.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 219. PIN Change/Unblock hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 800 IBM @server zSeries 900		Not Supported
IBM @server zSeries 990 IBM @server zSeries 890	PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor	ISO-3 PIN block format is not supported. AMEXPCU1 and AMEXPCU2 keywords not supported.
IBM System z9 EC and z9 BC	Crypto Express2 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC). AMEXPCU1 and AMEXPCU2 keywords require May, 2012 or later version of LIC.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC). AMEXPCU1 and AMEXPCU2 keywords require May, 2012 or later version of LIC for Crypto Express2. AMEXPCU1 and AMEXPCU2 keywords require June, 2012 or later version of LIC for Crypto Express3.
z196 z114	Crypto Express3 Coprocessor	AMEXPCU1 and AMEXPCU2 keywords require June, 2012 or later version of LIC.
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	AMEXPCU1 and AMEXPCU2 keywords require June, 2012 or later version of LIC for Crypto Express3. AMEXPCU1 and AMEXPCU2 keywords require September, 2012 or later version of LIC for Crypto Express4.

## Secure Messaging for Keys (CSNBSKY and CSNESKY)

The Secure Messaging for Keys callable service will encrypt a text block including a clear key value decrypted from an internal or external DES token. The text block is normally a "Value" field of a secure message TLV (Tag/Length/Value) element of a secure message. TLV is defined in ISO/IEC 7816-4.

The callable service name for AMODE(64) invocation is CSNESKY.

### Format

```
CALL CSNBSKY(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    input_key_identifier,
    key_encrypting_key_identifier,
    secmsg_key_identifier,
    text_length,
    clear_text,
    initialization_vector,
    key_offset,
    key_offset_field_length,
    enciphered_text,
    output_chaining_vector )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

## Secure Messaging for Keys

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The valid values are 0 and 1.

### **rule\_array**

Direction: Input

Type: Character String

Keywords that provides control information to the callable service. The processing method is the encryption mode used to encrypt the message.

*Table 220. Rule Array Keywords for Secure Messaging for Keys*

Keyword	Meaning
<i>Enciphering mode (optional)</i>	
TDES-CBC	Use CBC mode to encipher the message (default).
TDES-ECB	Use ECB mode to encipher the message.

### **input\_key\_identifier**

Direction: Input/Output

Type: String

The internal token, external token, or key label of an internal token of a double length DES key. The key is recovered in the clear and placed in the text to be encrypted. The control vector of the DES key must not prohibit export.

### **key\_encrypting\_key\_identifier**

Direction: Input/Output

Type: String

If the *input\_key\_identifier* is an external token, then this parameter is the internal token or the key label of the internal token of IMPORTER or EXPORTER. If it is not, it is a null token. If a key label is specified, the key label must be unique.

### **secmsg\_key\_identifier**

Direction: Input/Output

Type: String

The internal token or key label of a secure message key for encrypting keys. This key is used to encrypt the updated *clear\_text* containing the recovered DES key.

### **text\_length**

Direction: Input

Type: Integer

The length of the *clear\_text* parameter that follows. Length must be a multiple of eight. Maximum length is 4K.

### **clear\_text**

Direction: Input

Type: String

Clear text that contains the recovered DES key at the offset specified and is then encrypted. Any padding or formatting of the message must be done by the caller on input.

#### **initialization\_vector**

Direction: Input

Type: String

The 8-byte supplied string for the TDES-CBC mode of encryption. The *initialization\_vector* is XORed with the first 8 bytes of *clear\_text* prior to encryption. This field is ignored for TDES-ECB mode.

#### **key\_offset**

Direction: Input

Type: Integer

The offset within the *clear\_text* parameter at *key\_offset* where the recovered clear *input\_key\_identifier* value is to be placed. The first byte of the *clear\_text* field is offset 0.

#### **key\_offset\_field\_length**

Direction: Input

Type: Integer

The length of the field within *clear\_text* parameter at *key\_offset* where the recovered clear *input\_key\_identifier* value is to be placed. Length must be a multiple of eight and is equal to the key length of the recovered key. The key must fit entirely within the *clear\_text*.

#### **enciphered\_text**

Direction: Output

Type: String

The field where the enciphered text is returned. The length of this field must be at least as long as the *clear\_text* field.

#### **output\_chaining\_vector**

Direction: Output

Type: String

This field contains the last 8 bytes of enciphered text and is used as the *initialization\_vector* for the next encryption call if data needs to be chained for TDES-CBC mode. No data is returned for TDES-ECB.

## **Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

SAF will be invoked to check authorization to use the secure messaging for keys service and any key labels specified as input.

Keys only appear in the clear within the secure boundary of the cryptographic coprocessor and never in host storage.

## **Access Control Point**

The **Secure Messaging for Keys** access control point controls the function of this service.



## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 221. Secure messaging for keys required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	

## Secure Messaging for PINs (CSNBSPN and CSNESP)

The Secure Messaging for PINs callable service will encrypt a text block including a clear PIN block recovered from an encrypted PIN block. The input PIN block will be reformatted if the block format in the *input\_PIN\_profile* is different than the block format in the *output\_PIN\_profile*. The clear PIN block will only be self encrypted if the SELFENC keyword is specified in the *rule\_array*. The text block is normally a 'Value' field of a secure message TLV (Tag/Length/Value) element of a secure message. TLV is defined in ISO/IEC 7816-4.

An enhanced PIN security mode on the CEX3C and CEX4C is available to implement restrictions required by the ANSI X9.8 PIN standard. To enforce these restrictions, you must enable the following control points in the default role.

- ANSI X9.8 PIN - Enforce PIN block restrictions
- ANSI X9.8 PIN - Allow modification of PAN
- ANSI X9.8 PIN - Allow only ANSI PIN blocks

The callable service name for AMODE(64) invocation is CSNESP.

## Format

```
CALL CSNBSPN(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    input_PIN_block,
    PIN_encrypting_key_identifier,
    input_PIN_profile,
    input_PAN_data,
    secmsg_key_identifier,
    output_PIN_profile,
    output_PAN_data,
    text_length,
    clear_text,
    initialization_vector,
    PIN_offset,
    PIN_offset_field_length,
    enciphered_text,
    output_chaining_vector )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

## Secure Messaging for PINs

The number of keywords you are supplying in the *rule\_array* parameter. The valid values are 0, 1, or 2.

### **rule\_array**

Direction: Input

Type: Character String

Keywords that provide control information to the callable service. The processing method is the algorithm used to create the generated key. The keywords are left justified and padded on the right with blanks.

Table 222. Rule Array Keywords for Secure Messaging for PINs

Keyword	Meaning
<i>Enciphering mode (optional)</i>	
TDES-CBC	Use CBC mode to encipher the message (default).
TDES-ECB	Use ECB mode to encipher the message.
<i>PIN encryption (optional)</i>	
CLEARPIN	Recovered clear input PIN block (may be reformatted) is placed in the clear in the message for encryption with the secure message key (default).
SELFENC	Recovered clear input PIN block (may be reformatted) is self-encrypted and then placed in the message for encryption with the secure message key.

### **input\_PIN\_block**

Direction: Input

Type: String

The 8-byte input PIN block that is to be recovered in the clear and perhaps reformatted, and then placed in the *clear\_text* to be encrypted.

### **PIN\_encrypting\_key\_identifier**

Direction: Input/Output

Type: String

The internal token or key label of the internal token of the PIN encrypting key used in encrypting the *input\_PIN\_block*. The key must be an IPINENC key.

### **input\_PIN\_profile**

Direction: Input

Type: Character String

The three 8-byte character elements that contain information necessary to extract the PIN from a formatted PIN block. The valid input PIN formats are ISO-0, ISO-1, ISO-2 and ISO-3. See “The PIN Profile” on page 471 for additional information.

### **input\_PAN\_data**

Direction: Input

Type: Character String

The 12 digit personal account number (PAN) if the input PIN format is ISO-0 only. Otherwise, the parameter is ignored.

### **secmsg\_key\_identifier**

Direction: Input/Output

Type: String

The internal token or key label of an internal token of a secure message key for encrypting PINs. This key is used to encrypt the updated *clear\_text*.

**output\_PIN\_profile**

Direction: Input

Type: String

The three 8-byte character elements that contain information necessary to create a formatted PIN block. If reformatting is not required, the *input\_PIN\_profile* and the *output\_PIN\_profile* must specify the same PIN block format. Output PIN block formats supported are ISO-0, ISO-1, ISO-2 and ISO-3.

**output\_PAN\_data**

Direction: Input

Type: String

The 12 digit personal account number (PAN) if the output PIN format is ISO-0 only. Otherwise, this parameter is ignored.

**text\_length**

Direction: Input

Type: Integer

The length of the *clear\_text* parameter that follows. Length must be a multiple of eight. Maximum length is 4K.

**clear\_text**

Direction: Input

Type: String

Clear text that contains the recovered and/or reformatted/encrypted PIN at offset specified and then encrypted. Any padding or formatting of the message must be done by the caller on input.

**initialization\_vector**

Direction: Input

Type: String

The 8-byte supplied string for the TDES-CBC mode of encryption. The *initialization\_vector* is XORed with the first 8 bytes of *clear\_text* prior to encryption. This field is ignored for TDES-ECB mode.

**PIN\_offset**

Direction: Input

Type: Integer

The offset within the *clear\_text* parameter where the reformatted PIN block is to be placed. The first byte of the *clear\_text* field is offset 0.

**PIN\_offset\_field\_length**

Direction: Input

Type: Integer

The length of the field within *clear\_text* parameter at *PIN\_offset* where the recovered clear *input\_PIN\_block* value is to be placed. The PIN block may be

## Secure Messaging for PINs

self-encrypted if requested by the rule array. Length must be eight. The PIN block must fit entirely within the *clear\_text*.

### enciphered\_text

Direction: Output

Type: String

The field where the enciphered text is returned. The length of this field must be at least as long as the *clear\_text* field.

### output\_chaining\_vector

Direction: Output

Type: String

This field contains the last 8 bytes of enciphered text and is used as the *initialization\_vector* for the next encryption call if data needs to be chained for TDES-CBC mode. No data is returned for TDES-ECB.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

SAF will be invoked to check authorization to use the secure messaging for PINs service and any key labels specified as input.

Keys only appear in the clear within the secure boundary of the cryptographic coprocessors and never in host storage.

## Access Control Point

The **Secure Messaging for PINs** access control point controls the function of this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 223. Secure messaging for PINs required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	ISO-3 PIN block format is not supported.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	ISO-3 PIN block format is not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).

Table 223. Secure messaging for PINs required hardware (continued)

Server	Required cryptographic hardware	Restrictions
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	

## SET Block Compose (CSNDSBC and CSNFSBC)

The SET Block Compose callable service performs DES-encryption of data, OAEP-formatting through a series of SHA-1 hashing operations, and the RSA-encryption of the Optimal Asymmetric Encryption Padding (OAEP) block.

The callable service name for AMODE(64) invocation is CSNFSBC.

### Format

```
CALL CSNDSBC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    block_contents_identifier,
    XData_string_length,
    XData_string,
    data_to_encrypt_length,
    data_to_encrypt,
    data_to_hash_length,
    data_to_hash,
    initialization_vector,
    RSA_public_key_identifier_length,
    RSA_public_key_identifier,
    DES_key_block_length,
    DES_key_block,
    RSA_OAEP_block_length,
    RSA_OAEP_block,
    chaining_vector,
    DES_encrypted_data_block )
```

### Parameters

#### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### reason\_code

## SET Block Compose

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The value must be 1 or 2.

### **rule\_array**

Direction: Input

Type: Character String

Keywords that provides control information to the callable service. The keyword must be in 8 bytes of contiguous storage, left-justified and padded on the right with blanks.

*Table 224. Keywords for SET Block Compose Control Information*

Keyword	Meaning
<i>Block Type (required)</i>	
SET1.00	The structure of the RSA-OAEP encrypted block is defined by SET protocol.
<i>Formatting Information (optional)</i>	
DES-ONLY	DES encryption only is to be performed; no RSA-OAEP formatting will be performed. (See Usage Notes.)

### **block\_contents\_identifier**

Direction: Input

Type: String

A one-byte string, containing a binary value that will be copied into the Block Contents (BC) field of the SET DB data block (indicates what data is carried in the Actual Data Block, ADB, and the format of any extra data (*XData\_string*)). This parameter is ignored if DES-ONLY is specified in the rule-array.

### **XData\_string\_length**

Direction: Input Type: Integer

The length in bytes of the data contained within *XData\_string*. The maximum length is 94 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

### **XData\_string**

Direction: Input Type: String

Extra-encrypted data contained within the OAEP-processed and RSA-encrypted block. The format is indicated by *block\_contents\_identifier*. For a *XData\_string\_length* value of zero, *XData\_string* must still be specified, but will be ignored by ICSF. The string is treated as a string of hexadecimal digits. This parameter is ignored if DES-ONLY is specified in the rule-array.

### **data\_to\_encrypt\_length**

Direction: Input/Output Type: Integer

The length in bytes of data that is to be DES-encrypted. The length has a maximum value of 32 MB minus 8 bytes to allow for up to 8 bytes of padding. The data is identified in the *data\_to\_encrypt* parameter. On output, this value is updated with the length of the encrypted data in the *DES\_encrypted\_data\_block*.

### **data\_to\_encrypt**

Direction: Input Type: String

The data that is to be DES-encrypted (with a 64-bit DES key generated by this service). The data will be padded by this service according to the PKCS #5 padding rules.

### **data\_to\_hash\_length**

Direction: Input Type: Integer

The length in bytes of the data to be hashed. The hash is an optional part of the OAEP block. If the *data\_to\_hash\_length* is 0, no hash will be included in the OAEP block. This parameter is ignored if DES-ONLY is specified in the *rule\_array* parameter.

### **data\_to\_hash**

Direction: Input Type: String

The data that is to be hashed and included in the OAEP block. No hash is computed or inserted in the OAEP block if the *data\_to\_hash\_length* is 0. This parameter is ignored if DES-ONLY is specified in the *rule\_array* parameter.

### **initialization\_vector**

Direction: Input Type: String

An 8-byte string containing the initialization vector to be used for the cipher block chaining for the DES encryption of the data in the *data\_to\_encrypt* parameter. The same initialization vector must be used to perform the DES decryption of the data.



## SET Block Compose

### **RSA\_public\_key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *RSA\_public\_key\_identifier* field. The maximum size is 2500 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

### **RSA\_public\_key\_identifier**

Direction: Input

Type: String

A string containing either the key label of the RSA public key or the RSA public key token to be used to perform the RSA encryption of the OAEP block. The modulus bit length of the key must be 1024 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

### **DES\_key\_block\_length**

Direction: Input/Output

Type: Integer

The length of the *DES\_key\_block*. The current length of this field is defined to be exactly 64 bytes.

### **DES\_key\_block**

Direction: Input/Output

Type: String

The DES key information returned from a previous SET Block Compose service. The contents of the *DES\_key\_block* is the 64-byte DES internal key token (containing the DES key enciphered under the host master key). Your application program must not change the data in this string.

### **RSA\_OAEP\_block\_length**

Direction: Input/Output

Type: Integer

The length of a block of storage to hold the *RSA-OAEP\_block*. The length must be at least 128 bytes on input. The length value will be updated on exit with the actual length of the *RSA-OAEP\_block*, which is exactly 128 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

### **RSA\_OAEP\_block**

Direction: Output

Type: String

The OAEP-formatted data block, encrypted under the RSA public key passed as *RSA\_public\_key\_identifier*. When the OAEP-formatted data block is returned, it is left justified within the *RSA-OAEP\_block* field if the input field length (*RSA-OAEP\_block\_length*) was greater than 128 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

### **chaining\_vector**

Direction: Input/Output

Type: String

An 18-byte field that ICSF uses as a system work area. Your application program must not change the data in this string. This field is ignored by this service, but must be specified.

**DES\_encrypted\_data\_block**

Direction: Output

Type: String

The DES-encrypted data block (data passed in as *data\_to\_encrypt*). The length of the encrypted data is returned in *data\_to\_encrypt\_length*. The *DES\_encrypted\_data\_block* may be 8 bytes longer than the length of the *data\_to\_encrypt* because of padding added by this service.

**Restrictions**

Not all CCA implementations support a key label as input in the *RSA\_public\_key\_identifier* parameter. Some implementations may only support a key token.

The *data\_to\_encrypt* and the *DES\_encrypted\_data\_block* cannot overlap.

The maximum data block that can be supplied for DES encryption is the limit as expressed by the Encipher callable service.

**CCF Systems only:** NOCV keys must be installed in the CKDS to use SET block compose service on a CDMF-only system.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

The first time the SET Block Compose service is invoked to form an RSA-OAEP block and DES-encrypt data for communication between a specific source and destination (for example, between the merchant and payment gateway), do not specify the DES-ONLY keyword. A DES key will be generated by the service and returned in the key token contained in the *DES\_key\_block*. On subsequent calls to the Compose SET Block service for communication between the same source and destination, the DES key can be re-used. The caller of the service must supply the *DES\_key\_block*, the *DES\_key\_block\_length*, the *data\_to\_encrypt*, the *data\_to\_encrypt\_length*, and the rule-array keywords SET1.00 and DES-ONLY. You do not need to supply the block contents identifier, XDATA string and length, RSA-OAEP block and length, and RSA public key information, although you must still specify the parameters. For this invocation, the RSA-OAEP formatting is bypassed and only DES encryption is performed, using the supplied DES key.

**Access Control Point**

The **SET Block Compose** access control point controls the function of this service.

**Required Hardware**

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

## SET Block Compose

Table 225. SET block compose required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	If there are no PCI Cryptographic Coprocessors online, the request is routed to the Cryptographic Coprocessor Feature.
	PCI Cryptographic Coprocessor	This service routes the request to a PCI Cryptographic Coprocessor to perform the RSA-OAEP processing.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor	
	Crypto Express4 Coprocessor	

---

## SET Block Decompose (CSNDSBD and CSNFSBD)

Decomposes the RSA-OAEP block and the DES-encrypted data block of the SET protocol to provide unencrypted data back to the caller.

The callable service name for AMODE(64) invocation is CSNFSBD.

## Format

```
CALL CSNDSBD(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    RSA_OAEP_block_length,
    RSA_OAEP_block,
    DES_encrypted_data_block_length,
    DES_encrypted_data_block,
    initialization_vector,
    RSA_private_key_identifier_length,
    RSA_private_key_identifier,
    DES_key_block_length,
    DES_key_block,
    block_contents_identifier,
    XData_string_length,
    XData_string,
    chaining_vector,
    data_block,
    hash_block_length,
    hash_block)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes,” on page 781 lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes,” on page 781 lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

## SET Block Decompose

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The value must be 1 or 2.

### **rule\_array**

Direction: Input

Type: String

One keyword that provides control information to the callable service. The keyword indicates the block type. The keyword must be in 8 bytes of contiguous storage, left-justified and padded on the right with blanks.

*Table 226. Keywords for SET Block Compose Control Information*

Keyword	Meaning
<i>Block Type (required)</i>	
SET1.00	The structure of the RSA-OAEP encrypted block is defined by SET protocol.
<i>Formatting Information (optional)</i>	
DES-ONLY	DES decryption only is to be performed; no RSA-OAEP block decryption will be performed. (See Usage Notes.)
PINBLOCK	Specifies that the OAEP block will contain PIN information in the XDATA field, including an ISO-0 format PIN block. The <i>DES_key_block</i> must be 128 bytes in length and contain a IPINENC or OPINENC key. The PIN block will be encrypted under the PIN encrypting key. The PIN information and the encrypted PIN block are returned in the <i>XDATA_string</i> parameter.

### **RSA\_OAEP\_block\_length**

Direction: Input

Type: Integer

The length of *RSA-OAEP\_block* must be 128 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

### **RSA\_OAEP\_block**

Direction: Input

Type: String

The RSA-encrypted OAEP-formatted data block. This parameter is ignored if DES-ONLY is specified in the rule-array.

### **DES\_encrypted\_data\_block\_length**

Direction: Input/Output

Type: Integer

The length in bytes of the DES-encrypted data block. The input length must be a multiple of 8 bytes. Updated on return to the length of the decrypted data returned in *data\_block*. The maximum value of *DES\_encrypted\_data\_block\_length* is 32MB bytes.

### **DES\_encrypted\_data\_block**

Direction: Input

Type: String

The DES-encrypted data block. The data will be decrypted and passed back as *data\_block*.

#### **initialization\_vector**

Direction: Input

Type: String

An 8-byte string containing the initialization vector to be used for the cipher block chaining for the DES decryption of the data in the *DES\_encrypted\_data\_block* parameter. You must use the same initialization vector that was used to perform the DES encryption of the data.

#### **RSA\_private\_key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *RSA\_private\_key\_identifier* field. The maximum size is 2500 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

#### **RSA\_private\_key\_identifier**

Direction: Input

Type: String

A key label of the RSA private key or an internal token of the RSA private key to be used to decipher the RSA-OAEP block passed in *RSA-OAEP\_block*. The modulus bit length of the key must be 1024. This parameter is ignored if DES-ONLY is specified in the rule-array.

#### **DES\_key\_block\_length**

Direction: Input/Output

Type: Integer

The length of the *DES\_key\_block*. The current length of this field may be 64 or 128 bytes. If rule array keyword PINBLOCK is specified, the length must be 128 bytes.

#### **DES\_key\_block**

Direction: Input/Output

Type: String

The *DES\_key\_block* contains either one or two DES internal key tokens. If only one token is specified on input, it contains either a null DES token (or binary zeros) or (if DES-ONLY is specified) the DES key information returned from a previous SET Block Decompose service invocation. This is the 64-byte DES internal key token formed with the DES key which was retrieved from the RSA-OAEP block and enciphered under the host master key. Your application must not change this DES key information. If two tokens are specified in the *DES\_key\_block*, the first 64 bytes contain the DES token described previously. The second 64 bytes, used when PINBLOCK is specified in the rule array, contains the DES internal token or the CKDS key label of the IPINENC or OPINENC key used to encrypt the PIN block returned to the caller in the *XDATA\_string* parameter. If a key label is specified, it must be left-justified and padded on the right with blanks.

#### **block\_contents\_identifier**

Direction: Output

Type: String

## SET Block Decompose

A one-byte string, containing the binary value from the block contents (BC) field of the SET data block (DB). It indicates what data is carried in the actual data block (ADB) and the format of any extra data (*XData\_string*). This parameter is ignored if DES-ONLY is specified in the rule-array.

### **XData\_string\_length**

Direction: Input/Output

Type: Integer

The length of a string where the data contained within *XData\_string* will be returned. The string must be at least 94 bytes in length. The value will be updated upon exit with the actual length of the returned *XData\_string*. This parameter is ignored if DES-ONLY is specified in the rule-array.

### **XData\_string**

Direction: Output

Type: String

Extra-encrypted data contained within the OAEP-processed and RSA-encrypted block. The format is indicated by *block\_contents\_identifier*. The string is treated by ICSF as a string of hexadecimal digits. The service will always return the data from the beginning of the *XDataString* to the end of the SET DB block, a maximum of 94 bytes of data. The caller must examine the value returned in *block\_contents\_identifier* to determine the actual length of the *XDataString*. This parameter is ignored if DES-ONLY is specified in the rule-array.

### **chaining\_vector**

Direction: Input/Output

Type: String

An 18-byte field that ICSF uses as a system work area. Your application program must not change the data in this string. This field is ignored by this service, but must be specified.

### **data\_block**

Direction: Output

Type: String

The data that was decrypted (passed in as *DES\_encrypted\_data\_block*). Any padding characters are removed.

### **hash\_block\_length**

Direction: Input/Output

Type: Integer

The length in bytes of the SHA-1 hash returned in *hash\_block*. On input, this parameter must be set to the length of the *hash\_block* field. The length must be at least 20 bytes. On output, this field is updated to reflect the length of the SHA-1 hash returned in the *hash\_block* field (exactly 20 bytes). This parameter is ignored if DES-ONLY is specified in the *rule\_array* parameter.

### **hash\_block**

Direction: Output

Type: String

The SHA-1 hash extracted from the RSA-OAEP block. This parameter is ignored if DES-ONLY is specified in the *rule\_array* parameter.

## Restrictions

Not all CCA implementations support a key label as input in the *RSA\_private\_key\_identifier* parameter. Some implementations may only support a key token.

The RSA private key used by this service must have been generated as a signature-only key. This restriction does not apply if you are running on the IBM @server zSeries 990 and subsequent releases.

The *data\_block* and the *DES\_encrypted\_data\_block* cannot overlap.

**CCF Systems only:** The ANSI system keys must be installed in the CKDS to use the SET block decompose service on a CDMF-only system.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

When the SET Block Decompose service is invoked without the DES-ONLY keyword, the DES key is retrieved from the RSA-OAEP block and returned in the key token contained in the *DES\_key\_block*. On subsequent calls to the SET Block Decompose service, a caller can re-use the DES key. The caller of the service must supply the *DES\_key\_block*, the *DES\_key\_block\_length*, the *DES\_encrypted\_data\_block*, the *DES\_encrypted\_data\_block\_length*, the initialization and chaining vectors, and the *rule\_array* keywords SET1.00 and DES-ONLY. The RSA private key information, RSA-OAEP block and length, XData string and length, and hash block and length need not be supplied (although the parameters must still be specified). For this invocation, the decryption of the RSA-OAEP block is bypassed; only DES decryption is performed, using the supplied DES key.

When the SET Block Decompose service is invoked with the PINBLOCK keyword, DES-ONLY may not also be specified. If both of these rule array keywords are specified, the service will fail. If PINBLOCK is specified and the *DES\_key\_block\_length* field is not 128, the service will fail.

## Access Control Points

The **SET Block Decompose** access control point controls the function of this service. If a PIN-block encrypting key is supplied in the *DES\_key\_block*, the access control point matching the key type of the key must be enabled in the ICSF role.

Table 227. Required access control points for PIN-block encrypting key

PIN-block encrypting key-type	Access control point
OPINENC	SET Block Decompose - PIN Extension OPINENC
IPINENC	SET Block Decompose - PIN Extension IPINENC

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.



## SET Block Decompose

Table 228. SET block decompose required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	If there is no PCI Cryptographic Coprocessor available, the request will be processed on the Cryptographic Coprocessor Feature.
	PCI Cryptographic Coprocessor	A PCI Cryptographic Coprocessor is required if: <ul style="list-style-type: none"><li>the <i>RSA_private_key_identifier</i> specifies a retained private key</li><li>the <i>RSA_private_key_identifier</i> specifies a CRT private key</li><li>the PINBLOCK rule array keyword is specified</li></ul> The service has a preference for being processed on a PCI Cryptographic Coprocessor so that the symmetric key does not appear in the clear.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	

## Transaction Validation (CSNBTRV and CSNETRV)

The transaction validation callable service supports the generation and validation of American Express card security codes (CSC). This service generates and verifies transaction values based on information from the transaction and a cryptographic key. You select the algorithm, validation method, and either the generate or verify mode, through rule-array keywords.

For the American Express process, the control vector supplied with the cryptographic key must indicate a MAC or MACVER class key. The key may be single or double length. DATAM and DATAMV keys are not supported. The MAC

generate control vector bit must be on (bit 20) if you request CSC generation and MAC verify bit (bit 21) must be on if you request verification.

The callable service name for AMODE(64) invocation is CSNETRV.

## Format

```
CALL CSNBTRV(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    transaction_key_identifier_length,
    transaction_key_identifier,
    transaction_info_length,
    transaction_info,
    validation_values_length,
    validation_values )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The valid values are 1 or 2.

## Transaction Validation

### rule\_array

Direction: Input

Type: Character String

Keywords that provides control information to the callable service. The keywords are left-justified in an 8-byte field and padded on the right with blanks. The keywords must be in contiguous storage. Specify one, two or three of the values in Table 229.

Table 229. Rule Array Keywords for Transaction Validation

Keyword	Meaning
<i>American Express card security codes (required)</i>	
CSC-3	3-digit card security code (CSC) located on the signature panel. <b>VERIFY</b> implied.
CSC-4	4-digit card security code (CSC) located on the signature panel. <b>VERIFY</b> implied.
CSC-5	5-digit card security code (CSC) located on the signature panel. <b>VERIFY</b> implied.
CSC-345	Generate 5-byte, 4-byte, 3-byte values when given an account number and an expiration date, <b>GENERATE</b> implied.
<i>Operation (optional)</i>	
VERIFY	Specifies verification of the value presented in the validation values variable.
GENERATE	Specifies generation of the value presented in the validation values variable.
<i>Card Security Code Algorithm (One, optional)</i>	
CSC-V1	Specifies use of CSC version 1.0 algorithm for generating or verifying the validation values.
CSC-V2	Specifies use of CSC version 2.0 algorithm for generating or verifying the validation values.

### transaction\_key\_identifier\_length

Direction: Input

Type: Integer

The length of the *transaction\_key\_identifier* parameter.

### transaction\_key\_identifier

Direction: Input

Type: String

The labelname or internal token of a MAC or MACVER class key. Key may be single or double length. When the CSC-V2 keyword is specified, the key must be a double-length key.

### transaction\_info\_length

Direction: Input

Type: Integer

The length of the *transaction\_info* parameter. For American Express CSC codes, this length must be 19 if the algorithm for CSC v1.0 is specified and it must be 22 if the algorithm for CSC v2.0 is specified.

### transaction\_info

Direction: Input

Type: String

Account information in character format. For American Express CSC-V1, this is a 19-byte field containing the concatenation of the 4-byte expiration data (in the format YYMM) and the 15-byte American Express account number. For CSC-V2, the string variable will contain the concatenation of the 4-byte expiration date in the format of (YYMM) , the 15-byte American Express account number and the 3-byte service code.

**validation\_values\_length**

Direction: Input/Output

Type: Integer

The length of the *validation\_values* parameter. Maximum value for this field is 64.

**validation\_values**

Direction: Input

Type: String

This variable contains American Express CSC values. The data is output for **GENERATE** and input for **VERIFY**.

*Table 230. Output description for validation values*

Operation	Element Description
<b>GENERATE</b> and <b>CSC-345</b>	5555544444333 where:  55555 = CSC 5 value 4444 = CSC 4 value 333 = CSC 3 value
<b>VERIFY</b> and <b>CSC-3</b>	333 = CSC 3 value
<b>VERIFY</b> and <b>CSC-4</b>	4444 = CSC 4 value
<b>VERIFY</b> and <b>CSC-5</b>	55555 = CSC 5 value

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

## Access Control Points

The following table shows the access control points in the ICSF role that control the function of this service.

*Table 231. Required access control points for Transaction Validation*

Operation keyword	Security code keyword	Access control point
GENERATE	CSC-345	Transaction Validation - Generate
VERIFY	CSC-3	Transaction Validation - Verify CSC-3
VERIFY	CSC-4	Transaction Validation - Verify CSC-4
VERIFY	CSC-5	Transaction Validation - Verify CSC-5

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 232. Transaction validation required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 800 IBM @server zSeries 900		Not Supported
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	Requires May 2004 or later version of Licensed Internal Code (LIC)
IBM @server zSeries 890	Crypto Express2 Coprocessor	CSC-V1 and CSC-V2 keywords not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	CSC-V1 and CSC-V2 keywords require May, 2012 or later version of LIC.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	CSC-V1 and CSC-V2 keywords require May, 2012 or later version of LIC for Crypto Express2. CSC-V1 and CSC-V2 keywords require June, 2012 or later version of LIC for Crypto Express3.
z196 z114	Crypto Express3 Coprocessor	CSC-V1 and CSC-V2 keywords require June, 2012 or later version of LIC.
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	CSC-V1 and CSC-V2 keywords require June, 2012 or later version of LIC for Crypto Express3. CSC-V1 and CSC-V2 keywords require September, 2012 or later version of LIC for Crypto Express4.

## VISA CVV Service Generate (CSNBCSG and CSNECSG)

Use the VISA CVV Service Generate callable service to generate a:

- VISA Card Verification Value (CVV)
- MasterCard Card Verification Code (CVC)
- Diner's Club Card Verification Value (CVV)

as defined for track 2.

This service generates a CVV that is based upon the information that the *PAN\_data*, the *expiration\_date*, and the *service\_code* parameters provide.

The service uses the Key-A and the Key-B keys to cryptographically process this information. Key-A and Key-B can be single-length DATA or MAC keys or a combined Key-A, Key-B double length DATA or MAC key. If the requested CVV is shorter than 5 characters, the CVV is padded on the right by space characters. The

CVV is returned in the 5-byte variable that the *CVV\_value* parameter identifies. When you verify a CVV, compare the result to the value that the *CVV\_value* supplies.

The callable service name for AMODE(64) invocation is CSNECSG.

## Format

```
CALL CSNBCSG(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    PAN_data,  
    expiration_date,  
    service_code,  
    CVV_key_A_Identifier,  
    CVV_key_B_Identifier,  
    CVV_value)
```

## Parameters

**return code**

Direction: Output	Type: Integer
-------------------	---------------

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

## reason code

Direction: Output	Type: Integer
-------------------	---------------

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit data length**

Direction: Input/Output	Type: Integer
-------------------------	---------------

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

**exit\_data**

Direction: Input/Output      Type: String

The data that is passed to the installation exit.

**rule\_array\_count**

Direction: Input	Type: Integer
------------------	---------------

## VISA CVV Service Generate

The number of keywords you are supplying in the *rule\_array* parameter. The parameter *rule\_array\_count* must be 0, 1, or 2.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields, and padded on the right with blanks. All keywords must be in contiguous storage.

*Table 233. CVV Generate Rule Array Keywords*

Keyword	Meaning
<i>PAN data length (optional)</i>	
PAN-13	Specifies that the length of the PAN data is 13 bytes. <b>PAN-13 is the default value.</b>
PAN-14	Specifies that the length of the PAN data is 14 bytes.
PAN-15	Specifies that the length of the PAN data is 15 bytes.
PAN-16	Specifies that the length of the PAN data is 16 bytes.
PAN-17	Specifies that the length of the PAN data is 17 bytes.
PAN-18	Specifies that the length of the PAN data is 18 bytes.
PAN-19	Specifies that the length of the PAN data is 19 bytes. Requires z990, z890, z9 EC or z9 BC with Jan. 2005 or higher version of Licensed Internal Code (LIC).
<i>CVV length (optional)</i>	
CVV-1	Specifies that the CVV is to be computed as one byte, followed by 4 blanks. <b>CVV-1 is the default value.</b>
CVV-2	Specifies that the CVV is to be computed as 2 bytes, followed by 3 blanks.
CVV-3	Specifies that the CVV is to be computed as 3 bytes, followed by 2 blanks.
CVV-4	Specifies that the CVV is to be computed as 4 bytes, followed by 1 blank.
CVV-5	Specifies that the CVV is to be computed as 5 bytes.

### **PAN\_data**

Direction: Input

Type: String

The *PAN\_data* parameter specifies an address that points to the place in application data storage that contains personal account number (PAN) information in character form. The PAN is the account number as defined for the track-2 magnetic-stripe standards.

- If the **PAN-13** keyword is specified in the rule array, 13 characters are processed.
- If the **PAN-14** keyword is specified in the rule array, 14 characters are processed.
- If the **PAN-15** keyword is specified in the rule array, 15 characters are processed.
- If the **PAN-16** keyword is specified in the rule array, 16 characters are processed.

- If the **PAN-17** keyword is specified in the rule array, 17 characters are processed.
- If the **PAN-18** keyword is specified in the rule array, 18 characters are processed.
- If the **PAN-19** keyword is specified in the rule array, 19 characters are processed.

Even if you specify the **PAN-13**, **PAN-14** or **PAN-15** keywords, the server might copy 16 bytes to a work area. Therefore ensure that the callable service can address 16 bytes of storage.

#### **expiration\_date**

Direction: Input

Type: String

The *expiration\_date* parameter specifies an address that points to the place in application data storage that contains the card expiration date in numeric character form in a 4-byte field. The application programmer must determine whether the CVV will be calculated with the date form of YYMM or MMY.

#### **service\_code**

Direction: Input

Type: String

The *service\_code* parameter specifies an address that points to the place in application data storage that contains the service code in numeric character form in a 3-byte field. The service code is the number that the track-2 magnetic-stripe standards define. The service code of '000' is supported.

#### **CVV\_key\_A\_Identifier**

Direction: Input/Output

Type: String

A 64-byte string that is the internal key token containing a single- or double-length DATA or MAC key or the label of a CKDS record containing a single- or double-length DATA or MAC key.

When this key is a double-length key, *CVV\_key\_B\_identifier* must be 64 byte of binary zero. When a double-length MAC key is used, the CV bits 0-3 must indicate a CVVKEY-A key (0010).

A single-length key contains the key-A key that encrypts information in the CVV process. The left half of a double-length key contains the key-A key that encrypts information in the CVV process and the right half contains the key-B key that decrypts information.

#### **CVV\_key\_B\_Identifier**

Direction: Input/Output

Type: String

A 64-byte string that is the internal key token containing a single-length DATA or MAC key or the label of a CKDS record containing a single-length DATA or MAC key. When *CVV\_key\_A\_identifier* a double-length key, this parameter must be 64 byte of binary zero. The key contains the key-B key that decrypts information in the CVV process.

#### **CVV\_value**

Direction: Output

Type: String



## VISA CVV Service Generate

The *CVV\_value* parameter specifies an address that points to the place in application data storage that will be used to store the computed 5-byte character output value.

### Restrictions

The CVV generate callable service is not supported on CCF systems with a CDMF-only configuration.

### Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

PAN-19 requires the z990, z890, z9 EC, z9 BC, z10 EC or z10 BC with Jan. 2005 or higher version of Licensed Internal Code (LIC).

### Access Control Point

The **VISA CVV Generate** access control point controls the function of this service.

### Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 234. VISA CVV service generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	The request is processed on the CCF if Key-A and Key-B are both DATA keys. MAC and MACVER keys are not supported.  PAN-14, PAN-15, PAN-17, PAN-18 and PAN-19 are not supported.
	PCI Cryptographic Coprocessor	The request is processed on a PCICC if Key-A or Key-B is a MAC key. MACVER keys are not supported.  PAN-14, PAN-15, PAN-17, PAN-18 and PAN-19 are not supported.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	MACVER keys are not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	MACVER keys are not supported.
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	MACVER keys are not supported.
	Crypto Express3 Coprocessor	

Table 234. VISA CVV service generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
z196 z114	Crypto Express3 Coprocessor	MACVER keys are not supported.  Combined CVV keys require the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	MACVER keys are not supported.

## VISA CVV Service Verify (CSNBCSV and CSNECSV)

Use the VISA CVV Service Verify callable service to verify a:

- VISA Card Verification Value (CVV)
- MasterCard Card Verification Code (CVC)
- Diner's Club Card Verification Value (CVV)

as defined for track 2.

This service verifies a CVV that is based upon the information that the *PAN\_data*, the *expiration\_date*, and the *service\_code* parameters provide.

The service uses the Key-A and the Key-B keys to cryptographically process this information. If the requested CVV is shorter than 5 characters, the CVV is padded on the right by space characters. On an IBM zSeries 900 or lower, the user must pad out the *CVV\_value* parameter with blanks if the supplied CVV is less than 5 characters. The generated CVV is then compared to the value that the *CVV\_value* supplies for verification.

The callable service name for AMODE(64) invocation is CSNECSV.

## Format

```
CALL CSNBCSV(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    PAN_data,
    expiration_date,
    service_code,
    CVV_key_A_Identifier,
    CVV_key_B_Identifier,
    CVV_value)
```

## Parameters

**return\_code**

Direction: Output

Type: Integer

## VISA CVV Service Verify

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The parameter *rule\_array\_count* must be 0, 1, or 2.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields, and padded on the right with blanks. All keywords must be in contiguous storage.

Table 235. CVV Verify Rule Array Keywords

Keyword	Meaning
<i>PAN data length (optional)</i>	
PAN-13	Specifies that the length of the PAN data is 13 bytes. <b>PAN-13 is the default value.</b>
PAN-14	Specifies that the length of the PAN data is 14 bytes.
PAN-15	Specifies that the length of the PAN data is 15 bytes.
PAN-16	Specifies that the length of the PAN data is 16 bytes.
PAN-17	Specifies that the length of the PAN data is 17 bytes.
PAN-18	Specifies that the length of the PAN data is 18 bytes.
PAN-19	Specifies that the length of the PAN data is 19 bytes. Requires z990, z890, z9 EC or z9 BC with Jan. 2005 or higher version of Licensed Internal Code (LIC).

Table 235. CVV Verify Rule Array Keywords (continued)

Keyword	Meaning
<i>CVV length (optional)</i>	
CVV-1	Specifies that the CVV is to be computed as one byte, followed by 4 blanks. <b>CVV-1 is the default value.</b>
CVV-2	Specifies that the CVV is to be computed as 2 bytes, followed by 3 blanks.
CVV-3	Specifies that the CVV is to be computed as 3 bytes, followed by 2 blanks.
CVV-4	Specifies that the CVV is to be computed as 4 bytes, followed by 1 blank.
CVV-5	Specifies that the CVV is to be computed as 5 bytes.

**PAN\_data**

Direction: Input

Type: String

The *PAN\_data* parameter specifies an address that points to the place in application data storage that contains personal account number (PAN) information in character form. The PAN is the account number as defined for the track-2 magnetic-stripe standards.

- If the **PAN-13** keyword is specified in the rule array, 13 characters are processed.
- If the **PAN-14** keyword is specified in the rule array, 14 characters are processed.
- If the **PAN-15** keyword is specified in the rule array, 15 characters are processed.
- If the **PAN-16** keyword is specified in the rule array, 16 characters are processed.
- If the **PAN-17** keyword is specified in the rule array, 17 characters are processed.
- If the **PAN-18** keyword is specified in the rule array, 18 characters are processed.
- If the **PAN-19** keyword is specified in the rule array, 19 characters are processed.

Even if you specify the **PAN-13**, **PAN-14** or **PAN-15** keywords, the server might copy 16 bytes to a work area. Therefore ensure that the callable service can address 16 bytes of storage.

**expiration\_date**

Direction: Input

Type: String

The *expiration\_date* parameter specifies an address that points to the place in application data storage that contains the card expiration date in numeric character form in a 4-byte field. The application programmer must determine whether the CVV will be calculated with the date form of YYMM or MMY.

**service\_code**

Direction: Input

Type: String

## VISA CVV Service Verify

The *service\_code* parameter specifies an address that points to the place in application data storage that contains the service code in numeric character form in a 3-byte field. The service code is the number that the track-2 magnetic-stripe standards define. The service code of '000' is supported.

### CVV\_key\_A\_Identifier

Direction: Input/Output

Type: String

A 64-byte string that is the internal key token containing a single- or double-length DATA or MAC key or the label of a CKDS record containing a single- or double-length DATA or MAC key.

When this key is a double-length key, *CVV\_key\_B\_identifier* must be 64 byte of binary zero. When a double-length MAC key is used, the CV bits 0-3 must indicate a CVVKEY-A key (0010).

A single-length key contains the key-A key that encrypts information in the CVV process. The left half of a double-length key contains the key-A key that encrypts information in the CVV process and the right half contains the key-B key that decrypts information.

### CVV\_key\_B\_Identifier

Direction: Input/Output

Type: String

A 64-byte string that is the internal key token containing a single-length DATA or MAC key or the label of a CKDS record containing a single-length DATA or MAC key. When *CVV\_key\_A\_identifier* a double-length key, this parameter must be 64 byte of binary zero. The key contains the key-B key that decrypts information in the CVV process.

### CVV\_value

Direction: Input

Type: String

The *CVV\_value* parameter specifies an address that contains the CVV value which will be compared to the computed CVV value. This is a 5-byte field.

On an IBM zSeries 900, the user must pad out the *CVV\_value* parameter with blanks if the supplied CVV is less than 5 characters.

## Restrictions

The CVV verify callable service is not supported on CCF systems with a CDMF-only configuration..

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

PAN-19 requires the z990, z890, z9 EC, z9 BC, z10 EC or z10 BC with Jan. 2005 or higher version of Licensed Internal Code (LIC).

## Access Control Points

The **VISA CVV Verify** access control point controls the function of this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 236. VISA CVV service verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	The request is processed on the CCF if Key-A and Key-B are both DATA keys. MAC and MACVER keys are not supported.  PAN-14, PAN-15, PAN-17, PAN-18 and PAN-19 are not supported.
	PCI Cryptographic Coprocessor	The request is processed on a PCICC if Key-A or Key-B is a MAC or MACVER key.  PAN-14, PAN-15, PAN-17, PAN-18 and PAN-19 are not supported.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor  Crypto Express3 Coprocessor	
z196 z114	Crypto Express3 Coprocessor	Combined CVV keys require the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12	Crypto Express3 Coprocessor	
	Crypto Express4 Coprocessor	



---

## Chapter 9. Using Digital Signatures

This topic describes the PKA callable services that support using digital signatures to authenticate messages.

- “Digital Signature Generate (CSNDDSG and CSNFDSG)”
- “Digital Signature Verify (CSNDDSV and CSNFDSV)” on page 566

---

### Digital Signature Generate (CSNDDSG and CSNFDSG)

Use the digital signature generate callable service to generate a digital signature using a PKA private key, or, for some limited functions, a secure PKCS #11 private key. The digital signature generate callable service may use an RSA, DSS, or ECC private key, depending on the algorithm you are using. DSS is not supported on the PCIXCC or Crypto Express coprocessor.

Private keys must be valid for signature usage. This service supports these methods:

- ANSI X9.30 (DSS)
- ANSI X9.30 (ECDSA)
- ANSI X9.31 (RSA)
- ISO 9796-1 (RSA)
- RSA DSI PKCS 1.0 and 1.1 (RSA)
- Padding on the left with zeros (RSA)

**Notes:**

1. The maximum signature length is 512 bytes (4096 bits).
2. For secure PKCS #11 private keys, the only supported services are ANSI X9.30 (ECDSA) and RSA PKCS 1.1.

The input text should have been previously hashed using either the one-way hash generate callable service or the MDC generation callable service. If the signature formatting algorithm specifies ANSI X9.31, you must specify the hash algorithm used to hash the text (SHA-1 or RPMD-160). See “Formatting Hashes and Keys in Public-Key Cryptography” on page 952.

For PKA keys, if the *private\_key\_identifier* specifies an RSA private key, you select the method of formatting the text through the *rule\_array* parameter. If the *private\_key\_identifier* specifies a DSS private key, the DSS signature generated is according to ANSI X9.30. For DSS, the signature is generated on a 20-byte hash created from SHA-1 algorithm. If the *private\_key\_identifier* specifies an ECC private key, the ECC signature is generated according to ANSI X9.30.

For secure PKCS #11 keys, if the *private\_key\_identifier* specifies an RSA private key, you must select the PKCS-1.1 method of formatting the text through the *rule\_array* parameter. If the *private\_key\_identifier* specifies an ECC private key, the ECC signature is generated according to ANSI X9.30.

**Note:** For RSA PKCS 1.0 or 1.1, the message digest and the message-digest algorithm identifier are combined into an ASN.1 value of type DigestInfo, which is BER-encoded to give an octet string D (see Table 237). D is the text string supplied in the *hash* variable.



The callable service name for AMODE(64) invocation is CSNFD SG.

### Format

```
CALL CSNDDSG(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    private_key_identifier_length,  
    private_key_identifier,  
    hash_length,  
    hash,  
    signature_field_length,  
    signature_bit_length,  
    signature_field)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

#### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The value may be 0 1, 2, or 3.

#### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. One keyword specifies the method for calculating the digital signature. Another keyword specifies formatting of the hash value for RSA digital signature generation. A third keyword specifies the hash method used to prepare the hash value for RSA digital signature generation. Table 237 lists the keywords. Each keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

Table 237. Keywords for Digital Signature Generate Control Information

Keyword	Meaning
<i>Digital Signature Formatting Method (optional, valid for RSA digital signature generation only)</i>	
ISO-9796	Calculate the digital signature on the <i>hash</i> according to ISO-9796-1. Any hash method is allowed. This is the default.
PKCS-1.0	Calculate the digital signature on the BER-encoded ASN.1 value of the type DigestInfo containing the hash according to the RSA Data Security, Inc. Public Key Cryptography Standards #1 block type 00. The text must have been hashed prior to inputting to this service.
PKCS-1.1	Calculate the digital signature on the BER-encoded ASN.1 value of the type DigestInfo containing the hash according to the RSA Data Security, Inc. Public Key Cryptography Standards #1 block type 01. The text must have been hashed prior to inputting to this service.
ZERO-PAD	Format the hash by padding it on the left with binary zeros to the length of the RSA key modulus. Any supported hash function is allowed.
X9.31	Format according to the ANSI X9.31 standard. The input text must have been previously hashed with one of these hash algorithms:
<i>Hash Method Specification: Required with X9.31</i>	
RPMD-160	Hash the input text using the RIPEMD-160 hash method.
SHA-1	Hash the input text using the SHA-1 hash method.
<i>Signature algorithm (optional, supported on the CEX3C or CEX4C coprocessor)</i>	
RSA	RSA or DSS processing is to occur.
ECDSA	The elliptic curve digital signature algorithm is to be used. When specified, this is the only keyword permitted in the Rule Array.

**private\_key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *private\_key\_identifier* field. The maximum size is 3500 bytes.

**private\_key\_identifier**

## Digital Signature Generate

Direction: Input

Type: String

For PKA keys, this is an internal token or label of an RSA or DSS private key or Retained key. If the signature format is X9.31, the modulus of the RSA key must have a length of at least 1024 bits. If the signature algorithm is ECDSA, this must be a token or label of an ECC private key.

For secure PKCS #11 keys, this is the 44-byte handle of the private key, prefixed with an EBCDIC equal sign character ('=' or x'7E'), and padded on the right with spaces for a total length of 64 bytes.

### hash\_length

Direction: Input

Type: Integer

The length of the *hash* parameter in bytes. It must be the exact length of the text to sign. The maximum size is 512 bytes. If you specify ZERO-PAD in the *rule\_array* parameter, the length is restricted to 36 bytes unless the RSA key is a signature only key, then the maximum length is 512 bytes.

On the IBM @server zSeries 990 and subsequent releases, the hash length limit is controlled by a new access control point. Only RSA key management keys are affected by this access control point. The limit for RSA signature use only keys is 512 bytes. This new access control point is always disabled in the Default role. You must have a TKE workstation to enable it.

### hash

Direction: Input

Type: String

The application-supplied text on which to generate the signature. The input text must have been previously hashed, and for PKCS formatting, it must be BER-encoded as previously described. For X9.31, the hash algorithms must have been either SHA-1 or RIPEMD-160. See the *rule\_array* parameter for more information.

### signature\_field\_length

Direction: Input/Output

Type: Integer

The length in bytes of the *signature\_field* to contain the generated digital signature. Upon return, this field contains the actual length of the generated signature. The maximum size is 512 bytes.

**Note:** For RSA, this must be at least the RSA modulus size (rounded up to a multiple of 32 bytes for the X9.31 signature format, or one byte for all other signature formats). For DSS, this must be at least 40 bytes.

For RSA and DSS, this field is updated with the minimum byte length of the digital signature.

For ECDSA, signature algorithm R concatenated with S is the digital signature. The maximum output value will be 1042 bits (131 bytes). The size of the signature is determined by the size of P. Both R and S will have size P. For prime curves, the maximum is  $2 * 521$  bits. For brain pool curves, the maximum size is  $2 * 512$  bits.

### signature\_bit\_length

Direction: Output

Type: Integer

The bit length of the digital signature generated. For ISO-9796 this is 1 less than the modulus length. For other RSA processing methods, this is the modulus length. For DSS, this is 320.

**signature\_field**

Direction: Output

Type: String

The digital signature generated is returned in this field. The digital signature is in the low-order bits (right-justified) of a string whose length is the minimum number of bytes that can contain the digital signature. This string is left-justified within the *signature\_field*. Any unused bytes to the right are undefined.

**Restrictions**

Although ISO-9796 does not require the input hash to be an integral number of bytes in length, this service requires you to specify the *hash\_length* in bytes.

X9.31 requires the RSA token to have a modulus bit length of at least 1024 bits and the length must also be a multiple of 256 bits (or 32 bytes).

The length of the *hash* parameter in bytes. It must be the exact length of the text to sign. The maximum size is 512 bytes. If you specify ZERO-PAD in the *rule\_array* parameter, the length is restricted to 36 bytes unless the RSA key is a signature only key, then the maximum length is 512 bytes.

For PKA keys, on the IBM @server zSeries 990 and subsequent releases, the hash length limit is controlled by a new access control point. If ON (enabled), the maximum hash length limit for ZERO-PAD is the modulus length of the PKA private key. If OFF (disabled), the maximum hash length limit for ZERO-PAD is 36 bytes. Only RSA key management keys are affected by this access control point. The limit for RSA signature use only keys is 512 bytes. This new access control point is always disabled in the Default role. You must have a TKE workstation to enable it.

**Authorization**

To use this service with a secure PKCS #11 private key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access) to the containing PKCS #11 token.

To use this service with a secure PKCS #11 private key that is a private object, the caller must have USER (READ) authority (user access) to the containing PKCS #11 token.

See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the SO and User PKCS #11 roles.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS, PKDS, or TKDS.

## Digital Signature Generate

For PKA private keys, the **Digital Signature Generate** access control point controls the function of this service.

| For secure PKCS #11 private keys, the Sign with private keys access control point  
| controls the function of this service. For more information on the access control  
| points of the Enterprise PKCS #11 coprocessor, see PKCS #11 Access Control Points  
| *inz/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

The length of the hash for ZERO-PAD is restricted to 36 bytes. If the **DSG ZERO-PAD unrestricted hash length** access control point is enabled in the ICSF role, the length of the hash is not restricted. This access control is disabled by default.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 238. Digital signature generate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	<p>ECC not supported.</p> <p>The request is processed on the CCF when:</p> <ul style="list-style-type: none"> <li>the modulus bit length of the RSA key is less than 512 bits</li> <li>the key specified is a DSS key</li> <li>the key specified is a X'02' private modulus-exponent RSA key</li> <li>the key specified is a X'06' private modulus-exponent RSA key and the key use bits indicate signature only</li> <li>the key specified is a X'06' private modulus-exponent RSA key and the key use bits indicate key-management use and the SMK is equal to the KMMK</li> </ul> <p>RSA keys with moduli greater than 1024-bit length are not supported.</p>
	PCI Cryptographic Coprocessor	<p>ECC not supported.</p> <p>The request is processed on the PCICC when</p> <ul style="list-style-type: none"> <li>the key specified is a X'08' CRT RSA key</li> <li>the key specified is a retained key. The request will be routed to the specific coprocessor of the retained key.</li> <li>the key specified is a X'06' private modulus-exponent RSA key and the key use bits indicate signature only</li> <li>the key specified is a X'06' private modulus-exponent RSA key and the key use bits indicate key-management use and the SMK is equal to the KMMK</li> <li>the key specified is a X'06' private modulus-exponent RSA key and the key use bits indicate key-management use and the SMK is not equal to the KMMK</li> </ul> <p>RSA keys with moduli greater than 2048-bit length are not supported.</p>

Table 238. Digital signature generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 990  IBM @server zSeries 890	PCI X Cryptographic Coprocessor  Crypto Express2 Coprocessor	ECC not supported.  DSS tokens are not supported.  ZERO-PAD hash length is controlled by an access control point. When enabled, the hash length limit is 36 bytes. When disabled, the hash length limit is the modulus byte length of the RSA key. This access control point is always disabled and can only be enabled with TKE V4.0 or higher.  RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z9 EC  IBM System z9 BC	Crypto Express2 Coprocessor	ECC not supported.  DSS tokens are not supported.  ZERO-PAD hash length is controlled by an access control point. When enabled, the hash length limit is 36 bytes. When disabled, the hash length limit is the modulus byte length of the RSA key. This access control point is always disabled and can only be enabled with TKE V4.0 or higher.  RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).

Table 238. Digital signature generate required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	ECC not supported.  DSS tokens are not supported.  ZERO-PAD hash length is controlled by an access control point. When enabled, the hash length limit is 36 bytes. When disabled, the hash length limit is the modulus byte length of the RSA key. This access control point is always disabled and can only be enabled with TKE V4.0 or higher.  RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express3 Coprocessor	ZERO-PAD hash length is controlled by an access control point. When enabled, the hash length limit is 36 bytes. When disabled, the hash length limit is the modulus byte length of the RSA key. This access control point is always disabled and can only be enabled with TKE V4.0 or higher.  DSS tokens are not supported.  ECC support requires the Sep. 2010 licensed internal code (LIC).
z196 z114	Crypto Express3 Coprocessor	ZERO-PAD hash length is controlled by an access control point. When enabled, the hash length limit is 36 bytes. When disabled, the hash length limit is the modulus byte length of the RSA key. This access control point is always disabled and can only be enabled with TKE V4.0 or higher.  DSS tokens are not supported.
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 CCA Coprocessor	ZERO-PAD hash length is controlled by an access control point. When enabled, the hash length limit is 36 bytes. When disabled, the hash length limit is the modulus byte length of the RSA key. This access control point is always disabled and can only be enabled with TKE V4.0 or higher.  DSS tokens are not supported.
	Crypto Express4 Enterprise PKCS #11 coprocessor	Required to use a secure PKCS #11 private key



### Digital Signature Verify (CSNDDSV and CSNFDSV)

Use the digital signature verify callable service to verify a digital signature using a PKA public key.

- The digital signature verify callable service can use the RSA, DSS, or ECC public key, depending on the digital signature algorithm used to generate the signature. DSS is supported only on the IBM @server zSeries 900.
- The digital signature verify callable service can also use the public keys that are contained in trusted blocks regardless of whether the block also contains rules to govern its use when generating or exporting keys with the RKX service. If the TPK-ONLY keyword is used in the **rule\_array**, an error will occur if the **PKA\_public\_key\_identifier** does not contain a trusted block.

This service supports these methods:

- ANSI X9.30 (DSS and ECC)
- ANSI X9.31 (RSA)
- ISO 9796 (RSA)
- RSA DSI PKCS 1.0 and 1.1 (RSA)
- Padding on the left with zeros (RSA)

Input text should have been previously hashed. You can use either the one-way hash generate callable service or the MDC generation callable service. See also “Formatting Hashes and Keys in Public-Key Cryptography” on page 952.

**Note:** The maximum signature length is 512 bytes.

The callable service name for AMODE(64) invocation is CSNFDSV.

### Format

```
CALL CSNDDSV(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    PKA_public_key_identifier_length,  
    PKA_public_key_identifier,  
    hash_length,  
    hash,  
    signature_field_length,  
    signature_field)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

#### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The value must be 0, 1, or 2.

#### **rule\_array**

Direction: Input

Type: String

Contains an array of keywords that provide control information to the callable service. One keyword specifies the method to use to verify the RSA digital signature. Another keyword specifies the input token is a Trusted Block. A third keyword specifies the algorithm used to validate the signature. Table 239 lists the keywords. Each keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

Table 239. Keywords for Digital Signature Verify Control Information

Keyword	Meaning
<b>Digital Signature Formatting Method (optional, RSA only)</b>	
X9.31	Format according to the ANSI X9.31 standard.
ISO-9796	Calculate the digital signature on the hash according to ISO 9796-1. Any hash method is allowed. This is the default.
PKCS-1.0	Calculate the digital signature on the BER-encoded ASN.1 value of the type DigestInfo containing the hash according to the RSA Data Security, Inc., <i>Public Key Cryptography Standards</i> #1 block type 00 and compare to the digital signature. The text must have been hashed prior to inputting to this service.
PKCS-1.1	Calculate the digital signature on the BER-encoded ASN.1 value of the type DigestInfo containing the hash according to the RSA Data Security, Inc., <i>Public Key Cryptography Standards</i> #1 block type 01 and compare to the digital signature. The text must have been hashed prior to inputting to this service.

Table 239. Keywords for Digital Signature Verify Control Information (continued)

Keyword	Meaning
ZERO-PAD	Format the hash by padding it on the left with binary zeros to the length of the PKA key modulus. Any supported hash function is allowed.
<b>PKA public key token type (one, optional)</b>	
TPK-ONLY	The PKA_public_key_identifier must be a trusted block that contains, at a minimum, two sections: <ol style="list-style-type: none"> <li>1. Trusted Block Information section 0x14 which is required for all trusted blocks and</li> <li>2. Trusted Public Key section 0x11 which contains the trusted public key and usage rules that indicate whether or not the trusted public key can be used in digital signature operations.</li> </ol>
<b>Signature Algorithm (optional, supported on the CEX3C or CEX4C coprocessor)</b>	
RSA	RSA or DSS processing is to occur. This is the default value.
ECDSA	The elliptic curve digital signature algorithm is to be used. When specified, this is the only keyword permitted in the Rule Array.

### PKA\_public\_key\_identifier\_length

Direction: Input

Type: Integer

The length of the *PKA\_public\_key\_identifier* parameter containing the public key token or label. The maximum size is 3500 bytes.

### PKA\_public\_key\_identifier

Direction: Input

Type: String

A token or label of the RSA or DSS public key or internal trusted block. If this parameter contains a token or the label of an Internal Trusted Block, the *rule\_array* parameter must specify TPK-ONLY. If the signature algorithm is ECDSA, this must be a token label of an ECC public key.

### hash\_length

Direction: Input

Type: Integer

The length of the *hash* parameter in bytes. It must be the exact length of the text that was signed. The maximum size is 512 bytes.

### hash

Direction: Input

Type: String

The application-supplied text on which the supplied signature was generated. The text must have been previously hashed and, for PKCS formatting, BER-encoded as previously described.

### signature\_field\_length

Direction: Input

Type: Integer

The length in bytes of the *signature\_field* parameter. The maximum size is 512 bytes.

**signature\_field**

Direction: Input

Type: String

This field contains the digital signature to verify. The digital signature is in the low-order bits (right-justified) of a string whose length is the minimum number of bytes that can contain the digital signature. This string is left-justified within the *signature\_field*.

**Restrictions**

The ability to recover a message from a signature (which ISO-9796 allows but does not require) is **not** supported.

The exponent of the RSA public key must be odd.

Although ISO-9796 does not require the input hash to be an integral number of bytes in length, this service requires you to specify the *hash\_length* in bytes.

X9.31 requires the RSA token to have a modulus bit length of at least 1024 bits and the length must also be a multiple of 256 bits (or 32 bytes).

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

For DSS if  $r=0$  or  $s=0$  then verification always fails. The DSS digital signature is of the form  $r || s$ , each 20 bytes.

**Access Control Point**

The **Digital Signature Verify** access control point controls the function of this service.

**Required Hardware**

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 240. Digital signature verify required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	<p>ECC not supported.</p> <p>Trusted key block not supported.</p> <p>TPK-ONLY keyword not supported.</p> <p>RSA keys with moduli greater than 1024-bit length are not supported.</p>

Table 240. Digital signature verify required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	ECC not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	DSS tokens are not supported.
	PCI Cryptographic Accelerator	Trusted key block not supported.
		TPK-ONLY keyword not supported.
		RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z9 EC	Crypto Express2 Coprocessor	ECC not supported.
IBM System z9 BC	Crypto Express2 Accelerator	DSS tokens are not supported.
		RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC	Crypto Express2 Coprocessor	ECC not supported.
IBM System z10 BC	Crypto Express2 Accelerator	DSS tokens are not supported.
		RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express3 Coprocessor	ECC not supported.
	Crypto Express3 Accelerator	DSS tokens not supported.
z196	Crypto Express3 Coprocessor	DSS tokens are not supported.
z114	Crypto Express3 Accelerator	RSA clear key support with moduli within the range 2048-bit and 4096-bit requires the Sep. 2011 or later licensed internal code (LIC).
IBM zEnterprise EC12	Crypto Express3 Coprocessor	DSS tokens are not supported.
	Crypto Express3 Accelerator	
	Crypto Express4 Coprocessor	
	Crypto Express4 Accelerator	

---

## Chapter 10. Managing PKA Cryptographic Keys

This topic describes the callable services that generate and manage PKA keys.

- “PKA Key Generate (CSNDPKG and CSNFPKG)”
- “PKA Key Import (CSNDPKI and CSNFPKI)” on page 578
- “PKA Key Token Build (CSNDPKB and CSNFPKB)” on page 583
- “PKA Key Token Change (CSNDKTC and CSNFKTC)” on page 596
- “PKA Key Translate (CSNDPKT and CSNFPKT)” on page 599
- “PKA Public Key Extract (CSNDPKX and CSNFPKX)” on page 604
- “Retained Key Delete (CSNDRKD and CSNFRKD)” on page 606
- “Retained Key List (CSNDRKL and CSNFRKL)” on page 609

---

### PKA Key Generate (CSNDPKG and CSNFPKG)

Use the PKA key generate callable service to generate these PKA keys:

- PKA internal tokens for use with the DSS algorithm in the digital signature services
- RSA keys for use on the Cryptographic Coprocessor Feature, PCI Cryptographic Coprocessor, PCI X Cryptographic Coprocessor, or CCA Crypto Express coprocessor.
- ECC keys for use on a CEX3C or CEX4C.

Input to the PKA key generate callable service is either a skeleton key token that has been built by the PKA key token build service or a valid internal RSA token. PKG will generate a key with the same modulus length and the same exponent. In the case of a valid internal ECC token, PKG will generate a key based on the curve type and size. Internal tokens with a 'X'09' section are not supported.

DSS key generation requires this information in the input skeleton token:

- Size of modulus p in bits
- Prime modulus p
- Prime divisor q
- Public generator g
- Optionally, the private key name

DSS standards define restrictions on p, q, and g. (Refer to the Federal Information Processing Standard (FIPS) Publication 186 for DSS standards.) This callable service does not verify all of these restrictions. If you do not follow these restrictions, the keys you generate may not be valid DSS keys. The PKA Key Token Build service or an existing internal or external PKA DSS token can generate the input skeleton token, but all of the preceding must be provided. You can extract the DSS public key token from the internal private key token by calling the PKA public key extract callable service.

**Note:** DSS keys are not supported on a PCIXCC or CCA Crypto Express coprocessor.

RSA key generation requires this information in the input skeleton token:

- Size of the modulus in bits. The modulus for modulus-exponent form keys is between 512 and 1024. The CRT modulus is between 512 and 4096. The modulus for the variable-length-modulus-exponent form is between 512 and 4096.

## PKA Key Generate

RSA key generation has these restrictions: For modulus-exponent, there are restrictions on modulus, public exponent, and private exponent. For CRT, there are restrictions on dp, dq, U, and public exponent. See the Key value structure in “PKA Key Token Build (CSNDPKB and CSNFPKB)” on page 583 for a summary of restrictions.

**Note:** The Transaction Security System PKA96 PKA key generate verb supports RSA key generation only; it does not support DSS key generation.

ECC key generation requires this information in the skeleton token:

- The key type: ECC
- The type of curve: Prime or Brainpool
- The size of P in bits: 192, 224, 256, 384 or 521 for Prime curves and 160, 192, 224, 256, 320, 384, or 512 for Brainpool curves
- Key usage information
- Optionally, application associated data

The generated ECC private key will be returned in one of the following forms:

- Clear key
- Encrypted key enciphered under the ECC master key
- Encrypted key enciphered by an AES transport key

The callable service name for AMODE(64) invocation is CSNFPKG.

## Format

```
CALL CSNDPKG(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    regeneration_data_length,  
    regeneration_data,  
    skeleton_key_identifier_length,  
    skeleton_key_identifier,  
    transport_key_identifier,  
    generated_key_token_length,  
    generated_key_token)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned

to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

## exit\_data\_length

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

## exit\_data

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

## rule\_array\_count

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. Value may be 1 or 2.

## rule\_array

Direction: Input

Type: String

A keyword that provides control information to the callable service. See Table 241 for a list. A keyword is left-justified in an 8-byte field and padded on the right with blanks.

Table 241. Keywords for PKA Key Generate Rule Array

Keyword	Meaning
<i>Private Key Encryption (required)</i>	
CLEAR	Return the private key in clear text. The private key in clear text is an external token. Only valid for RSA and ECC keys.
MASTER	Encipher the private key under the master key. The keyword is not supported if a skeleton token with a X'09' section is provided.
RETAIN	Retain the private key within a cryptographic coprocessor for additional security. This is only valid for RSA signature keys. Because of this, the RETAIN keyword is not supported for: <ul style="list-style-type: none"> <li>a skeleton token with a X'09', X'30', or X'31' section provided.</li> <li>an ECC token.</li> </ul>
XPORT	Encipher the private key under the <i>transport_key_identifier</i> . This keyword is valid only for RSA and ECC keys.
<i>Options (optional)</i>	



## PKA Key Generate

Table 241. Keywords for PKA Key Generate Rule Array (continued)

Keyword	Meaning
CLONE	Mark a generated and retained private key as usable in cryptographic engine cloning process. This keyword is supported only if RETAIN is also specified. Only valid for RSA keys. The keyword is not supported for: <ul style="list-style-type: none"><li>a skeleton token with a X'09' section is provided.</li><li>an ECC token.</li></ul>
<i>Processing Controls (Optional when regeneration_data_length is non-zero)</i>	
ITER-38	When <i>regeneration_data</i> is specified, this keyword will cause the service to generate key values that are FIPS and ANSI X9.31 compliant.
<i>Transport Key Type (one optional)</i>	
OKEK-DES	The transport key identifier identifies a DES KEK token. This is the default value.
OKEK-AES	The transport key identifier identifies an AES KEK token.

### **regeneration\_data\_length**

Direction: Input

Type: Integer

The value must be 0 for DSS and ECC tokens. For RSA tokens, the *regeneration\_data\_length* can be non-zero. If it is non-zero, it must be between 8 and 512 bytes inclusive.

### **regeneration\_data**

Direction: Input

Type: String

This field points to a string variable containing a string used as the basis for creating a particular public-private key pair in a repeatable manner.

### **skeleton\_key\_identifier\_length**

Direction: Input

Type: Integer

The length of the *skeleton\_key\_identifier* parameter in bytes. The maximum allowed value is 3500 bytes.

### **skeleton\_key\_identifier**

Direction: Input

Type: String

The application-supplied skeleton key token generated by PKA key token build or label of the token that contains the required network quantities for DSS key generation, the required curve type and bit length for ECC key generation, or the required modulus length and public exponent for RSA key generation. If RETAIN was specified and the *skeleton\_key\_identifier* is a label, the label must match the private key name of the key.

For DSS and RSA keys, the *skeleton\_key\_identifier* parameter must contain a token which specifies a modulus length in the range 512 – 4096 bits.

### **transport\_key\_identifier**

Direction: Input

Type: String

A variable-length field containing an AES or DES key identifier used to encrypt the generated key. For RSA keys, this may be an AES or DES transport key. When there is an RSA private key section X'30' or X'31' in the skeleton token, an AES transport key must be specified. The key-usage field in the AES key must allow the key to wrap an RSA key. For all other RSA private key sections, a DES transport key must be specified. For ECC keys, this must be an AES transport key which is able to wrap an ECC key.

If the XPORT Rule is not specified or the key being generated is a DSS key, this parameter must be 64 bytes of binary zeros.

For XPORT rule, this is an IMPORTER or EXPORTER key or the label of an IMPORTER or EXPORTER key. If you specify a label, it must resolve uniquely to either an IMPORTER or EXPORTER key. This parameter is a:

- 64-byte label of a CKDS record that contains the transport key.
- 64-byte DES internal key token containing the transport key.
- a variable-length AES internal key token containing the transport key.

## generated\_key\_token\_length

Direction: Input/Output

Type: Integer

The length of the generated key token. The field is checked to ensure it is at least equal to the token being returned. The maximum size is 3500 bytes. On output, this field is updated with the actual token length.

## generated\_key\_token

Direction: Input/Output

Type: String

The internal token or label of the generated DSS, ECC, or RSA key. The label can be that of a retained key for most RSA key tokens.

Checks are made to ensure that:

- An ECC Token in the PKDS will only be overlayed if an ECC token is specified in the *skeleton\_key\_identifier*
- A retained key is not overlayed in PKDS. If the label is that of a retained key, the private name in the token must match the label name. If a label is specified in the *generated\_key\_token* field, the *generated\_key\_token\_length* returned to the application will be the same as the input length. If RETAIN was specified, but the *generated\_key\_token* was not specified as a label, the generated key length returned to the application will be zero (the key was retained in the cryptographic coprocessor). If the record already exists in the PKDS with the same label as the one specified as the *generated\_key\_token*, the record will be overwritten with the newly generated key token (unless the PKDS record is an existing retained private key, in which case it cannot be overwritten). If there is no existing PKDS record with this label in the case of generating a retained key, a record will be created. For generation of a non-retained key, if a label is specified in the *generated\_key\_token* field, a record must already exist in the PKDS with this same label or the service will fail.

## PKA Key Generate

### Restrictions

2048-bit RSA keys may have a public exponent in the range of 1-256 bytes. 2049- to 4096-bit RSA key public exponents are restricted to the values 3 and 65537.

### Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

### Access Control Points

The **PKA Key Generate** access control point controls the function of this service. Additional access control points control the use of rule array keys.

Table 242. Required access control points for PKA Key Generate rule array keys

Key algorithm	Rule array keyword	Access control point
RSA	CLEAR	PKA Key Generate – Clear RSA keys
ECC	CLEAR	PKA Key Generate – Clear ECC keys
RSA	CLONE	PKA Key Generate - Clone

To generate keys based on the value supplied in the *regeneration\_data* variable, you must enable at least one of these access control points:

- When not using the RETAIN keyword, **PKA Key Generate - Permit Regeneration Data**
- When using the RETAIN keyword, **PKA Key Generate - Permit Regeneration Data Retain**

For ECC keys, when an transport key is specified, the **Prohibit weak wrapping - Transport keys** access control point can be enabled in the active role to prevent stronger keys from being wrapped by weaker keys.

### Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 243. PKA key generate required hardware

Server	Required Cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	ECC not supported.  The service examines the skeleton token and routes the generation request to the appropriate cryptographic processor. If the skeleton is a DSS key token, processing takes place on the Cryptographic Coprocessor Feature.
	PCI Cryptographic Coprocessor	ECC not supported.  The service examines the skeleton token and routes the generation request to the appropriate cryptographic processor. If the skeleton is a DSS key token, processing takes place on the Cryptographic Coprocessor Feature.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	ECC not supported.  DSS tokens are not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ECC not supported.  DSS tokens are not supported.  RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	ECC not supported.  DSS tokens are not supported.  RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express3 Coprocessor	DSS tokens are not supported.  ECC support requires the Sep. 2010 licensed internal code (LIC).
z196 z114	Crypto Express3 Coprocessor	DSS tokens are not supported.  ECC Clear Key and Internal token support requires the Sep. 2010 licensed internal code (LIC).  ECC External token and Diffie-Hellman support requires the Sep. 2011 or later licensed internal code (LIC).  Wrapping of RSA keys with the ECC master key or AES transport keys is not supported.

## PKA Key Generate

Table 243. PKA key generate required hardware (continued)

Server	Required Cryptographic hardware	Restrictions
IBM zEnterprise EC12	Crypto Express3 Coprocessor  Crypto Express4 Coprocessor	DSS tokens are not supported.

## PKA Key Import (CSNDPKI and CSNFPKI)

Use this service to import an external PKA private key token. (The private key must consist of a PKA private key and public key.) The secret values of the key may be:

- Clear
- Encrypted under a limited-authority DES importer key or an AES importer key if the *source\_key\_identifier* is an RSA token
- Encrypted under an AES Key Encryption Key if the *source\_key\_identifier* is an ECC token

This service can also import a clear PKA key. The PKA key token build service creates a clear PKA key token.

This service can also import an external trusted block token for use with the remote key export callable service.

Output of this service is an ICSF internal token of the RSA, DSS, or ECC private key or trusted block.

The callable service name for AMODE(64) invocation is CSNFPKI.

**Restriction:** DSS keys are not supported on the PCIXCC or CCA Crypto Express coprocessor.

## Format

```
CALL CSNDPKI(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    source_key_identifier_length,  
    source_key_identifier,  
    importer_key_identifier,  
    target_key_identifier_length,  
    target_key_identifier)
```

## Parameters

**return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

## reason\_code

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

## exit data

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

## rule array count

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. This may be 0 or 1.

**rule array**

Direction: Input

Type: Character String

The `rule_array` parameter is an array of keywords. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The `rule_array` keywords are:

Table 244. Keywords for PKA Key Import

Keyword	Meaning
<i>Token Type (optional)</i>	
RSA	Specifies that the key token is for an RSA key. This is the default.
ECC	Specifies that the key token is for an ECC key.
<i>Transport key type (optional)</i>	
IKEK-AES	The <i>importer_key_identifier</i> is a AES key.
IKEK-DES	The <i>importer_key_identifier</i> is a DES key. This is the default.

source key identifier length

Direction: Input

Type: Integer

## PKA Key Import

The length of the *source\_key\_identifier* parameter. The maximum size is 3500 bytes.

### **source\_key\_identifier**

Direction: Input

Type: String

Contains an external token or label of a PKA private key, without section identifier 0x14 (Trusted Block Information), or the trusted block in external form as produced by the Trusted Block Create (CSNDTBC and CSNETBC) service with the ACTIVATE keyword.

If a PKA private key without the section identifier 0x14 is passed in:

- There are no qualifiers. A retained key can not be used.
- The key token must contain both public-key and private-key information. The private key can be in cleartext or it can be enciphered.
- This is the output of the PKA key generate (CSNDPKG) callable service or the PKA key token build (CSNDPKB) callable service.
- If encrypted, it was created on another platform.

If a PKA key token with section 0x14 is passed in:

- This service will be used to encipher the MAC key within the trusted block under the PKA master key instead of the IMP-PKA key-encrypting key.
- The *importer\_key\_identifier* must contain an IMP-PKA KEK in this case.

### **importer\_key\_identifier**

Direction: Input/Output

Type: String

| A variable-length field containing an AES or DES key identifier used to wrap  
| the imported key. For RSA keys, this is either a DES limited authority transport  
| key (IMP-PKA) or an AES transport key. For trusted blocks, this must be a DES  
| limited authority transport key (IMP-PKA). For ECC keys, this must be an AES  
| transport key.

This parameter contains one of the following:

- 64-byte label of a CKDS record that contains the transport key.
- 64-byte DES internal key token containing the transport key.
- a variable-length AES internal key token containing the transport key.

This parameter is ignored for clear tokens.

### **target\_key\_identifier\_length**

Direction: Input/Output

Type: Integer

The length of the *target\_key\_identifier* parameter. The maximum size is 3500 bytes. On output, and if the size is of sufficient length, the variable is updated with the actual length of the *target\_key\_identifier* field.

### **target\_key\_identifier**

Direction: Input/Output

Type: String

This field contains the internal token or label of the imported PKA private key or a Trusted Block. If a label is specified on input, a PKDS record with this label must exist. The PKDS record with this label will be overwritten with

imported key unless the existing record is a retained key. If the record is a retained key, the import will fail. A retained key record cannot be overwritten. If no label is specified on input, this field is ignored.

## Restrictions

This service imports RSA keys of up to 4096 bits. However, the hardware configuration sets the limits on the modulus size of keys for digital signatures and key management; thus, the key may be successfully imported but fail when used if the limits are exceeded.

The *importer\_key\_identifier* is a limited-authority key-encrypting key.

CRT form tokens with a private section ID of X'05' cannot be imported into ICSF.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

An RSA modulus-exponent form token imported on a PCICC, PCIXCC, or CCA Crypto Express coprocessor results in a X'06' format, while a token imported on a Cryptographic Coprocessor Feature will result in a X'02' format. If the modulus length is less than 512, the token will be imported on the CCF, and it will be X'02' format.

This service imports keys of any modulus size up to 4096 bits. However, the hardware configuration sets the limits on the modulus size of keys for digital signatures and key management; thus, the key may be successfully imported but fail when used if the limits are exceeded.

## Access Control Points

The **PKA Key Import** access control point controls the function of this service. If the *source\_key\_token* parameter points to a trusted block, the **PKA Key Import - Import an External Trusted Block** access control point must also be enabled.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.



Table 245. PKA key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	<p>The request will be processed on the CCF when</p> <ul style="list-style-type: none"> <li>the <i>source_key_identifier</i> contains an RSA modulus-exponent private key with a modulus length of less than 512 bits</li> <li>the <i>source_key_identifier</i> contains a DSS private key</li> </ul> <p>RSA keys with moduli greater than 1024-bit length are not supported.</p>
	PCI Cryptographic Coprocessor	<p>The request will be processed on the PCICC when</p> <ul style="list-style-type: none"> <li>the <i>source_key_identifier</i> contains an RSA modulus-exponent private key with a modulus length of a least 512 bits</li> <li>the <i>source_key_identifier</i> contains an RSA CRT private key</li> </ul> <p>RSA keys with moduli greater than 2048-bit length are not supported.</p>
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	DSS tokens are not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	<p>DSS tokens are not supported.</p> <p>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</p>
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	<p>DSS tokens are not supported.</p> <p>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</p>
z196 z114	Crypto Express3 Coprocessor	<p>DSS tokens are not supported.</p> <p>ECC External token and Diffie-Hellman support requires the Sep. 2011 or later licensed internal code (LIC).</p> <p>Importing RSA keys wrapped with an AES transport key is not supported.</p>
IBM zEnterprise EC12	Crypto Express3 Coprocessor	DSS tokens are not supported.
	Crypto Express4 Coprocessor	

## PKA Key Token Build (CSNDPKB and CSNFPKB)

This callable service can be used create PKA key tokens. Specifically, it can be used to:

- build external PKA key tokens containing unencrypted private key for DSS, ECC, or RSA keys. You can use this token as input to the PKA Key Import service to obtain an operational internal token containing an enciphered private key.
- build external RSA key tokens with the private key for use with the PKA Key Translate service.
- build a skeleton token for DSS, ECC, and RSA keys that you can use as input to the PKA Key Generate service.
- build a public key token containing a clear unencrypted public key for an ECC, RSA, or DSS key and return the public key in a token format that other PKA services can use directly.

DSS key generation requires this information in the input skeleton token:

- Size of modulus  $p$  in bits
- Prime modulus  $p$
- Prime divisor  $q$
- Public generator  $g$
- Optionally, the private key name

**Note:** DSS standards define restrictions on the prime modulus  $p$ , prime divisor  $q$ , and public generator  $g$ . (Refer to the Federal Information Processing Standard (FIPS) Publication 186 for DSS standards.) This callable service does not verify all of these restrictions. If you do not follow the restrictions, the keys you generate may not be valid DSS keys.

**Restriction:** DSS is supported only on the z900. PKA key token build will still build DSS tokens, but they can only be used on the z900.

ECC key generation requires this information in the skeleton token:

- The key type: ECC
- The type of curve: Prime or Brainpool
- The size of  $P$  in bits: 192, 224, 256, 384 or 521 for Prime curves and 160, 192, 224, 256, 320, 384, or 521 for Brainpool curves
- Key usage information
- Optionally, application associated data

RSA key generation requires this information in the skeleton token:

- In modulus-exponent form:
  - the length of the modulus  $n$  in bits (512-4096)
  - the length of the public exponent  $e$  (optional). There are restrictions on the value and length of the public exponent when the length of the modulus is greater than 2048
  - the length of the private exponent  $d$  (optional)
  - the public exponent  $e$  (optional)
- In Chinese Remainder Theorem form:
  - the length of the modulus  $n$  in bits (512-4096)
  - the length of the public exponent  $e$  (optional)

## PKA Key Token Build

- | – the public exponent *e* (optional)
- | – other optional lengths

The callable service name for AMODE(64) invocation is CSNFPKB.

## Format

```
CALL CSNFPKB(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_value_structure_length,  
    key_value_structure,  
    private_key_name_length,  
    private_key_name,  
    user_definable_associated_data_length,  
    user_definable_associated_data,  
    reserved_2_length,  
    reserved_2,  
    reserved_3_length,  
    reserved_3,  
    reserved_4_length,  
    reserved_4,  
    reserved_5_length,  
    reserved_5,  
    key_token_length,  
    key_token)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. Value must be 1, 2 or 3.

**rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. Table 246 lists the keywords. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

Table 246. Keywords for PKA Key Token Build Control Information

Keyword	Meaning
<i>Key Type (required)</i>	
DSS-PRIV	This keyword indicates building a key token containing both public and private DSS key information. The parameter <i>key_value_structure</i> identifies the input key values, if supplied.
DSS-PUBL	This keyword indicates building a key token containing public DSS key information. The parameter <i>key_value_structure</i> identifies the input key values, if supplied.
RSA-CRT	This keyword indicates building a token containing an RSA private key in the optimized Chinese Remainder Theorem (CRT) form. The parameter <i>key_value_structure</i> identifies the input key values, if supplied.
RSA-PRIV	This keyword indicates building a token containing both public and private RSA key information. The parameter <i>key_value_structure</i> identifies the input key values, if supplied.
RSA-PUBL	This keyword indicates building a token containing public RSA key information. The parameter <i>key_value_structure</i> identifies the input values, if supplied.
RSAMEVAR	This keyword is for creating a key token for an RSA public and private key pair in modulus-exponent form whose modulus is 512 bits or greater.
RSA-AESM	This keyword is for creating a key token for an RSA public and private key in modulus-exponent format. The object protection key is an AES key. The private key section id is X'30'.
RSA-AESC	This keyword is for creating a key token for an RSA public and private key in Chinese-Remainder Theorem format. The object protection key is an AES key. The private key section id is X'31.
ECC-PAIR	This keyword indicates building a token containing both public and private ECC key information. The parameter <i>key_value_structure</i> identifies the input key values, if supplied.

Table 246. Keywords for PKA Key Token Build Control Information (continued)

Keyword	Meaning
ECC-PUBL	This keyword indicates building a token containing public ECC key information. The parameter <i>key_value_structure</i> identifies the input values, if supplied.
<b>Key Usage Control (optional)</b>	
KEY-MGMT	Indicates that an RSA or ECC private key can be used in both the symmetric key import and the digital signature generate callable services.
KM-ONLY	Indicates that an RSA or ECC private key can be used only in symmetric key distribution.
SIG-ONLY	Indicates that an RSA or ECC private key cannot be used in symmetric key distribution. This is the default. Note that for DSS-PRIV the keyword is allowed but extraneous; DSS keys are defined only for digital signature.
<i>Translate Control (optional, only allowed with key types RSA-AESM, RSA-AESC, RSA-PRIV, RSAMEVAR, RSA-CRT, and ECC-PAIR and is valid with all key usage rules.)</i>	
XLATE-OK	Specifies that the private key material can be translated.
NO-XLATE	Indicates key translation is not allowed. This is the default.

**key\_value\_structure\_length**

Direction: Input

Type: Integer

This is a segment of contiguous storage containing a variable number of input clear key values. The length depends on the key type parameter in the rule array and on the actual values input. The length is in bytes.

Table 247. Key Value Structure Length Maximum Values for Key Types

Key Type	Key Value Structure Maximum Value
DSS-PRIV	436
DSS-PUBL	416
RSA-CRT	3500
RSAMEVAR	3500
RSA-AESC	3500
RSA-AESM	3500
RSA-PRIV	648
RSA-PUBL	520
ECC-PAIR	207
ECC-PUBL	139

**key\_value\_structure**

Direction: Input

Type: String

This is a segment of contiguous storage containing a variable number of input clear key values and the lengths of these values in bits or bytes, as specified. The structure elements are ordered, of variable length, and the input key values must be right-justified within their respective structure elements and

padding on the left with binary zeros. If the leading bits of the modulus are zero's, don't count them in the length. Table 248 defines the structure and contents as a function of key type.

Table 248. Key Value Structure Elements for PKA Key Token Build

Offset	Length (bytes)	Description
<b>Key Value Structure: Optimized RSA, Chinese Remainder Theorem form (RSA-CRT, RSA-AESC)</b>		
000	002	Modulus length in bits (512 to 4096). This is required.
002	002	Modulus field length in bytes, "nnn." This value can be zero if the key token is used as a <i>skeleton_key_token</i> in the PKA key generate callable service. This value must not exceed 512.
004	002	Public exponent field length in bytes, "eee." This value can be zero if the key token is used as a <i>skeleton_key_token</i> in the PKA key generate callable service.
006	002	Reserved, binary zero.
008	002	Length of the prime number, p, in bytes, "ppp." This value can be zero if the key token is used as a <i>skeleton_key_token</i> in the PKA key generate callable service. Maximum size of p + q is 512 bytes.
010	002	Length of the prime number, q, in bytes, "qqq." This value can be zero if the key token is used as a <i>skeleton_key_token</i> in the PKA key generate callable service. Maximum size of p + q is 512 bytes.
012	002	Length of $d_p$ , in bytes, "rrr." This value can be zero if the key token is used as a <i>skeleton_key_token</i> in the PKA key generate callable service. Maximum size of $d_p + d_q$ is 512 bytes.
014	002	Length of $d_q$ , in bytes, "sss." This value can be zero if the key token is used as a <i>skeleton_key_token</i> in the PKA key generate callable service. Maximum size of $d_p + d_q$ is 512 bytes.
016	002	Length of U, in bytes, "uuu." This value can be zero if the key token is used as a <i>skeleton_key_token</i> in the PKA key generate callable service. Maximum size of U is 512 bytes.
018	nnn	Modulus, n.

Table 248. Key Value Structure Elements for PKA Key Token Build (continued)

Offset	Length (bytes)	Description
018 + nnn	eee	Public exponent, e. This is an integer such that $1 < e < n$ . e must be odd. When you are building a <i>skeleton_key_token</i> to control the generation of an RSA key pair, the public key exponent can be one of these values: 3, 65537 ( $2^{16} + 1$ ), or 0 to indicate that a full random exponent should be generated. The exponent field can be a null-length field if the exponent value is 0.
018 + nnn + eee	ppp	Prime number, p.
018 + nnn + eee + ppp	qqq	Prime number, q.
018 + nnn + eee + ppp + qqq	rrr	$d_p = d \text{ mod}(p-1)$ .
018 + nnn + eee + ppp + qqq + rrr	sss	$d_q = d \text{ mod}(q-1)$ .
018 + nnn + eee + ppp + qqq + rrr + sss	uuu	$U = q^{-1} \text{ mod}(p)$ .
<b>Key Value Structure: RSA Modulus-Exponent form (RSA-PRIV, RSA-PUBL, RSAMEVAR, RSA-AESM)</b>		
000	002	Modulus length in bits. This is required. When building a skeleton token, the modulus length in bits must be greater than or equal to 512 bits.
002	002	Modulus field length in bytes, "XXX". This value must not exceed 512 when the RSA-PUBL, RSA-AESM, or RSAMEVAR keyword is used, and must not exceed 128 when the RSA-PRIV keyword is used.  This service can build a key token for a public RSA key with a 4096-bit modulus length, or it can build a key token for a 1024-bit modulus length private key.

Table 248. Key Value Structure Elements for PKA Key Token Build (continued)

Offset	Length (bytes)	Description
004	002	Public exponent field length in bytes, "YYY". This value must not exceed 512 when either the RSA-PUBL, RSA-AESM, or RSAMEVAR keyword is used, and must not exceed 128 when the RSA-PRIV keyword is used. This value can be zero if you are using the key token as a skeleton token in the PKA key generate verb. In this case, a random exponent is generated. To obtain a fixed, predetermined public key exponent, you can supply this field and the public exponent as input to the PKA key generate verb.
006	002	Private exponent field length in bytes, "ZZZ". This field can be zero, indicating that private key information is not provided. This value must not exceed 128 bytes. This value can be zero if you are using the key token as a skeleton token in the PKA key generate verb.
008	XXX	Modulus, n. This is an integer such that $1 < n < 2^{2048}$ . The n is the product of p and q for primes p and q.
008 + XXX	YYY	RSA public exponent, e. This is an integer such that $1 < e < n$ . e must be odd. When you are building a <i>skeleton_key_token</i> to control the generation of an RSA key pair, the public key exponent can be one of these values: 3, 65537 ( $2^{16} + 1$ ), or 0 to indicate that a full random exponent should be generated. The exponent field can be a null-length field if the exponent value is 0.
008 + XXX + YYY	ZZZ	RSA secret exponent d. This is an integer such that $1 < d < n$ . The value of d is $e^{-1} \bmod (p-1)(q-1)$ . This can be a null-length field if you are using the key token as a skeleton token in the PKA key generate verb.
<b>Key Value Structure: DSS Private or DSS Public form (DSS-PRIV, DSS-PUBL)</b>		
000	002	Modulus length in bits. This is required.



Table 248. Key Value Structure Elements for PKA Key Token Build (continued)

Offset	Length (bytes)	Description
002	002	Prime modulus field length in bytes, "XXX". You can supply this as a network quantity to the ICSF PKA key generate callable service, which uses the quantity to generate DSS keys. The maximum allowed value is 128.
004	002	Prime divisor field length in bytes, "YYY". You can supply this as a network quantity to the ICSF PKA key generate callable service, which uses the quantity to generate DSS keys. The allowed values are 0 or 20 bytes.
006	002	Public generator field length in bytes, "ZZZ". You can supply this in a skeleton token as a network quantity to the ICSF PKA key generate callable service, which uses the quantity to generate DSS keys. The maximum allowed value is 128 bytes and is exactly the same length as the prime modulus.
008	002	Public key field length in bytes, "AAA". This field can be zero, indicating that the ICSF PKA key generate callable service generates a value at random from supplied or generated network quantities. The maximum allowed value is 128 bytes and is exactly the same length as the prime modulus.
010	002	Secret key field length in bytes, "BBB". This field can be zero, indicating that the ICSF PKA key generate callable service generates a value at random from supplied or generated network quantities. The allowed values are 0 or 20 bytes.
012	XXX	DSS prime modulus p. This is an integer such that $2^{L-1} < p < 2^L$ . The p must be prime. You can supply this value in a skeleton token as a network quantity; it is used in the algorithm that generates DSS keys.
012 + XXX	YYY	DSS prime divisor q. This is an integer that is a prime divisor of p-1 and $2^{159} < q < 2^{160}$ . You can supply this value in a skeleton token as a network quantity; it is used in the algorithm that generates DSS keys.

Table 248. Key Value Structure Elements for PKA Key Token Build (continued)

Offset	Length (bytes)	Description
012 + XXX+ YYY	ZZZ	DSS public generator g. This is an integer such that $1 < g < p$ . You can supply this value in a skeleton token as a network quantity; it is used in the algorithm that generates DSS keys.
012 + XXX+ YYY+ ZZZ	AAA	DSS public key y. This is an integer such that $y = g^x \text{ mod } p$ .
012 + XXX+ YYY+ ZZZ+ AAA	BBB	DSS secret private key x. This is an integer such that $0 < x < q$ . The x is random. You need not supply this value if you specify DSS-PUBL in the rule array.
<b>Key Value Structure: ECC Private/public key pair form (ECC_PAIR)</b>		
000	001	Curve type x'00' Prime Curve x'01' Brainpool Curve
001	001	Reserved x'00'
002	002	Length of p in bits 0x'00C0' Prime P-192 0x'00E0' Prime P-224 0x'0100' Prime P-256 0x'0180' Prime P-384 0x'0209' Prime P-521 0x'00A0' Brain Pool P-160 0x'00C0' Brain Pool P-192 0x'00E0' Brain Pool P-224 0x'0100' Brain Pool P-256 0x'0140' Brain Pool P-320 0x'0180' Brain Pool P-384 0x'0200' Brain Pool P512.

## PKA Key Token Build

Table 248. Key Value Structure Elements for PKA Key Token Build (continued)

Offset	Length (bytes)	Description
004	002	ddd, This field is the length of the private key d value in bytes, This value can be zero if the key token is used as a skeleton key token in the PKA Key Generate callable service. The maximum value could be up to 66 bytes.
006	002	xxx, This field is the length of the public key Q value in bytes. This value can be zero if the key token is used as a skeleton key token in the PKA Key Generate callable service. The maximum value could be up to 133 bytes which includes one byte to indicate if the value is compressed.
008	ddd	Private key d
008 + ddd	xxx	Public Key value Q
<b>Key value Structure: ECC Public form (ECC_PUBL)</b>		
000	001	Curve type: <b>0x'00'</b> Prime Curve <b>0x'01'</b> Brain Pool Curve
000	001	Reserved x'00'

Table 248. Key Value Structure Elements for PKA Key Token Build (continued)

Offset	Length (bytes)	Description
002	002	<p>Length of p in bits</p> <p>0x'00C0' Prime P-192</p> <p>0x'00E0' Prime P-224</p> <p>0x'0100' Prime P-256</p> <p>0x'0180' Prime P-384</p> <p>0x'0209' Prime P-521</p> <p>0x'00A0' Brain Pool P-160</p> <p>0x'00C0' Brain Pool P-192</p> <p>0x'00E0' Brain Pool P-224</p> <p>0x'0100' Brain Pool P-256</p> <p>0x'0140' Brain Pool P-320</p> <p>0x'0180' Brain Pool P-384</p> <p>0x'0200' Brain Pool P512.</p>
004	002	<p>xxx, This field is the length of the public key Q value in bytes. This value can be zero if the key token is used as a skeleton key token in the PKA Key Generate callable service. The maximum value could be up to 133 bytes which includes a one byte value indicating compressed or uncompressed key value.</p>
006	xxx	Public key value Q

**Notes:**

1. All length fields are in binary.
2. All binary fields (exponent, lengths, modulus, and so on) are stored with the high-order byte field first. This integer number is right-justified within the key structure element field.
3. You must supply all values in the structure to create a token containing an RSA or DSS private key for input to the PKA key import service.

**private\_key\_name\_length**

Direction: Input

Type: Integer

## PKA Key Token Build

The length can be 0 or 64.

### **private\_key\_name**

Direction: Input

Type: EBCDIC character

This field contains the name of a private key. The name must conform to ICSF label syntax rules. That is, allowed characters are alphanumeric, national (@, #, \$) or period (.). The first character must be alphabetic or national. The name is folded to upper case and converted to ASCII characters. ASCII is the permanent form of the name because the name should be independent of the platform. The name is then cryptographically coupled with clear private key data prior to its encryption of the private key. Because of this coupling, the name can never change when the key token is already imported. The parameter is not valid with key types DSS-PUBL or RSA-PUBL.

### **user\_definable\_associated\_data\_length**

Direction: Input

Type: Integer

The length of the *user\_definable\_associated\_data* parameter.

Valid for Rule Array Key Type of ECC-PAIR with a maximum value of 100 and must be set to 0 for all other Rule Array Key Types.

### **user\_definable\_associated\_data**

Direction: Input

Type: String

The *user\_definable\_associated\_data* parameter is a pointer to a string variable containing the associated data that will be placed following the IBM associated data in the token. The associated data is data whose integrity but not confidentiality is protected by a key wrap mechanism. It can be used to bind usage control information.

Valid for Rule Array Key Type of ECC-PAIR and is ignored for all others.

### **reserved\_2\_length**

Direction: Input

Type: Integer

Length in bytes of a reserved parameter. You must set this variable to 0.

### **reserved\_2**

Direction: Input

Type: String

The *reserved\_2* parameter identifies a string that is reserved. The service ignores it.

### **reserved\_3\_length**

Direction: Input

Type: Integer.

Length in bytes of a reserved parameter. You must set this variable to 0.

### **reserved\_3**

Direction: Input

Type: String

The *reserved\_3* parameter identifies a string that is reserved. The service ignores it.

#### **reserved\_4\_length**

Direction: Input

Type: Integer.

Length in bytes of a reserved parameter. You must set this variable to 0.

#### **reserved\_4**

Direction: Input

Type: String

The *reserved\_4* parameter identifies a string that is reserved. The service ignores it.

#### **reserved\_5\_length**

Direction: Input

Type: Integer.

Length in bytes of a reserved parameter. You must set this variable to 0.

#### **reserved\_5**

Direction: Input

Type: String

The *reserved\_5* parameter identifies a string that is reserved. The service ignores it.

#### **key\_token\_length**

Direction: Input/Output

Type: Integer

Length of the returned key token. The service checks the field to ensure it is at least equal to the size of the token to return. On return from this service, this field is updated with the exact length of the *key\_token* created. On input, a size of 3500 bytes is sufficient to contain the largest *key\_token* created.

#### **key\_token**

Direction: Output

Type: String

The returned key token containing an unenciphered private or public key. The private key is in an external form that can be exchanged with different Common Cryptographic Architecture (CCA) PKA systems. You can use the public key token directly in appropriate ICSF signature verification or key management services.

## **Usage Notes**

If you are building a skeleton for use in a PKA Key Generate request to generate a retained PKA private key, you must build a private key name section in the skeleton token.

## **Required Hardware**

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

## PKA Key Token Build

Table 249. PKA key token build required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None.	
z114		
IBM zEnterprise EC12	None.	

## PKA Key Token Change (CSNDKTC and CSNFKTC)

The PKA Key Token Change callable service changes PKA key tokens (RSA, DSS, and ECC) or trusted block key tokens, from encipherment under the cryptographic coprocessor's old RSA master key or ECC master key to encipherment under the current cryptographic coprocessor's RSA master key or ECC master key.

- For RSA and DSS key tokens - Key tokens must be private internal PKA key tokens to be changed by this service. PKA private keys encrypted under the Key Management Master Key (KMMK) cannot be reenciphered using this services unless the KMMK has the same value as the Signature Master Key (SMK).
- For trusted block key tokens - Trusted block key tokens must be internal.
- For ECC key tokens - key tokens must be private internal ECC key tokens encrypted under the ECC master key.

The callable service name for AMODE(64) invocation is CSNFKTC.

### Format

```
CALL CSNDKTC(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_identifier_length,  
    key_identifier)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The value must be 1 or 2.

### **rule\_array**

Direction: Input

Type: String

The process rules for the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 250. Rule Array Keywords for PKA Key Token Change

Keyword	Meaning
<i>Algorithm (optional)</i>	
RSA	Specifies that the key token is for a RSA or DSS key or trusted block token. This is the default.
ECC	Specifies that the key token is for an ECC key.
<i>Reencipherment method (required)</i>	



## PKA Key Token Change

Table 250. Rule Array Keywords for PKA Key Token Change (continued)

Keyword	Meaning
RTCMK	<p>If the <i>key_identifier</i> is an RSA key token, the service will change an RSA private key from encipherment with the old RSA master key to encipherment with the current RSA master key.</p> <p>If the <i>key_identifier</i> is a trusted block token, the service will change the trusted block's embedded MAC key from encipherment with the old RSA master key to encipherment with the current RSA master key.</p> <p>If the <i>key_identifier</i> is an ECC key token, the service will change an ECC private key from encipherment with the old ECC master key to encipherment with the current ECC master key.</p>

### key\_identifier\_length

Direction: Input

Type: Integer

The length of the *key\_identifier* parameter. The maximum size is 3500 bytes.

### key\_identifier

Direction: Input/Output

Type: String

Contains an internal key token of an internal RSA, DSS, ECC, or trusted block key.

If the key token is an RSA key token, the private key within the token is securely reenciphered under the current RSA or ECC master key.

If the key token is a Trusted Block key token, the MAC key within the token is securely reenciphered under the current RSA master key.

If the key token is an ECC key token, the private key within the token is securely reenciphered under the current ECC master key.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

To use this service, PKA callable services must be enabled for all RSA and DSS token types. For systems with CEX3C or CEX4C coprocessors, there is no PKA callable services control. The RSA master key must be valid to use this service.

To use this service for ECC tokens, the ECC master key must be valid.

## Access Control Points

The **PKA Key Token Change RTCMK** access control point controls the function of this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 251. PKA key token change required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	ECC not supported.  Trusted key blocks are not supported.  RSA keys with moduli greater than 2048-bit length are not supported.  Only the RTCMK rule keyword is applicable. Additional keywords, if specified, are ignored.
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	ECC not supported.  Trusted key blocks are not supported.
IBM @server zSeries 890	Crypto Express2 Coprocessor	RSA keys with moduli greater than 2048-bit length are not supported.
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	ECC not supported.  RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor	ECC not supported.  RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).
	Crypto Express3 Coprocessor	ECC support requires the Sep. 2010 licensed internal code (LIC).
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor	
	Crypto Express4 Coprocessor	

## PKA Key Translate (CSNDPKT and CSNFPKT)

The PKA key translate callable service is used to do the following:

- Translate a CCA RSA key token into an external smart card key token.

The source CCA RSA key token must be wrapped with a transport key-encrypting key (KEK). The XLATE bit must also be turned on in the key usage byte of the source token. The source token is unwrapped using the specified source transport KEK. The target key token will be wrapped with the specified target transport KEK. Existing information in the target token is overwritten. There are restrictions on which type key can be used for the source and target transport key tokens. These restrictions are enforced by access control points.

- Convert the object protection key (OPK) in an CCA RSA private key token from a DES key to an AES key.

## PKA Key Translate

| The service will convert an existing internal or external RSA private key token.  
| The modulus-exponent and Chinese Remainder Theorem forms are supported.  
| Private key section identifiers 0x06, 0x08, and 0x09 can be converted.

The callable service name for AMODE(64) invocation is CSNFPKT.

## Format

```
CALL CSNDPKT(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    source_key_identifier_length,  
    source_key_identifier,  
    source_transport_key_identifier_length,  
    source_transport_key_identifier,  
    target_transport_key_identifier_length,  
    target_transport_key_identifier,  
    target_key_token_length,  
    target_key_token)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. Value must be 1.

**rule\_array**

Direction: Input

Type: String

A keyword that provides control information to the callable service. See Table 252 for a list. A keyword is left-justified in an 8-byte field and padded on the right with blanks.

Table 252. Keywords for PKA Key Generate Rule Array

Keyword	Meaning
<i>Smartcard Format (required)</i>	
INTDWAKW	Specifies that the source key is an internal DES wrapped token to be converted to an AESKW wrapped token.
EXTDWAKW	Specifies that the source key is an external DES wrapped token to be converted to an AESKW wrapped token.
SCVISA	This keyword indicates translating the key into the smart card Visa proprietary format.
SCCOMME	This keyword indicates translating the key into the smart card Modulus-Exponent format.
SCCOMCRT	This keyword indicates translating the key into the smart card Chinese Remainder Theorem format.

**source\_key\_identifier\_length**

Direction: Input

Type: Integer

Length in bytes of the *source\_key\_identifier* variable. The maximum length is 3500 bytes.

**source\_key\_identifier**

Direction: Input

Type: String

This field contains either a key label identifying an RSA private key token or an RSA public-private key token. For smart card processing, the key must be in an external key token. For OPK conversion, the token may be internal or external. External tokens are wrapped with a DES key encrypting key. When an internal token is specified, the transport keys are not used.

**source\_transport\_key\_identifier\_length**

Direction: Input

Type: Integer

Length in bytes of the *source\_transport\_key\_identifier* parameter. This value must be 64. For format rule INTDWAKW, the length must be zero.

**source\_transport\_key\_identifier**

Direction: Input/Output

Type: String

## PKA Key Translate

This field contains an internal token or label of a DES key-encrypting key. This key is used to unwrap the input RSA key token specified with parameter *source\_key\_identifier*. See “Access Control Points” for details on the type of transport key that can be used

### target\_transport\_key\_identifier\_length

Direction: Input

Type: Integer

Length in bytes of the *target\_transport\_key\_identifier* parameter. When a DES key-encrypting is used, this value must be 64. When an AES key-encrypting key is used, this value is the length of the token. The maximum length is 725. For INTDWAKW, the length must be zero.

### target\_transport\_key\_identifier

Direction: Input/Output

Type: String

This field contains an internal token or label of a DES key-encrypting key. This key is used to wrap the output RSA key returned with parameter *target\_key\_token*. See “Access Control Points” for details on the type of transport key that can be used.

### target\_key\_token\_length

Direction: Input/Output

Type: Integer

Length in bytes of the *target\_key\_token* parameter. On output, the value in this variable is updated to contain the actual length of the *target\_key\_token* produced by the callable service. The maximum length is 3500 bytes.

### target\_key\_token

Direction: Output

Type: String

This field contains the RSA key in the smartcard format specified in the rule array and is protected by the key-encrypting key specified in the *target\_transport\_key* parameter. This is not a CCA token, and cannot be stored in the PKDS.

## Restrictions

CCA RSA ME tokens will not be translated to the SCCOMCRT format. CCA RSA CRT tokens will not be translated to the SCCOMME format. SCVISA only supports Modulus-Exponent (ME) keys.

## Access Control Points

There are access control points that control use of the format rule array keywords and the type of transport keys that can be used.

Table 253. Required access control points for PKA Key Translate

Rule array keyword	Access control point
INTDWAKW	PKA Key Translate – Translate internal key token
EXTDWAKW	PKA Key Translate – Translate external key token
SCVISA	PKA Key Translate - from CCA RSA to SC Visa Format

Table 253. Required access control points for PKA Key Translate (continued)

Rule array keyword	Access control point
SCCOMME	PKA Key Translate - from CCA RSA to SC ME Format
SCCOMCRT	PKA Key Translate - from CCA RSA to SC CRT Format

These access control points control the key type combination shown in this table. One of these access control points must be enabled.

Table 254. Required access control points for source/target transport key combinations

Source transport key type	Target transport key type	Access control point
EXPORTER	EXPORTER	PKA Key Translate - from source EXP KEK to target EXP KEK
IMPORTER	EXPORTER	PKA Key Translate - from source IMP KEK to target EXP KEK
IMPORTER	IMPORTER	PKA Key Translate - from source IMP KEK to target IMP KEK
EXPORTER	IMPORTER	(Not allowed)

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 255. PKA key translate required hardware

Server	Required Cryptographic hardware	Restrictions
IBM @server zSeries 900		This callable service is not supported.
IBM @server zSeries 990		This callable service is not supported.
IBM @server zSeries 890		
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	Requires the Apr. 2009 or later licensed internal code (LIC).
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	Requires the Apr. 2009 or later licensed internal code (LIC).
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	

---

### PKA Public Key Extract (CSNDPKX and CSNFPKX)

Use the PKA public key extract callable service to extract a PKA public key token from a supplied PKA internal or external private key token. This service performs no cryptographic verification of the PKA private token. You can verify the private token by using it in a service such as digital signature generate.

The callable service name for AMODE(64) invocation is CSNFPKX.

#### Format

```
CALL CSNDPKX(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    source_key_identifier_length,  
    source_key_identifier,  
    target_public_key_token_length,  
    target_public_key_token)
```

#### Parameters

##### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

##### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

##### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

##### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

##### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. The value must be 0.

## rule\_array

Direction: Input

Type: String

Reserved field. This field is not used, but you must specify it.

## source\_key\_identifier\_length

Direction: Input

Type: integer

The length of the *source\_key\_identifier* parameter. The maximum size is 3500 bytes. When the *source\_key\_identifier* parameter is a key label, this field specifies the length of the label.

## source\_key\_identifier

Direction: Input/output

Type: string

The internal or external token of a PKA private key or the label of a PKA private key. This can be the input or output from PKA key import or from PKA key generate.

This service supports:

- RSA private key token formats supported on the PCICCC, PCIXCC, or CCA Crypto Express coprocessor. If the *source\_key\_identifier* specifies a label for a private key that has been retained within a PCICCC, PCIXCC, or CEX2C, this service extracts only the public key section of the token.
- ECC private key token formats supported on the CEX3C and CEX4C.

## target\_public\_key\_token\_length

Direction: Input/Output

Type: Integer

The length of the *target\_public\_key\_token* parameter. The maximum size is 3500 bytes. On output, this field will be updated with the actual byte length of the *target\_public\_key\_token*.

## target\_public\_key\_token

Direction: Output

Type: String

This field contains the token of the extracted PKA public key.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the PKDS.

This service extracts the public key from the internal or external form of a private key. However, it does not check the cryptographic validity of the private token.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.



## PKA Public Key Extract

Table 256. PKA public key extract build required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None	
IBM @server zSeries 990	None	
IBM @server zSeries 890		
IBM System z9 EC	None	
IBM System z9 BC		
IBM System z10 EC	None	
IBM System z10 BC		
z196	None	
z114		
IBM zEnterprise EC12	None	

## Retained Key Delete (CSNDRKD and CSNFRKD)

Use the retained key delete callable service to delete a key that has been retained within the PCICC, PCIXCC, or CCA Crypto Express coprocessor. This service also deletes the record that contains the associated key token from the PKDS. It also allows the deletion of a retained key in the PCICC, PCIXCC, or CCA Crypto Express coprocessor even if there isn't a PKDS record, or deletion of a PKDS record for a retained key even if the PCICC, PCIXCC, or CCA Crypto Express coprocessor holding the retained key is not online. Use the *rule\_array* parameter specifying the FORCE keyword and serial number of the PCICC, PCIXCC, or CCA Crypto Express coprocessor that contains the retained key to be deleted. If a PKDS record exists for the same label, but the serial number doesn't match the serial number in *rule\_array*, the service will fail. If any applications still need the public key, use public key extract to create a public key token prior to deletion of the retained key.

The callable service name for AMODE(64) invocation is CSNFRKD.

### Format

```
CALL CSNDRKD(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_label)
```

### Parameters

**return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

#### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords supplied in the *rule\_array* parameter. The value may be 0 or 2.

#### **rule\_array**

Direction: Input

Type: Character String

This parameter may be FORCE and the PCICC, PCIXCC, or CCA Crypto Express coprocessor serial number.

#### **key\_label**

Direction: Input

Type: String

A 64-byte label of a key that has been retained in a PCICC, PCIXCC, or CCA Crypto Express coprocessor.

## **Usage Notes**

ICSF calls the Security Server (RACF) to check authorization to use the Retained Key Delete service and the label of the key specified in *key\_label*.

Retained private keys are domain-specific. Only the LPAR domain that created a Retained private key can delete the key via the Retained Key Delete service.

When a Retained key is deleted using the Retained Key Delete service, ICSF records this event in a type 82 SMF record with a subtype of 15.

If the Retained key does not exist in the PCICC, PCIXCC, or CCA Crypto Express coprocessor and the PKDS record exists and the domain that created the retained key matches the domain of the requestor, ICSF deletes the PKDS record. This

## Retained Key Delete

situation may occur if the PCICCC, PCIXCC, or CCA Crypto Express coprocessor has been zeroized through TKE or the service processor.

If a PKDS record containing the retained key exists but the PCICCC, PCIXCC, or CCA Crypto Express coprocessor holding the retained key is not online, ICSF deletes the PKDS record if the FORCE keyword is specified. The serial number specified in the rule array must be the serial number of the coprocessor where the Retained key was created. The key token in the PKDS record contains this serial number, and the serial number is used to verify that the PKDS record can be deleted.

If the retained key exists on the specified PCICCC, PCIXCC, or CCA Crypto Express coprocessor but there is no corresponding PKDS record, ICSF deletes the retained key from the PCICCC, PCIXCC, or CCA Crypto Express coprocessor if the FORCE keyword is specified.

## Access Control Point

The **Retained Key Delete** access control point controls the function of this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 257. Retained key delete required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	

The callable service name for AMODE(64) invocation is CSNFRKL.

```
CALL CSNDRKL(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_label_mask,  
    retained_keys_count,  
    key_labels_count,  
    key_labels)
```

## Type: Integer

## Retained Key List

The number of keywords supplied in the *rule\_array* parameter. The value must be 0.

### **rule\_array**

Direction: Input

Type: Character String

This parameter is ignored by ICSF.

### **key\_label\_mask**

Direction: Input

Type: String

A 64-byte key label mask that is used to filter the list of key names returned by the verb. You can use a wild card (\*) to identify multiple keys retained within the PCICC, PCIXCC, or CCA Crypto Express coprocessor.

**Note:** If an asterisk (\*) is used, it must be the last character in *key\_label\_mask*. There can only be one \*.

### **retained\_keys\_count**

Direction: Output

Type: Integer

An integer variable to receive the number of retained keys stored within all active PCICC, PCIXCC, or CCA Crypto Express coprocessor.

### **key\_labels\_count**

Direction: Input/Output

Type: Integer

On input this variable defines the maximum number of key labels to be returned. On output this variable defines the total number of key labels returned. The maximum value for this field is 100. The value returned in the *retained\_keys\_count* variable can be larger if you have not provided for the return of a sufficiently large number of key labels in the *key\_labels\_count* field.

### **key\_labels**

Direction: Output

Type: String

A string variable where the key label information will be returned. This field must be at least 64 times the key label count value. The key label information is a string of zero or more 64-byte entries. The first 64-byte entry contains a PCICC, PCIXCC, or CCA Crypto Express coprocessor card serial number, and is followed by one or more 64-byte entries that each contain a key label of a key retained within that PCICC, PCIXCC, or CCA Crypto Express coprocessor. The format of the first 64-byte entry is as follows:

/nnnnnnnnbbbb...bbb

where

"/" is the character "/" (EBCDIC: X'61')

"nnnnnnnn" is the 8-byte cryptographic coprocessor card serial number

"bbbbbb...bbb" is 55 bytes of blank pad characters  
(EBCDIC: X'40')

This information (64-byte card serial number entry followed by one or more 64-byte label entries) is repeated for each active PCICC, PCIXCC, or CCA Crypto Express coprocessor that contains retained keys that match the *key\_label\_mask*. All data returned is EBCDIC characters. The number of bytes of

information returned is governed by the value specified in the *key\_labels\_count* field. The *key\_labels* field must be large enough to hold the number of 64-byte labels specified in the *key\_labels\_count* field plus one 64-byte entry for each active PCICC, PCIXCC, or CCA Crypto Express coprocessor (a maximum of 64 PCICCs, PCIXCCs, or CCA Crypto Express coprocessors).

## Usage Notes

Not all CCA platforms may support multiple PCICC, PCIXCC, or CCA Crypto Express coprocessor cards. In the case where only one card is supported, the *key\_labels* field will contain one or more 64-byte entries that each contain a key label of a key retained within the PCICC, PCIXCC, or CCA Crypto Express coprocessor. There will be no 64-byte entry or entries containing a PCICC, PCIXCC, or CCA Crypto Express coprocessor card serial number.

ICSF calls RACF to check authorization to use the Retained Key List service.

ICSF caller must be authorized to the *key\_label\_mask* name including the \*.

Retained private keys are domain-specific. ICSF lists only those keys that were created by the LPAR domain that issues the Retained Key List request.

## Access Control Points

The **Retained Key List** access control point controls the function of this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 258. Retained key list required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor Crypto Express3 Coprocessor	
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express3 Coprocessor Crypto Express4 Coprocessor	

## Retained Key List

---

## Chapter 11. Key Data Set Management

ICSF provides key stores for symmetric and asymmetric operational key tokens. Symmetric keys tokens (AES, DES and HMAC) are stored in the Cryptographic Key Data Set (CKDS). Asymmetric keys tokens (DSS, RSA, and ECC) and trusted blocks are stored in the PKA Key Data Set (PKDS).

This topic describes the callable services that manage key tokens in the key stores.

- “CKDS Key Record Create (CSNBKRC and CSNEKRC)”
- “CKDS Key Record Create2 (CSNBKRC2 and CSNEKRC2)” on page 615
- “CKDS Key Record Delete (CSNBKRD and CSNEKRD)” on page 617
- “CKDS Key Record Read (CSNBKRR and CSNEKRR)” on page 619
- “CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2)” on page 621
- “CKDS Key Record Write (CSNBKRW and CSNEKRW)” on page 623
- “CKDS Key Record Write2 (CSNBKRW2 and CSNEKRW2)” on page 626
- “Coordinated KDS Administration (CSFCRC and CSFCRC6)” on page 628
- “PKDS Key Record Create (CSNDKRC and CSNFKRC)” on page 632
- “PKDS Key Record Delete (CSNDKRD and CSNFKRD)” on page 634
- “PKDS Key Record Read (CSNDKRR and CSNFKRR)” on page 636
- “PKDS Key Record Write (CSNDKRW and CSNFKRW)” on page 638

---

### CKDS Key Record Create (CSNBKRC and CSNEKRC)

Use the CKDS key record create callable service to add a key record to the CKDS that will be used to store AES and DES tokens. The record contains a key token set to binary zeros and is identified by the label passed in the *key\_label* parameter. This service updates both the DASD copy of the CKDS currently in use by ICSF and the in-storage copy of the CKDS.

The callable service name for AMODE(64) invocation is CSNEKRC).

#### Format

```
CALL CSNBKRC(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    key_label)
```

#### Parameters

##### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

##### **reason\_code**



## CKDS Key Record Create

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **key\_label**

Direction: Input

Type: Character string

The 64-byte label of a record in the CKDS that is the target of this service. The created record contains a key token set to binary zeros and has a key type of NULL.

## Restrictions

The record must have a unique label. Therefore, there cannot be another record in the CKDS with the same label and a different key type.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes

The CKDS key record create callable service checks the syntax of the label provided in the *key\_label* parameter to ensure that it follows the KGUP rules. To bypass label syntax checking, use a preprocessing exit to turn on the bypass parse bit in the Exit Parameter Control Block (EXPB). For more information about preprocessing exits and the EXPB, refer to the *z/OS Cryptographic Services ICSF System Programmer's Guide*.

You must use either the CKDS key record create callable service or KGUP to create an initial record in the CKDS prior to using the CKDS key record write service to update the record with a valid key token. Your applications perform better if you use KGUP to create the initial records and REFRESH the entire in-storage copy of the CKDS, rather than using CKDS key record create to create the initial NULL key entries. This is particularly true if you are creating a large number of key records. CKDS key record create adds a record to a portion of the CKDS that is searched sequentially during key retrieval. Using KGUP followed by a REFRESH puts the null key records in the portion of the CKDS that is ordered in key-label/type sequence. A binary search of the key-label/type sequenced part of the CKDS is more efficient than searching the sequentially ordered section.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 259. CKDS record create required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None.	
z114		
IBM zEnterprise EC12	None.	

## CKDS Key Record Create2 (CSNBKRC2 and CSNEKRC2)

Use this service to add a key record to the CKDS. The record will contain a null key token or the key token supplied in the *key\_token* parameter. The record is identified by the label passed in the *key\_label* parameter.

The callable service name for AMODE(64) is CSNEKRC2.

## Format

```
CALL CSNBKRC2(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_label,
    key_token_length,
    key_token )
```

## Parameters

### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

## CKDS Key Record Create2

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value must be 0.

### **rule\_array**

Direction: Input

Type: String

This parameter is ignored by ICSF.

### **key\_label**

Direction: Input

Type: String

The 64-byte label of a record in the CKDS to be created.

### **key\_token\_length**

Direction: Input

Type: Integer

The length of the field containing the token to be written to the CKDS. If zero is specified, a null token will be added to the CKDS. The maximum value is 725.

### **key\_token**

Direction: Input/Output

Type: String

A symmetric internal token to be written to the CKDS if *key\_token\_length* is non-zero. If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 260. CKDS Key Record Create2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None.	
z114		
IBM zEnterprise EC12	None.	

## CKDS Key Record Delete (CSNBKRD and CSNEKRD)

Use the CKDS key record delete callable service to delete a key record containing a DES or AES token from both the DASD copy of the CKDS and the in-storage copy.

The callable service name for AMODE(64) invocation is CSNEKRD.

## Format

```
CALL CSNBKRD(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_label)
```

## Parameters

### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### reason\_code

## CKDS Key Record Delete

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords supplied in the *rule\_array* parameter. This number must always be 1.

### **rule\_array**

Direction: Input

Type: Character string

The 8 byte keyword that defines the action to be performed. The keyword must be LABEL-DL.

### **key\_label**

Direction: Input

Type: Character string

The 64-byte label of a record in the CKDS that is the target of this service. The record can contain an AES or a DES key token. The record pointed to by this label is deleted.

## Restrictions

The record defined by the *key\_label* must be unique. If more than one record per label is found, the service fails.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes

Secure key tokens cannot be processed when the master key is not loaded.

Clear AES and DES tokens can be processed on a system without a cryptographic coprocessor or accelerator.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 261. CKDS record delete required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None.	
z114		
IBM zEnterprise EC12	None	

## CKDS Key Record Read (CSNBKRR and CSNEKRR)

Use the CKDS key record read callable service to copy an internal AES or DES key token from the in-storage CKDS to application storage. Other cryptographic services can then use the copied key token directly. The key token can also be used as input to the token copying functions of key generate, key import, or secure key import services to create additional NOCV keys.

The callable service name for AMODE(64) invocation is CSNEKRR.

## Format

```
CALL CSNBKRR(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_label,
    key_token)
```

## Parameters

**return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

## CKDS Key Record Read

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it indicating specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **key\_label**

Direction: Input

Type: Character string

The 64-byte label of a record containing an AES or DES token in the in-storage CKDS. The internal key token in this record is returned to the caller.

### **key\_token**

Direction: Output

Type: String

The 64-byte internal key token retrieved from the in-storage CKDS.

## Restrictions

The record defined by the *key\_label* parameter must be unique and must already exist in the CKDS.

If the internal key token is a clear key token, the token is not returned to the caller unless the caller is in supervisor state or system key.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes

Clear AES and DES tokens can be processed on a system without a cryptographic coprocessor or accelerator.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 262. CKDS record read required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None.	
z114		
IBM zEnterprise EC12	None	

## CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2)

Use this callable service to copy a key token from the in-storage CKDS to application storage. Other cryptographic services can then use the copied key token directly.

The callable service name for AMODE(64) is CSNEKRR2.

### Format

```
CALL CSNBKRR2(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_label,
    key_token_length,
    key_token )
```

### Parameters

#### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### reason\_code

Direction: Output

Type: Integer



## CKDS Key Record Read2

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value may be 0.

### **rule\_array**

Direction: Input

Type: String

This parameter is ignored by ICSF.

### **key\_label**

Direction: Input

Type: String

The 64-byte label of a record in the CKDS to be retrieved.

### **key\_token\_length**

Direction: Input/Output

Type: Integer

The length of the buffer for the output token. On input, the length of the buffer. The minimum length is 64 bytes and the maximum length is 725 bytes. On output, this parameter will be updated with the length of the token returned in the *key\_token* parameter.

### **key\_token**

Direction: Output

Type: String

The buffer into which the return key token is written.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 263. CKDS key record read2 required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None.	
z114		
IBM zEnterprise EC12	None	

## CKDS Key Record Write (CSNBKRW and CSNEKRW)

Use the CKDS key record write callable service to write an internal AES or DES key token to the CKDS record specified by the *key\_label* parameter. This service supports writing a record to the CKDS which contains a key token with a control vector which is not supported by the Cryptographic Coprocessor Feature. These records will be written to the CKDS with a key type of CV, unless the key is a DES IMPORTER, EXPORTER, PINGEN, PINVER, IPINENC, or OPINENC type. These key types will be preserved in the CKDS record, even if the control vector is not supported by the Cryptographic Coprocessor Feature.

This service updates both the DASD copy of the CKDS currently in use by ICSF and the in-storage copy. The record you are updating must be unique and must already exist in both the DASD and in-storage copies of the CKDS.

This service supports writing a clear AES or DES key token with non-zero key values to the CKDS.

The callable service name for AMODE(64) invocation is CSNEKRW.

## Format

```
CALL CSNBKRW(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_token,
    key_label)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

#### **key\_token**

Direction: Input/output

Type: String

The 64-byte internal AES or DES key token that is written to the CKDS.

#### **key\_label**

Direction: Input

Type: Character string

The 64-byte label of a record in the CKDS that is the target of this service. The record is updated with the AES or DES internal key token supplied in the *key\_token* parameter.

### Restrictions

The record defined by the *key\_label* parameter must be unique and must already exist in the CKDS.

On CCF systems, writing a NOCV key-encrypting key is restricted to callers in supervisor mode or in system key.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

## Usage Notes

With a PCIXCC or CCA Crypto Express coprocessor, you can write NOCV keys to the CKDS without being in supervisor state.

Secure AES tokens in the CKDS can only be overwritten by a secure AES token encrypted under the same AES master keys. The same is true for secure DES tokens.

DES tokens cannot be overwritten by an AES token. AES tokens cannot be overwritten by a DES token.

Secure key tokens cannot be processed when the master key is not loaded.

Clear AES and DES tokens can be processed on a system without a cryptographic coprocessor or accelerator.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 264. CKDS record write required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None.	
z114		
IBM zEnterprise EC12	None	

## Related Information

You can use this service with the CKDS key record create callable service to write an initial record to key storage. Use it following the key import and key generate callable services to write an operational key imported or generated by these services directly to the CKDS.

## CKDS Key Record Write2 (CSNBKRW2 and CSNEKRW2)

Use the CKDS key record write2 callable service to write an internal symmetric key token to the variable-length CKDS record specified by the *key\_label* parameter. This service updates both the DASD copy of the CKDS currently in use by ICSF and the in-storage copy. The record you are updating must be unique and must already exist in both the DASD and in-storage copies of the CKDS.

The callable service name for AMODE(64) is CSNEKRW2.

### Format

```
CALL CSNBKRW2(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_token_length,
    key_token,
    key_label )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

#### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

#### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value may be 0.

**rule\_array**

Direction: Input

Type: String

This parameter is ignored by ICSF.

**key\_token\_length**

Direction: Input

Type: Integer

The length in bytes of the token to be written to the CKDS. The maximum value is 725.

**key\_token**

Direction: Input/Output

Type: String

An internal symmetric key token to be written to the CKDS. If the token supplied was encrypted under the old master key, the token will be returned encrypted under the current master key.

**key\_label**

Direction: Input

Type: String

The 64-byte label of a record in the CKDS to be overwritten.

## Usage Notes

The Usage Notes for the CKDS Key Record Write callable service also apply to the CKDS Key Record Write2 callable service when writing fixed-length symmetric key tokens (versions X'00', X'01', and X'04').

A key token cannot be overwritten by another key token that doesn't have the exact same algorithm and key type. For example:

- a DES key token cannot be overwritten by an AES token, and an AES key token cannot be overwritten by a DES token
- an AES HMAC token cannot be overwritten by an AES CIPHER token, and an AES CIPHER token cannot be overwritten by an AES HMAC token.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 265. CKDS key record write2 required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	

Table 265. CKDS key record write2 required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 990 IBM @server zSeries 890	None.	
IBM System z9 EC IBM System z9 BC	None.	
IBM System z10 EC IBM System z10 BC	None.	
z196 z114	None.	
IBM zEnterprise EC12	None	

## Coordinated KDS Administration (CSFCRC and CSFCRC6)

Use the coordinated KDS administration callable service to perform a coordinated KDS refresh or a coordinated KDS master key change.

Coordinated KDS refresh is only supported for the CKDS and PKDS. Coordinated KDS refresh is not supported for TKDS.

When used for master key change, applications can continue to run KDS update workloads in parallel, and ICSF guarantees that any dynamic updates will be reflected in the target data set. For coordinated KDS refresh, you should disable KDS update workloads when refreshing to a target data set that is different from the currently-active KDS. This is recommended, because updates occurring to the currently-active KDS might not be reflected in the target data set. ICSF does not enforce manual disablement of dynamic KDS updates prior to a coordinated refresh operation, and will itself internally suspend such updates until the coordinated refresh operation completes. Note that the recommendation to disable KDS updates does not apply to a coordinated refresh when the target data set is the same as the currently-active KDS. In this case, the updates to the currently-active KDS are guaranteed to be in the resulting in-storage KDS when the operation completes.

In a sysplex environment, this callable service enables an application to perform a coordinated sysplex-wide KDS refresh or KDS change master key operation from a single ICSF instance.

The callable service name for AMODE(64) invocation is CSFCRC6.

## Format

```
CALL CSFCRC (
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    function,
    new_data_set_name,
    data_set_type,
    backup_data_set_name,
    archive_data_set_name,
    feedback_length,
    feedback)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the **rule\_array** parameter. The value must be 0.

### **rule\_array**

Direction: Input

Type: String

This parameter is ignored.



## Coordinated KDS Administration

### **function**

Direction: Input

Type: Integer

The function to be performed by this callable service. The value must be 1 for coordinated change master key or 2 for coordinated refresh. The coordinated refresh function is only available for the CKDS and PKDS. Coordinated refresh is not supported by the TKDS. Coordinated change master key is available for the CKDS, PKDS, and TKDS.

### **new\_data\_set\_name**

Direction: Input

Type: String

The name of the new data set to be used by this callable service. For coordinated change master key this data set will be used to reencipher the active KDS data set, and will become the active KDS data set. For coordinated refresh this data set will become the active KDS dataset. This data set name must be a 44 character string with the data set name left justified and padded with blanks.

### **data\_set\_type**

Direction: Input

Type: Integer

The type of data set to be processed by the callable service. This value must be 1 for a CKDS, 2 for PKDS, or 3 for TKDS.

### **backup\_data\_set\_name**

Direction: Input

Type: String

The name of the backup data set to be used by this callable service when performing a coordinated change master key. This parameter is optional. If specified, a backup copy of the reenciphered KDS will be stored in this data set. This data set name must be a 44-character string with the data set name left justified and padded with blanks.

### **archive\_data\_set\_name**

Direction: Input

Type: String

The name of the archive data set to be used by the CRC callable service. This parameter is optional. If specified, the active KDS will be renamed to this data set name after performing the coordinated change master key or coordinated refresh to a new data set. This data set name must be a 44-character string with the data set name left justified and padded with blanks. The CRC service will take the suffix (usually .D or .DATA /.I or .INDEX) from the active KDS and apply them to the archive data set name. If the data or index name contains no suffix, or if the suffix applied to the archive data set name exceeds 44 characters, the request will be rejected.

### **feedback\_length**

Direction: Input

Type: Integer

The length of the feedback field used by the callable service.

**feedback**

Direction: Output

Type: String

A field provided by the caller for the callable service to return additional feedback in.

**Usage Notes**

One or more ICSF instances sharing the same active KDS in a sysplex create a KDS sysplex cluster. All KDS sysplex cluster members must be IPLed and started in order to perform a coordinated refresh or coordinated change master key operation. The coordinated KDS administration functions will not be queued for processing on inactive sysplex cluster members.

SAF will be invoked to verify the caller is authorized to use this callable service. The CSFCRC resource in the CSFSERV class protects access to this callable service. To access this service, callers will be required to have a UACC of READ for the CSFCRC resource.

The coordinated refresh function is only available for the CKDS and PKDS. This function is not supported for the TKDS. The coordinated change master key function is supported for the CKDS, PKDS, and TKDS.

A coordinated refresh on the active KDS requires KDS updates to be suspended. A refresh of the active KDS is only required when a utility (such as KGUP) has altered the KDS VSAM dataset outside of ICSF. Updates must be suspended in this case to allow the in-storage cache of the KDS VSAM data set to be rebuilt and loaded by ICSF.

**Required Hardware**

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 266. Coordinated KDS administration required hardware*

Server	Required cryptographic Hardware for Coordinated Change Master Key	Required cryptographic Hardware for Coordinated Refresh	Restrictions
IBM @server zSeries 900	None	None	This callable service is not supported.
IBM @server zSeries 990	PCIXCC or CEX2C	None	
IBM @server zSeries 890			
IBM System z9 EC	CEX2C	None	
IBM System z9 BC			
IBM System z10 EC	CEX2C or CEX3C	None	
IBM System z10 BC			

Table 266. Coordinated KDS administration required hardware (continued)

Server	Required cryptographic Hardware for Coordinated Change Master Key	Required cryptographic Hardware for Coordinated Refresh	Restrictions
z196	CEX3C	None	
z114			
IBM zEnterprise EC12	CEX3C CEX4C or CEX4P	None	

## PKDS Key Record Create (CSNDKRC and CSNFKRC)

This callable service writes a new record to the PKDS.

The callable service name for AMODE(64) invocation is CSNFKRC.

### Format

```
CALL CSNDKRC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    label,
    token_length,
    token)
```

### Parameters

#### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### reason\_code

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

#### exit\_data\_length

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

**exit\_data**

Direction: Input/Output Type: String

The data that is passed to the installation exit.

**rule\_array\_count**

Direction: Input Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. This parameter is ignored by ICSF.

**rule\_array**

Direction: Input Type: String

This parameter is ignored by ICSF.

**label**

Direction: Input Type: String

The label of the record to be created. A 64 byte character string.

**token\_length**

Direction: Input Type: Integer

The length of the field containing the token to be written to the PKDS. If zero is specified, a null token will be added to the PKDS. The maximum value of *token\_length* is the maximum length of a private RSA or DSS token.

**token**

Direction: Input Type: String

Data to be written to the PKDS if *token\_length* is non-zero. An RSA, DSS, or ECC private token in either external or internal format, or a DSS, RSA, or ECC public token.

## Usage Notes

To use this service, PKA callable services must be enabled for all RSA and DSS token types. For systems with CEX3C or CEX4C coprocessors, there is no PKA callable services control. The RSA master key must be valid to use this service.

To use this service for clear key ECC tokens, a current ECC master key is not required.

To use this service for encrypted key ECC tokens, the ECC master key must be valid.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

## PKDS Key Record Create

Table 267. PKDS key record create required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None	
z114		
IBM zEnterprise EC12	None	

---

## PKDS Key Record Delete (CSNDKRD and CSNFKRD)

Use PKDS key record delete to delete a record from the PKDS.

The callable service name for AMODE(64) invocation is CSNFKRD.

### Format

```
CALL CSNDKRD(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    label)
```

### Parameters

#### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### reason\_code

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned

to it that indicates specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### exit\_data\_length

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

#### exit\_data

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

#### rule\_array\_count

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. This value must be 0, or 1.

#### rule\_array

Direction: Input

Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 268. Keywords for PKDS Key Record Delete

Keyword	Meaning
<i>Deletion Mode (optional)</i> specifies whether the record is to be deleted entirely or whether only its contents are to be erased.	
LABEL-DL	Specifies that the record will be deleted from the PKDS entirely. This is the default deletion mode.
TOKEN-DL	Specifies that the only the contents of the record are to be deleted. The record will still exist in the PKDS, but will contain only binary zeroes.

#### label

Direction: Input

Type: String

The label of the record to be deleted. A 64 byte character string.

## Restrictions

This service cannot delete the PKDS record for a retained key.

## Usage Notes

To use this service, PKA callable services must be enabled for all RSA and DSS token types. For systems with CEX3C or CEX4C coprocessors, there is no PKA callable services control. The RSA master key must be valid to use this service.

## PKDS Key Record Delete

To use this service for clear key ECC tokens, a current ECC master key is not required.

To use this service for encrypted key ECC tokens, the ECC master key must be valid.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 269. PKDS key record delete required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None	
z114		
IBM zEnterprise EC12	None	

---

## PKDS Key Record Read (CSNDKRR and CSNFKRR)

Reads a record from the PKDS and returns the content of the record. This is true even when the record contains a null PKA token.

The callable service name for AMODE(64) invocation is CSNFKRR.

## Format

```
CALL CSNDKRR(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    label,  
    token_length,  
    token)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. This parameter is ignored by ICSF.

### **rule\_array**

Direction: Input

Type: String

This parameter is ignored by ICSF.

### **label**

Direction: Input

Type: String

The label of the record to be read. A 64 byte character string.

### **token\_length**

Direction: Input/Output

Type: Integer

The length of the area to which the record is to be returned. On successful completion of this service, *token\_length* will contain the actual length of the record returned.



## PKDS Key Record Read

**token**

Direction: Output

Type: String

Area into which the returned record will be written. The area should be at least as long as the record.

### Usage Notes

To use this service, PKA callable services must be enabled for all RSA and DSS token types. For systems with CEX3C or CEX4C coprocessors, there is no PKA callable services control. The RSA master key must be valid to use this service.

To use this service for clear key ECC tokens, a current ECC master key is not required.

To use this service for encrypted key ECC tokens, the ECC master key must be valid.

### Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 270. PKDS key record read required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None	
z114		
IBM zEnterprise EC12	None	

---

## PKDS Key Record Write (CSNDKRW and CSNFKRW)

Writes over an existing record in the PKDS.

The callable service name for AMODE(64) invocation is CSNFKRW.

## Format

```
CALL CSNDKRW(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    label,
    token_length,
    token)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in the *rule\_array* parameter. Its value must be 0 or 1.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

## PKDS Key Record Write

Table 271. Keywords for PKDS Key Record Write

Keyword	Meaning
<i>Write Mode (optional)</i> specifies the circumstances under which the record is to be written.	
CHECK	Specifies that the record will be written only if a record of type NULL with the same label exists in the PKDS. If such a record exists, ICSF overwrites it. This is the default condition.
OVERLAY	Specifies that the record will be overwritten regardless of the current content of the record. If a record with the same label exists in the PKDS, ICSF overwrites it.

### label

Direction: Input

Type: String

The label of the record to be overwritten. A 64 byte character string.

### token\_length

Direction: Input

Type: Integer

The length of the field containing the token to be written to the PKDS.

### token

Direction: Input

Type: String

The data to be written to the PKDS, which is a DSS, RSA, or ECC private token in either external or internal format, or a DSS, RSA, or ECC public token.

## Restrictions

This service cannot update a PKDS record for a retained key.

## Usage Notes

The PKDS Key Record Write service will only overwrite NULL tokens and tokens of the same type. For example an RSA token cannot overwrite an ECC or DSS token.

To use this service, PKA callable services must be enabled for all RSA and DSA token types. For systems with CEX3C or CEX4C coprocessors, there is no PKA callable services control. The RSA master key must be valid to use this service.

To use this service for clear key ECC tokens, a current ECC master key is not required.

To use this service for encrypted key ECC tokens, the ECC master key must be valid.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 272. PKDS key record write required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None.	
z114		
IBM zEnterprise EC12	None	

## PKDS Key Record Write

---

## Chapter 12. Utilities

This topic describes these callable services:

- “Character/Nibble Conversion (CSNBXBC and CSNBXCB)”
- “Code Conversion (CSNBXEA and CSNBXAE)” on page 645
- “ICSF Query Algorithm (CSFIQA and CSFIQA6)” on page 647
- “ICSF Query Facility (CSFIQF and CSFIQF6)” on page 652
- “X9.9 Data Editing (CSNB9ED)” on page 673

**Note:** These services are not dependent on the hardware. They will run on any server.

---

### Character/Nibble Conversion (CSNBXBC and CSNBXCB)

Use these utilities to convert a binary string to a character string (CSNBXBC) or convert a character string to a binary string (CSNBXCB).

These utilities do not support invocation in AMODE(64).

#### Format

```
CALL CSNBXBC(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    text_length,  
    source_text,  
    target_text,  
    code_table)
```

```
CALL CSNBXCB(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    text_length,  
    source_text,  
    target_text,  
    code_table)
```

#### Parameters

##### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

##### **reason\_code**

Direction: Output

Type: Integer

## Character/Nibble Conversion

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **text\_length**

Direction: Input/Output

Type: Integer

On input, the *text\_length* contains an integer that is the length of the *source\_text*. The length must be a positive nonzero value. On output, *text\_length* is updated with an integer that is the length of the *target\_text*.

### **source\_text**

Direction: Input

Type: String

This parameter contains the string to convert.

### **target\_text**

Direction: Output

Type: String

The converted text that the callable service returns.

### **code\_table**

Direction: Input

Type: String

A 16-byte conversion table. The code table for binary to EBCDIC conversion is 'X'F0F1F2F3F4F5F6F7F8F9C1C2C3C4C5C6'.

## Usage Notes

These services are structured differently from the other services. They run in the caller's address space in the caller's key and mode.

ICSF need not be active for you to run either of these services. No pre- or post-processing exits are enabled for these services, and no calls to RACF are issued when you run these services.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 273. Character/Nibble conversion required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None.	
z114		
IBM zEnterprise EC12	None	

## Code Conversion (CSNBXEA and CSNBXAE)

Use these utilities to convert ASCII data to EBCDIC data (CSNBXAE) or EBCDIC data to ASCII data (CSNBXEA).

These utilities do not support invocation in AMODE(64).

### Format

```
CALL CSNBXAE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    text_length,
    source_text,
    target_text,
    code_table)
```

```
CALL CSNBXEA(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    text_length,
    source_text,
    target_text,
    code_table)
```



### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

#### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

#### **text\_length**

Direction: Input

Type: Integer

The *text\_length* contains an integer that is the length of the *source\_text*. The length must be a positive nonzero value.

#### **source\_text**

Direction: Input

Type: String

This parameter contains the string to convert.

#### **target\_text**

Direction: Output

Type: String

The converted text that the callable service returns.

#### **code\_table**

Direction: Input

Type: String

A 256-byte conversion table. To use the default code table, you need to pass a full word of hexadecimal zero's. See Appendix G, "EBCDIC and ASCII Default Conversion Tables," on page 959 for contents of the default table.

**Note:** The Transaction Security System code table has 2 additional 8-byte fields that are not used in the conversion process. ICSF accepts either a 256-byte or a 272-byte code table, but uses only the first 256 bytes in the conversion.

## Usage Notes

These services are structured differently than the other services. They run in the caller's address space in the caller's key and mode. ICSF need not be active for you to run either of these services. No pre- or post-processing exits are enabled for these services, and no calls to RACF are issued when you run these services.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 274. Code conversion required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None.	
z114		
IBM zEnterprise EC12	None	

## ICSF Query Algorithm (CSFIQA and CSFIQA6)

Use this utility to retrieve information about the cryptographic and hash algorithms available. You can control the amount of data that is returned by passing in *rule\_array* keywords. Keyword values describe the cryptographic algorithm or hash algorithm you are interested in.

The service returns a table of information in the *returned\_data* parameter. A row of data consists of the algorithm name, the algorithm size, whether or not clear or secure keys are supported and what method ICSF will use to satisfy a request - CPU instructions, a cryptographic accelerator, a cryptographic coprocessor, or software. The service updates the *returned\_data\_length* field with the actual length of the output *returned\_data* field.

The callable service name for AMODE (64) invocation is CSFIQA6.

## Format

```
CALL CSFIQA(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    returned_data_length,
    returned_data,
    reserved_data_length,
    reserved_data)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in *rule\_array*. Value must be 0 or 1.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 275. Keywords for ICSF Query Algorithm

Keyword	Meaning
<i>ALGORITHM (optional)</i>	
AES	Advanced Encryption Standard - symmetric key algorithm
DES	Data Encryption Standard - single length symmetric key algorithm
DSS	Data Signature Standard - public key cryptography algorithm
ECC	Elliptic Curve Cryptography. All curve types.
ECC-PRIM	Elliptic Curve Cryptography using NIST approved PRIME curves
ECC-BP	Elliptic Curve Cryptography using Brain Pool Curves
HMAC	FIPS-198 keyed-hash message authentication code algorithm.
RSA	Rivest-Shamir-Adleman - public key cryptography algorithm, all usage types
RSA-SIG	Rivest-Shamir-Adleman - public key cryptography algorithm, signature usage.
RSA-KM	Rivest-Shamir-Adleman - public key cryptography algorithm, key management usage.
RSA-GEN	Rivest-Shamir-Adleman - public key cryptography algorithm, key generation.
SHA-1	Secure Hash Algorithm 1 - A one way hash algorithm
SHA-2	Secure Hash Algorithm 2 - A one way hash algorithm
MDC-2	Modification Detection Code 2 - MDC-2 specifies two encipherments per 8 bytes of input text
MDC-4	Modification Detection Code 4 - MDC-4 specifies four encipherments per 8 bytes of input text
MD5	Message Digest 5 - A one way hash algorithm
RPMD-160	RIPE MD-160 - A one way hash algorithm
RNGL	Random number generate long callable service
TDES	Data Encryption Standard - double and triple length symmetric key algorithm

### returned\_data\_length

Direction: Input/Output

Type: Integer

The length of the *returned\_data* parameter. Currently, the value must be large enough to handle the request. Allow additional space for future enhancements. On output, this field will contain the actual length of the data returned.

### returned\_data

Direction: Output

Type: String

This field will contain the table output from the service. Depending on the contents of *rule\_array*, multiple rows may be returned. One row in the table contains:

## ICSF Query Algorithm

Table 276. Output for ICSF Query Algorithm

Offset (hex)	Name	Description
0 (X'0')	Algorithm	<p>An 8-byte EBCDIC character string containing the name of the cryptographic algorithm. The character string is padded on the right with blanks. Possible values are:</p> <p>AES  DES (single length DES)  DSS  ECC-PRIM  ECC-BP (Brain Pool)  HMAC  MDC-2  MDC-4  MD5  RNGL  RPMD-160  RSA-GEN  RSA-KM  RSA-SIG  SHA-1  SHA-2  TDES (double and triple length DES)</p>
8 (X'8')	Size	<p>An 8-byte EBCDIC string representing the maximum key, modulus, p value, or hash size. The string is padded with blanks on the right. The size is in bits. This is true for all algorithms except RNGL. For RNGL, the size is in bytes.</p>
16 (X'10')	Key Security	<p>An 8-byte EBCDIC character string containing the string</p> <p>CLEAR  SECURE  NA</p> <p>The string is padded on the right with blanks.</p>
24(X'18')	Implementation	<p>An 8-byte EBCDIC character string containing how the algorithm is implemented. The string is padded on the right with blanks. Possible choices are:</p> <p>ACC - Cryptographic Accelerator  CCF - CCF  COP - Cryptographic Coprocessor  CPU - CPACF  SW - Software</p>

The rows are sorted in the following order:

- Algorithm name - alphabetically A to Z
- Algorithm size - numerically highest to least
- Key security - alphabetically A to Z
- Implementation - alphabetically A to Z

### reserved\_data\_length

Direction: Input

Type: Integer

The length of the *reserved\_data* parameter. Currently, the value must be 0.

**reserved\_data**

Direction: Ignored

Type: String

This field is currently not used.

## Usage Notes

The *rule\_array* keyword allows the caller to select how much information is returned. The returned data can describe all cryptographic support on the base system or it can be filtered by an algorithm.

For example, a *rule\_array\_count* of 0 will return information about all algorithms and key security. A *rule\_array\_count* of 1 and a keyword of 'AES' will return information about the AES algorithm support, both clear and secure AES keys.

Only cryptographic coprocessors in the active state are queried.

In general, a key security of SECURE implies that both SECURE and CLEAR key versions of the algorithm are supported by the processor or the cryptographic coprocessor. The exception is TDES support in CCF on a z900. Only SECURE TDES keys are supported.

This service lists an algorithm as being supported when the cryptographic coprocessor or accelerator is capable of performing the function. It does not reflect when an algorithm is unavailable because TKE was used to disable the function.

RNGL keyword refers to the Random Number Generate Long (CSFBRNGL) callable service. The following is returned for implementation:

- COP - when RNGL is implemented using the RNGL verb in the cryptographic coprocessor.
- CCF- when RNGL is implemented using the CCF random number generate function (z900 machines)
- SW - when RNGL is implemented using a loop around the RNG verb in the cryptographic coprocessor, creating the random number 8 bytes at a time.

When a row of the *returned\_data* table contains a Key Security value of SECURE and an Implementation value of CPU, this indicates that the CSNBSYE and CSNBSYD callable services support the use of key labels for encrypted keys stored in the CKDS. In other words, the required functions in ICSF, CPACF and the cryptographic coprocessor are available.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 277. ICSF Query Algorithm required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	

## ICSF Query Algorithm

Table 277. ICSF Query Algorithm required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None.	
z114		
IBM zEnterprise EC12	None	

## ICSF Query Facility (CSFIQF and CSFIQF6)

Use this utility to retrieve information about ICSF, the cryptographic coprocessors and the CCA code in the coprocessors. This information includes:

- general information about ICSF
- general information about CCA code in a coprocessor
- export control information from a coprocessor
- diagnostic information from a coprocessor

Coprocessor information requests may be directed to a specific ONLINE or ACTIVE coprocessor or any ACTIVE coprocessor.

This service has an interface similar to the IBM 4758 service CSUACFQ. Instead of the output being returned in the rule array, there is a separate output area. The format of the data returned remains the same. This service supports a subset of the keywords supported by CSUACFQ. For the same supported keywords, CSFIQF and CSUACFQ return the same coprocessor-specific information. The service returns information elements in the *returned\_data* field and updates the *returned\_data\_length* with the actual length of the output *returned\_data* field.

The callable service name for AMODE(64) invocation is CSFIQF6.

### Format

```
CALL CSFIQF(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    returned_data_length,  
    returned_data,  
    reserved_data_length,  
    reserved_data)
```

## Parameters

**return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

## reason\_code

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

## exit\_data

Direction: Ignored

Type: String

This field is ignored.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in *rule\_array*. Value must be 1 or 2

**rule\_array**

Direction: Input

Type: String

Keywords that provide control information to callable services. The keywords are left-justified in an 8-byte field and padded on the right with blanks. The keywords must be in contiguous storage. Specify one or two of the values in Table 278.

Table 278. Keywords for ICSF Query Service

Keyword	Meaning
<i>Coprocessor (optional) - parameter is ignored for ICSFSTAT, ICSFST2, and ICSFSP11</i>	
COPROCxx	Specifies the specific coprocessor to execute the request. xx may be 00 through 63 inclusive. This may be the processor number of any coprocessor. The processor number of any accelerator is not supported. If specified with rule STATP11, the processor number must be that of a Enterprise PKCS #11 coprocessor. For all other rules, it must be that of a CCA coprocessor.



Table 278. Keywords for ICSF Query Service (continued)

Keyword	Meaning
ANY	Process request on any ACTIVE cryptographic coprocessor. This is the default.
nnnnnnnn	Specifies the 8-byte serial number of the coprocessor to execute the request. If specified with rule STATP11, the processor number must be that of a Enterprise PKCS #11 coprocessor. For all other rules, it must be that of a CCA coprocessor.
<b>Information to return (required)</b>	
ICSFSTAT	Get ICSF related status information.
ICSFST2	Get coprocessor-related basic status information.
ICSFSP11	Get ICSF-related PKCS #11 status information
NUM-DECT	Get the number of bytes of storage required for the output of a STATDECT request.
STATAES	Get status information on AES enablement and the AES master key registers.
STATCCA	Get CCA-related status information.
STATCCAE	Get CCA-related extended status information.
STATCARD	Get coprocessor-related basic status information.
STATDECT	Get the PIN decimalization tables loaded. The format of the data is shown under the <i>returned_data</i> parameter. The length of the data is 20 bytes per decimalization table. The NUM-DECT option will return the storage required for this option. The maximum length of the data is 2000 bytes.
STATDIAG	Get coprocessor-related basic status information.
STATAPKA	Get status information on ECC enablement and the ECC master key registers.
STATEID	Get coprocessor-related basic status information.
STATEXPT	Get coprocessor-related basic status information.
WRAPMTHD	Get coprocessor-related default configuration setting for the wrapping method.
STATP11	Get Enterprise PKCS #11 coprocessor-related status information.

### returned\_data\_length

Direction: Input/Output

Type: Integer

The length of the *returned\_data* parameter. Currently, the value must be at least eight times the number of elements returned for the *rule\_array* keyword specified. Allow additional space for future enhancements. On output, this field will contain the actual length of the data returned.

### returned\_data

Direction: Output

Type: String/Integer

This field will contain the output from the service. The format of the output depends on the *rule\_array* keyword. The format of the data is defined in the tables below, which describe the output for each keyword.

When the format is 8-byte elements that contain numbers, those numbers are represented by numeric characters which are left-justified and padded on the right with space characters. For example, a *returned\_data* element which contains the number two will contain the character string '2 '.

For option NUM-DECT, the output is a 4-byte integer.

The output *returned\_data* for the ICSFSTAT keyword is defined in Table 279.

Table 279. Output for option ICSFSTAT

Element Number	Name	Description										
1	FMID	8-byte ICSF FMID										
2	ICSF Status Field 1	<div>Status of ICSF</div> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>ICSF started</td></tr><tr><td>1</td><td>ICSF initialized (CCVINIT is on)</td></tr><tr><td>2</td><td>SYM-MK (DES master key) valid (CCVTMK is on)</td></tr><tr><td>3</td><td>PKA callable services enabled (see “Usage Notes” on page 672)</td></tr></table>	Number	Meaning	0	ICSF started	1	ICSF initialized (CCVINIT is on)	2	SYM-MK (DES master key) valid (CCVTMK is on)	3	PKA callable services enabled (see “Usage Notes” on page 672)
Number	Meaning											
0	ICSF started											
1	ICSF initialized (CCVINIT is on)											
2	SYM-MK (DES master key) valid (CCVTMK is on)											
3	PKA callable services enabled (see “Usage Notes” on page 672)											
3	ICSF Status Field 2	<div>Status of ICSF</div> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>64-bit callers not supported</td></tr><tr><td>1</td><td>64-bit callers supported</td></tr><tr><td>2</td><td>64-bit callers supported, and a TKDS has been specified for the storage of persistent PKCS #11 objects.</td></tr></table>	Number	Meaning	0	64-bit callers not supported	1	64-bit callers supported	2	64-bit callers supported, and a TKDS has been specified for the storage of persistent PKCS #11 objects.		
Number	Meaning											
0	64-bit callers not supported											
1	64-bit callers supported											
2	64-bit callers supported, and a TKDS has been specified for the storage of persistent PKCS #11 objects.											

Table 279. Output for option ICSFSTAT (continued)

4	CPACF	CPACF availability	
		<b>Number</b>	<b>Meaning</b>
		0	CPACF not available
		1	SHA-1 available only
		2	DES/TDES enabled
		3	SHA-224 and SHA-256 are available
		4	SHA-224 and SHA-256, DES and TDES are available
		5	SHA-384 and SHA-512 are available
		6	SHA-384 and SHA-512, DES and TDES are available
		7	Encrypted CPACF functions available.
		8	OFB, CFB, and GCM CPACF functions are available.
5	AES	AES availability for clear keys	
		<b>Number</b>	<b>Meaning</b>
		0	AES not available
		1	AES software only
		2	AES-128
		3	AES-192 and AES-256
6	DSA	DSA algorithm availability	
		<b>Number</b>	<b>Meaning</b>
		0	DSA not available
		1	DSA 1024 key size
		2	DSA 2048 key size
7	RSA Signature	RSA Signature key length	
		<b>Number</b>	<b>Meaning</b>
		0	RSA not available
		1	RSA 1024 key size
		2	RSA 2048 key size
		3	RSA 4096 key size
8	RSA Key Management	RSA Key Management key length	
		<b>Number</b>	<b>Meaning</b>
		0	RSA not available
		1	RSA 1024 key size
		2	RSA 2048 key size
		3	RSA 4096 key size

Table 279. Output for option ICSFSTAT (continued)

9	RSA Key Generate	RSA Key Generate	
		<b>Number</b>	<b>Meaning</b>
		0	Service not available
		1	Service available - 2048 bit modulus
10	Accelerators	Availability of clear RSA key accelerators (PCICAs)	
		<b>Number</b>	<b>Meaning</b>
		0	Not available
		1	At least one available for application use.
11	Accelerator Key Size	Clear key size supported by Accelerators. There must be at least one Accelerator available for use for this field to contain valid information.	
		<b>Number</b>	<b>Meaning</b>
		0	RSA-ME key size of 2048, CRT key size of 2048.
		1	RSA-ME key size of 4096, CRT key size of 4096.
12	Future Use	Currently blanks	

The output *returned\_data* for the ICSFSP11 keyword is defined in Table 280.

Table 280. Output for option ICSFSP11

Element Number	Name	Description
1	P11-MK State	Status of the P11-MK
		<b>Number</b>
		<b>Meaning</b>
		0 P11-MK not active 1 P11-MK active
2	FIPS Mode	ICSF PKCS #11 FIPS mode
		<b>Number</b>
		<b>Meaning</b>
		0 FIPS no enforcement mode 1 FIPS compatibility mode 2 FIPS mode
3-12	Future use	Currently blanks

The output *returned\_data* for the ICSFST2 keyword is defined in Table 281.

Table 281. Output for option ICSFST2

Element Number	Name	Description
----------------	------	-------------

Table 281. Output for option ICSFST2 (continued)

1	Version	Version of the ICSFST2 returned_data. Initial value is 1. It covers elements 1 through 12.								
2	FMID	8-byte ICSF FMID.								
3	ICSF Status Field 1	Status of ICSF <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>PKA callable services disabled</td></tr><tr><td>1</td><td>PKA callable services enabled (see “Usage Notes” on page 672)</td></tr></table>	Number	Meaning	0	PKA callable services disabled	1	PKA callable services enabled (see “Usage Notes” on page 672)		
Number	Meaning									
0	PKA callable services disabled									
1	PKA callable services enabled (see “Usage Notes” on page 672)									
4	ICSF Status Field 2	Status of ICSF <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>PKCS #11 is not available</td></tr><tr><td>1</td><td>PKCS #11 is available</td></tr></table>	Number	Meaning	0	PKCS #11 is not available	1	PKCS #11 is available		
Number	Meaning									
0	PKCS #11 is not available									
1	PKCS #11 is available									
5	ICSF Status Field 3	Status of ICSF <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>ICSF started</td></tr><tr><td>1</td><td>ICSF initialized</td></tr><tr><td>2</td><td>AES master key valid</td></tr></table>	Number	Meaning	0	ICSF started	1	ICSF initialized	2	AES master key valid
Number	Meaning									
0	ICSF started									
1	ICSF initialized									
2	AES master key valid									
6	ICSF Status Field 4	Status of ICSF <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>Secure key AES not available</td></tr><tr><td>1</td><td>Secure key AES is available</td></tr></table>	Number	Meaning	0	Secure key AES not available	1	Secure key AES is available		
Number	Meaning									
0	Secure key AES not available									
1	Secure key AES is available									

Table 281. Output for option ICSFST2 (continued)

7	ICSF Status Field 5	<p>An 8-character numeric character string summarizing the current Key Store Policy.</p> <p>The first character in this string indicates if Key Token Authorization Checking controls have been enabled for the CKDS in either warning or fail mode, and, if so, if the Default Key Label Checking control has also been enabled. The numbers that can appear in the first character of this string are:</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>Key Token Authorization Checking is not enabled for the CKDS.</td></tr><tr><td>1</td><td>Key Token Authorization Checking for CKDS is enabled in FAIL mode. <b>Key Store Policy is active for CKDS.</b> Default Key Label Checking is not enabled.</td></tr><tr><td>2</td><td>Key Token Authorization Checking for CKDS is enabled in WARN mode. <b>Key Store Policy is active for CKDS.</b> Default Key Label Checking is not enabled.</td></tr><tr><td>3</td><td>Key Token Authorization Checking for CKDS is enabled in FAIL mode. <b>Key Store Policy is active for CKDS.</b> Default Key Label Checking is also enabled.</td></tr><tr><td>4</td><td>Key Token Authorization Checking for CKDS is enabled in WARN mode. <b>Key Store Policy is active for CKDS.</b> Default Key Label Checking is also enabled.</td></tr></table> <p>The second character in this string indicates if Duplicate Key Token Checking controls have been enabled for the CKDS. The numbers that can appear in the second character of this string are:</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>Duplicate Key Token Checking is not enabled for the CKDS.</td></tr><tr><td>1</td><td>Duplicate Key Token Checking is enabled for the CKDS. <b>Key Store Policy is active for CKDS.</b></td></tr></table>	Number	Meaning	0	Key Token Authorization Checking is not enabled for the CKDS.	1	Key Token Authorization Checking for CKDS is enabled in FAIL mode. <b>Key Store Policy is active for CKDS.</b> Default Key Label Checking is not enabled.	2	Key Token Authorization Checking for CKDS is enabled in WARN mode. <b>Key Store Policy is active for CKDS.</b> Default Key Label Checking is not enabled.	3	Key Token Authorization Checking for CKDS is enabled in FAIL mode. <b>Key Store Policy is active for CKDS.</b> Default Key Label Checking is also enabled.	4	Key Token Authorization Checking for CKDS is enabled in WARN mode. <b>Key Store Policy is active for CKDS.</b> Default Key Label Checking is also enabled.	Number	Meaning	0	Duplicate Key Token Checking is not enabled for the CKDS.	1	Duplicate Key Token Checking is enabled for the CKDS. <b>Key Store Policy is active for CKDS.</b>
Number	Meaning																			
0	Key Token Authorization Checking is not enabled for the CKDS.																			
1	Key Token Authorization Checking for CKDS is enabled in FAIL mode. <b>Key Store Policy is active for CKDS.</b> Default Key Label Checking is not enabled.																			
2	Key Token Authorization Checking for CKDS is enabled in WARN mode. <b>Key Store Policy is active for CKDS.</b> Default Key Label Checking is not enabled.																			
3	Key Token Authorization Checking for CKDS is enabled in FAIL mode. <b>Key Store Policy is active for CKDS.</b> Default Key Label Checking is also enabled.																			
4	Key Token Authorization Checking for CKDS is enabled in WARN mode. <b>Key Store Policy is active for CKDS.</b> Default Key Label Checking is also enabled.																			
Number	Meaning																			
0	Duplicate Key Token Checking is not enabled for the CKDS.																			
1	Duplicate Key Token Checking is enabled for the CKDS. <b>Key Store Policy is active for CKDS.</b>																			

Table 281. Output for option ICSFST2 (continued)

	<p>The third character in this string indicates if Key Token Authorization Checking controls have been enabled for the PKDS in either warning or fail mode, and, if so, if the Default Key Label Checking control has also been enabled. The numbers that can appear in the third character of this string are:</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>Key Token Authorization Checking is not enabled for the PKDS.</td></tr><tr><td>1</td><td>Key Token Authorization Checking for PKDS is enabled in FAIL mode. <b>Key Store Policy is active for PKDS.</b> Default Key Label Checking is not enabled.</td></tr><tr><td>2</td><td>Key Token Authorization Checking for PKDS is enabled in WARN mode. <b>Key Store Policy is active for PKDS.</b> Default Key Label Checking is not enabled.</td></tr><tr><td>3</td><td>Key Token Authorization Checking for PKDS is enabled in FAIL mode. <b>Key Store Policy is active for PKDS.</b> Default Key Label Checking is also enabled.</td></tr><tr><td>4</td><td>Key Token Authorization Checking for PKDS is enabled in WARN mode. <b>Key Store Policy is active for PKDS.</b> Default Key Label Checking is also enabled.</td></tr></table> <p>The fourth character in this string indicates if Duplicate Key Token Checking controls have been enabled for the PKDS. The numbers that can appear in the fourth character of this string are:</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>Duplicate Key Token Checking is not enabled for the PKDS.</td></tr><tr><td>1</td><td>Duplicate Key Token Checking is enabled for the PKDS. <b>Key Store Policy is active for PKDS.</b></td></tr></table>	Number	Meaning	0	Key Token Authorization Checking is not enabled for the PKDS.	1	Key Token Authorization Checking for PKDS is enabled in FAIL mode. <b>Key Store Policy is active for PKDS.</b> Default Key Label Checking is not enabled.	2	Key Token Authorization Checking for PKDS is enabled in WARN mode. <b>Key Store Policy is active for PKDS.</b> Default Key Label Checking is not enabled.	3	Key Token Authorization Checking for PKDS is enabled in FAIL mode. <b>Key Store Policy is active for PKDS.</b> Default Key Label Checking is also enabled.	4	Key Token Authorization Checking for PKDS is enabled in WARN mode. <b>Key Store Policy is active for PKDS.</b> Default Key Label Checking is also enabled.	Number	Meaning	0	Duplicate Key Token Checking is not enabled for the PKDS.	1	Duplicate Key Token Checking is enabled for the PKDS. <b>Key Store Policy is active for PKDS.</b>
Number	Meaning																		
0	Key Token Authorization Checking is not enabled for the PKDS.																		
1	Key Token Authorization Checking for PKDS is enabled in FAIL mode. <b>Key Store Policy is active for PKDS.</b> Default Key Label Checking is not enabled.																		
2	Key Token Authorization Checking for PKDS is enabled in WARN mode. <b>Key Store Policy is active for PKDS.</b> Default Key Label Checking is not enabled.																		
3	Key Token Authorization Checking for PKDS is enabled in FAIL mode. <b>Key Store Policy is active for PKDS.</b> Default Key Label Checking is also enabled.																		
4	Key Token Authorization Checking for PKDS is enabled in WARN mode. <b>Key Store Policy is active for PKDS.</b> Default Key Label Checking is also enabled.																		
Number	Meaning																		
0	Duplicate Key Token Checking is not enabled for the PKDS.																		
1	Duplicate Key Token Checking is enabled for the PKDS. <b>Key Store Policy is active for PKDS.</b>																		

Table 281. Output for option ICSFST2 (continued)

		<p>The fifth character in this string indicates if Granular Key Label Access controls have been enabled in WARN or FAIL mode. The numbers that can appear in the fifth character of this string are:</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>Granular Key Label Access controls are not enabled.</td></tr><tr><td>1</td><td>Granular Key Label Access control is enabled in FAIL mode</td></tr><tr><td>2</td><td>Granular Key Label Access control is enabled in WARN mode</td></tr></table> <p>The sixth character in this string indicates if Symmetric Key Label Export controls have been enabled for AES and/or DES keys. The numbers that can appear in the sixth character of this string are:</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>Symmetric Key Label Export controls are not enabled.</td></tr><tr><td>1</td><td>Symmetric Key Label Export control is enabled for DES keys only.</td></tr><tr><td>2</td><td>Symmetric Key Label Export control is enabled for AES keys only.</td></tr><tr><td>3</td><td>Symmetric Key Label Export controls are enabled for both DES and AES keys.</td></tr></table>	Number	Meaning	0	Granular Key Label Access controls are not enabled.	1	Granular Key Label Access control is enabled in FAIL mode	2	Granular Key Label Access control is enabled in WARN mode	Number	Meaning	0	Symmetric Key Label Export controls are not enabled.	1	Symmetric Key Label Export control is enabled for DES keys only.	2	Symmetric Key Label Export control is enabled for AES keys only.	3	Symmetric Key Label Export controls are enabled for both DES and AES keys.
Number	Meaning																			
0	Granular Key Label Access controls are not enabled.																			
1	Granular Key Label Access control is enabled in FAIL mode																			
2	Granular Key Label Access control is enabled in WARN mode																			
Number	Meaning																			
0	Symmetric Key Label Export controls are not enabled.																			
1	Symmetric Key Label Export control is enabled for DES keys only.																			
2	Symmetric Key Label Export control is enabled for AES keys only.																			
3	Symmetric Key Label Export controls are enabled for both DES and AES keys.																			



Table 281. Output for option ICSFST2 (continued)

		<p>The seventh character in this string indicates if PKA Key Management Extensions have been enabled in either WARN or FAIL mode, and, if so, whether a SAF key ring or a PKCS #11 token is identified as the trusted certificate repository. (The trusted certificate repository is identified using the APPLDATA field of the CSF.PKAEXTNS.ENABLE profile. If no value is specified in the APPLDATA field, a PKCS #11 token is assumed.) The numbers that can appear in the seventh character of this string are:</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>Symmetric Key Label Export controls are not enabled.</td></tr><tr><td>1</td><td>PKA Key Management Extensions control is enabled in FAIL mode. The trusted certificate repository is a SAF key ring.</td></tr><tr><td>2</td><td>PKA Key Management Extension control is enabled in FAIL mode. The trusted certificate repository is a PKCS #11 token.</td></tr><tr><td>3</td><td>PKA Key Management Extensions control is enabled in WARN mode. The trusted certificate repository is a SAF key ring.</td></tr><tr><td>4</td><td>PKA Key Management Extension control is enabled in WARN mode. The trusted certificate repository is a PKCS #11 token.</td></tr></table>	Number	Meaning	0	Symmetric Key Label Export controls are not enabled.	1	PKA Key Management Extensions control is enabled in FAIL mode. The trusted certificate repository is a SAF key ring.	2	PKA Key Management Extension control is enabled in FAIL mode. The trusted certificate repository is a PKCS #11 token.	3	PKA Key Management Extensions control is enabled in WARN mode. The trusted certificate repository is a SAF key ring.	4	PKA Key Management Extension control is enabled in WARN mode. The trusted certificate repository is a PKCS #11 token.
Number	Meaning													
0	Symmetric Key Label Export controls are not enabled.													
1	PKA Key Management Extensions control is enabled in FAIL mode. The trusted certificate repository is a SAF key ring.													
2	PKA Key Management Extension control is enabled in FAIL mode. The trusted certificate repository is a PKCS #11 token.													
3	PKA Key Management Extensions control is enabled in WARN mode. The trusted certificate repository is a SAF key ring.													
4	PKA Key Management Extension control is enabled in WARN mode. The trusted certificate repository is a PKCS #11 token.													
8	ICSF Status Field 6	<p>Status of ICSF</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>ICSF started</td></tr><tr><td>1</td><td>ICSF initialized</td></tr><tr><td>2</td><td>ECC master key valid, internal keys supported</td></tr><tr><td>3</td><td>ECC master key valid, external keys also supported</td></tr></table>	Number	Meaning	0	ICSF started	1	ICSF initialized	2	ECC master key valid, internal keys supported	3	ECC master key valid, external keys also supported		
Number	Meaning													
0	ICSF started													
1	ICSF initialized													
2	ECC master key valid, internal keys supported													
3	ECC master key valid, external keys also supported													
9	ICSF Status Field 7	<p>Status of ICSF</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>ICSF started</td></tr><tr><td>1</td><td>ICSF initialized</td></tr><tr><td>2</td><td>RSA master key valid</td></tr></table>	Number	Meaning	0	ICSF started	1	ICSF initialized	2	RSA master key valid				
Number	Meaning													
0	ICSF started													
1	ICSF initialized													
2	RSA master key valid													

Table 281. Output for option ICSFST2 (continued)

10	ICSF Status Field 8	Status of ICSF
		<b>Number</b> <b>Meaning</b>
		0                                      ICSF started
		1                                      ICSF initialized
		2                                      DES master key valid
11	ICSF Status Field 9	Status of ICSF
		<b>Number</b> <b>Meaning</b>
		0                                      PKA callable services disabled.
		1                                      PKA callable services enabled.
		See Usage Notes for additional information.
12	Future use	Currently blanks

Table 282. Output for option NUM-DECT

Element Number	Description
1	The number of bytes required for the output of a STATDECT request. This is the number of decimalization tables loaded times 20 bytes. This is a four-byte binary number.

Table 283. Output for option STATAES

Element Number	Name	Description
1	AES NMK Status	State of the AES new master key register:
		<b>Number</b> <b>Meaning</b>
		1              Register is clear
		2              Register contains a partially complete key
2	AES CMK Status	State of the AES current master key register:
		<b>Number</b> <b>Meaning</b>
		1              Register is clear
		2              Register contains a key
3	AES OMK Status	State of the AES old master key register:
		<b>Number</b> <b>Meaning</b>
		1              Register is clear
		2              Register contains a key
4	AES key length enablement	The maximum AES key length that is enabled by the function control vector. The value is 0 (if no AES key length is enabled in the FCV), 128, 192, or 256.

Table 284. Output for option STATCCA

Element Number	Name	Description								
1	NMK Status	State of the DES New Master Key Register: <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>1</td><td>Register is clear</td></tr><tr><td>2</td><td>Register contains a partially complete key</td></tr><tr><td>3</td><td>Register contains a complete key</td></tr></table>	Number	Meaning	1	Register is clear	2	Register contains a partially complete key	3	Register contains a complete key
Number	Meaning									
1	Register is clear									
2	Register contains a partially complete key									
3	Register contains a complete key									
2	CMK Status	State of the DES Current Master Key Register: <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>1</td><td>Register is clear</td></tr><tr><td>2</td><td>Register contains a key</td></tr></table>	Number	Meaning	1	Register is clear	2	Register contains a key		
Number	Meaning									
1	Register is clear									
2	Register contains a key									
3	OMK Status	State of the DES Old Master Key Register: <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>1</td><td>Register is clear</td></tr><tr><td>2</td><td>Register contains a key</td></tr></table>	Number	Meaning	1	Register is clear	2	Register contains a key		
Number	Meaning									
1	Register is clear									
2	Register contains a key									
4	CCA Application Version	A character string that identifies the version of the CCA application program that is running in the coprocessor.								
5	CCA Application Build Date	A character string containing the build date for the CCA application program that is running in the coprocessor.								
6	User Role	A character string containing the Role identifier which defines the host application user's current authority.								

Table 285. Output for option STATCCAE

Element Number	Name	Description
1	Symmetric NMK Status	State of the DES Symmetric New Master Key Register:
		<b>Number</b> <b>Meaning</b>
		1Register is clear
		2Register contains a partially complete key
		3Register contains a complete key
2	Symmetric CMK Status	State of the DES Symmetric Current Master Key Register:
		<b>Number</b> <b>Meaning</b>
		1Register is clear
		2Register contains a key

Table 285. Output for option STATCCAE (continued)

3	Symmetric OMK Status	State of the DES Symmetric Old Master Key Register:	
		<b>Number</b>	<b>Meaning</b>
		1	Register is clear
		2	Register contains a key
4	CCA Application Version	A character string that identifies the version of the CCA application program that is running in the coprocessor.	
5	CCA Application Build Date	A character string containing the build date for the CCA application program that is running in the coprocessor.	
6	User Role	A character string containing the Role identifier which defines the host application user's current authority.	
7	Asymmetric NMK Status	State of the RSA Asymmetric New Master Key Register:	
		<b>Number</b>	<b>Meaning</b>
		1	Register is clear
		2	Register contains a partially complete key
		3	Register contains a complete key
8	Asymmetric CMK Status	State of the RSA Asymmetric Current Master Key Register:	
		<b>Number</b>	<b>Meaning</b>
		1	Register is clear
		2	Register contains a key
9	Asymmetric OMK Status	State of the RSA Asymmetric Old Master Key Register:	
		<b>Number</b>	<b>Meaning</b>
		1	Register is clear
		2	Register contains a key

Table 286. Output for option STATCARD

Element Number	Name	Description
1	Number of installed adapters	The number of active cryptographic coprocessors installed in the machine. This only includes coprocessors that have CCA software loaded (including those with CCA UDX software).
2	DES hardware level	A numeric character string containing an integer value identifying the version of DES hardware that is on the coprocessor.
3	RSA hardware level	A numeric character string containing an integer value identifying the version of RSA hardware that is on the coprocessor.

Table 286. Output for option STATCARD (continued)

4	POST Version	A character string identifying the version of the coprocessor's Power-On Self Test (POST) firmware. The first four characters define the POST0 version and the last four characters define the POST1 version.
5	Coprocessor Operating System Name	A character string identifying the operating system firmware on the coprocessor. Padding characters are blanks.
6	Coprocessor Operating System Version	A character string identifying the version of the operating system firmware on the coprocessor.
7	Coprocessor Part Number	A character string containing the eight-character part number identifying the version of the coprocessor.
8	Coprocessor EC Level	A character string containing the eight-character EC (engineering change) level for this version of the coprocessor.
9	Miniboot Version	A character string identifying the version of the coprocessor's miniboot firmware. This firmware controls the loading of programs into the coprocessor.  The first four characters define the MiniBoot0 version and the last four characters define the MiniBoot1 version.
10	CPU Speed	A numeric character string containing the operating speed of the microprocessor chip, in megahertz.
11	Adapter ID (Also see element number 15)	A unique identifier manufactured into the coprocessor. The coprocessor's Adapter ID is an eight-byte binary value.
12	Flash Memory Size	A numeric character string containing the size of the flash EPROM memory on the coprocessor, in 64-kilobyte increments.
13	DRAM Memory Size	A numeric character string containing the size of the dynamic RAM (DRAM) on the coprocessor, in kilobytes.
14	Battery-Backed Memory Size	A numeric character string containing the size of the battery-backed RAM on the coprocessor, in kilobytes.
15	Serial Number	A character string containing the unique serial number of the coprocessor. The serial number is factory installed and is also reported by the CLU utility in a coprocessor signed status message.

For STATDECT, the output is a table of up to 100 PIN decimalization tables as shown in the following table. The maximum size is 2000 bytes.

Table 287. Output for option STATDECT

Offset	Field	Description
0	Number	Numeric character indicating the table number
3	State	Character indicating the state of the table  L        loaded A        active
4	Table	16-byte decimalization table

Table 288. Output for option STATDIAG

Element Number	Name	Description								
1	Battery State	<p>A numeric character string containing a value which indicates whether the battery on the coprocessor needs to be replaced:</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>1</td><td>Battery is good</td></tr><tr><td>2</td><td>Battery should be replaced</td></tr></table>	Number	Meaning	1	Battery is good	2	Battery should be replaced		
Number	Meaning									
1	Battery is good									
2	Battery should be replaced									
2	Intrusion Latch State	<p>A numeric character string containing a value which indicates whether the intrusion latch on the coprocessor is set or cleared:</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>1</td><td>Latch is cleared</td></tr><tr><td>2</td><td>Latch is set</td></tr></table>	Number	Meaning	1	Latch is cleared	2	Latch is set		
Number	Meaning									
1	Latch is cleared									
2	Latch is set									
3	Error Log Status	<p>A numeric character string containing a value which indicates whether there is data in the coprocessor CCA error log.</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>1</td><td>Error log is empty</td></tr><tr><td>2</td><td>Error log contains data but is not yet full</td></tr><tr><td>3</td><td>Error log is full</td></tr></table>	Number	Meaning	1	Error log is empty	2	Error log contains data but is not yet full	3	Error log is full
Number	Meaning									
1	Error log is empty									
2	Error log contains data but is not yet full									
3	Error log is full									
4	Mesh Intrusion	<p>A numeric character string containing a value to indicate whether the coprocessor has detected tampering with the protective mesh that surrounds the secure module — indicating a probable attempt to physically penetrate the module.</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>1</td><td>No intrusion detected</td></tr><tr><td>2</td><td>Intrusion attempt detected.</td></tr></table>	Number	Meaning	1	No intrusion detected	2	Intrusion attempt detected.		
Number	Meaning									
1	No intrusion detected									
2	Intrusion attempt detected.									
5	Low Voltage Detected	<p>A numeric character string containing a value to indicate whether a power supply voltage was under the minimum acceptable level. This may indicate an attempt to attack the security module.</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>1</td><td>Only acceptable voltages have been detected</td></tr><tr><td>2</td><td>A voltage has been detected under the low-voltage tamper threshold</td></tr></table>	Number	Meaning	1	Only acceptable voltages have been detected	2	A voltage has been detected under the low-voltage tamper threshold		
Number	Meaning									
1	Only acceptable voltages have been detected									
2	A voltage has been detected under the low-voltage tamper threshold									

Table 288. Output for option STATDIAG (continued)

6	High Voltage Detected	<p>A numeric character string containing a value to indicate whether a power supply voltage was higher than the maximum acceptable level. This may indicate an attempt to attack the security module.</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>1</td><td>Only acceptable voltages have been detected</td></tr><tr><td>2</td><td>A voltage has been detected that is higher than the high-voltage tamper threshold</td></tr></table>	Number	Meaning	1	Only acceptable voltages have been detected	2	A voltage has been detected that is higher than the high-voltage tamper threshold
Number	Meaning							
1	Only acceptable voltages have been detected							
2	A voltage has been detected that is higher than the high-voltage tamper threshold							
7	Temperature Range Exceeded	<p>A numeric character string containing a value to indicate whether the temperature in the secure module was outside of the acceptable limits. This may indicate an attempt to obtain information from the module:</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>1</td><td>Temperature is acceptable</td></tr><tr><td>2</td><td>Detected temperature is outside an acceptable limit</td></tr></table>	Number	Meaning	1	Temperature is acceptable	2	Detected temperature is outside an acceptable limit
Number	Meaning							
1	Temperature is acceptable							
2	Detected temperature is outside an acceptable limit							
8	Radiation Detected	<p>A numeric character string containing a value to indicate whether radiation was detected inside the secure module. This may indicate an attempt to obtain information from the module:</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>1</td><td>No radiation has been detected</td></tr><tr><td>2</td><td>Radiation has been detected</td></tr></table>	Number	Meaning	1	No radiation has been detected	2	Radiation has been detected
Number	Meaning							
1	No radiation has been detected							
2	Radiation has been detected							
9, 11, 13, 15, 17	Last Five Commands Run	These five rule-array elements contain the last five commands that were executed by the coprocessor CCA application. They are in chronological order, with the most recent command in element 9. Each element contains the security API command code in the first four characters and the subcommand code in the last four characters.						
10, 12, 14,16, 18	Last Five Return Codes	These five rule-array elements contain the SAPI return codes and reason codes corresponding to the five commands in rule-array elements 9, 11, 13, 15, and 17. Each element contains the return code in the first four characters and the reason code in the last four characters.						

Table 289. Output for option STATEID

Element Number	Name	Description
1	EID	During initialization, a value of zero is set in the coprocessor.

Table 290. Output for option STATEXPT

Element Number	Name	Description

Table 290. Output for option STATEXPT (continued)

1	Base CCA Services Availability	<p>A numeric character string containing a value to indicate whether base CCA services are available.</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>Base CCA services are not available</td></tr><tr><td>1</td><td>Base CCA services are available</td></tr></table>	Number	Meaning	0	Base CCA services are not available	1	Base CCA services are available
Number	Meaning							
0	Base CCA services are not available							
1	Base CCA services are available							
2	CDMF Availability	<p>A numeric character string containing a value to indicate whether CDMF is available.</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>CDMF encryption is not available</td></tr><tr><td>1</td><td>CDMF encryption is available</td></tr></table>	Number	Meaning	0	CDMF encryption is not available	1	CDMF encryption is available
Number	Meaning							
0	CDMF encryption is not available							
1	CDMF encryption is available							
3	56-bit DES Availability	<p>A numeric character string containing a value to indicate whether 56-bit DES encryption is available.</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>56-bit DES encryption is not available</td></tr><tr><td>1</td><td>56-bit DES encryption is available</td></tr></table>	Number	Meaning	0	56-bit DES encryption is not available	1	56-bit DES encryption is available
Number	Meaning							
0	56-bit DES encryption is not available							
1	56-bit DES encryption is available							
4	Triple-DES Availability	<p>A numeric character string containing a value to indicate whether triple-DES encryption is available.</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>Triple-DES encryption is not available</td></tr><tr><td>1</td><td>Triple-DES encryption is available</td></tr></table>	Number	Meaning	0	Triple-DES encryption is not available	1	Triple-DES encryption is available
Number	Meaning							
0	Triple-DES encryption is not available							
1	Triple-DES encryption is available							
5	SET Services Availability	<p>A numeric character string containing a value to indicate whether SET (Secure Electronic Transaction) services are available.</p> <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>0</td><td>SET Services are not available</td></tr><tr><td>1</td><td>SET Services are available</td></tr></table>	Number	Meaning	0	SET Services are not available	1	SET Services are available
Number	Meaning							
0	SET Services are not available							
1	SET Services are available							



Table 290. Output for option STATEXPT (continued)

6	Maximum Modulus for Symmetric Key Encryption	A numeric character string containing the maximum modulus size that is enabled for the encryption of symmetric keys. This defines the longest public-key modulus that can be used for key management of symmetric-algorithm keys.	
		<b>Number</b>	<b>Meaning</b>
		0	DSA not available
		1024	DSA 1024 key size
		2048	DSA 2048 key size
		4096	RSA 4096 key size

Table 291. Output for option STATAPKA

Element Number	Name	Description
1	ECC NMK status	The state of the ECC new master key register:
		<b>Number</b> <b>Meaning</b>
		1      Register is clear.
		2      Register contains a partially complete key.
		3      Register contains a complete key.
2	ECC CMK status	The state of the ECC current master key register:
		<b>Number</b> <b>Meaning</b>
		1      Register is clear. 2      Register contains a key.
3	ECC OMK status	The state of the ECC old master key register:
		<b>Number</b> <b>Meaning</b>
		1      Register is clear. 2      Register contains a key.
4	ECC key length enablement	The maximum ECC curve size that is enabled by the function control vector. The value will be 0 (if no ECC keys are enabled in the FCV) and 521 for the maximum size.

Table 292. Output for option WRAPMTHD

Element Number	Name	Description
1	Internal tokens	Default wrapping method for internal tokens.
		<b>Number</b> <b>Meaning</b>
		0      Keys will be wrapped with the original method
		1      Keys will be wrapped with the enhanced X9.24 method

Table 292. Output for option WRAPMTHD (continued)

2	External tokens	Default wrapping method for external tokens.	
		Number	Meaning
		0	Keys will be wrapped with the original method
		1	Keys will be wrapped with the enhanced X9.24 method

Table 293. Output for option STATP11

Element Number	Name	Description								
1	P11 NMK Status	State of the P11 new master key register: <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>1</td><td>Register is clear</td></tr><tr><td>2</td><td>Register contains an uncommitted key</td></tr><tr><td>3</td><td>Register contains a committed key</td></tr></table>	Number	Meaning	1	Register is clear	2	Register contains an uncommitted key	3	Register contains a committed key
Number	Meaning									
1	Register is clear									
2	Register contains an uncommitted key									
3	Register contains a committed key									
2	P11 CMK	Status State of the P11 current master key register: <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>1</td><td>Register is clear</td></tr><tr><td>2</td><td>Register contains a key</td></tr></table>	Number	Meaning	1	Register is clear	2	Register contains a key		
Number	Meaning									
1	Register is clear									
2	Register contains a key									
3	Compliance Mode	Current compliance mode for the coprocessor. <table><tr><th>Number</th><th>Meaning</th></tr><tr><td>n</td><td>Number that is the sum of the active compliance modes:<ul style="list-style-type: none"><li>• 1 - FIPS 2009</li><li>• 2 - BSI 2009</li><li>• 4 - FIPS 2011</li><li>• 8 - BSI 2011</li></ul></td></tr></table>	Number	Meaning	n	Number that is the sum of the active compliance modes: <ul style="list-style-type: none"><li>• 1 - FIPS 2009</li><li>• 2 - BSI 2009</li><li>• 4 - FIPS 2011</li><li>• 8 - BSI 2011</li></ul>				
Number	Meaning									
n	Number that is the sum of the active compliance modes: <ul style="list-style-type: none"><li>• 1 - FIPS 2009</li><li>• 2 - BSI 2009</li><li>• 4 - FIPS 2011</li><li>• 8 - BSI 2011</li></ul>									
4	Firmware version	Coprocessor PKCS #11 firmware version number as an 8-byte hexadecimal value.								
5	Serial Number	A character string containing the unique serial number of the coprocessor. The serial number is factory installed.								
6 – 12	Future use	Currently blanks								

### reserved\_data\_length

Direction: Input

Type: Integer

The length of the *reserved\_data* parameter. Currently, the value must be 0.

### reserved\_data

Direction: Input

Type: String

This field is currently not used.

## Usage Notes

RACF will be invoked to check authorization to use this service.

PKA key generate available indicates the PKA callable services are enabled and there is at least one ACTIVE coprocessor.

The options ICSFSTAT and ICSFST2 report on the state of PKA callable services. ICSFSTAT reports it in element 2. ICSFST2 reports it in elements 3 and 11. There is a subtle difference between the three options. ICSFSTAT reports PKA callable services as enabled only after the DES master key is loaded and valid. ICSFSTAT does not report PKA callable services as enabled when only the AES master key is loaded and valid. Option ICSFST2 element 3 reports PKA callable services as enabled when the DES and/or AES master key is loaded and valid. Option ICSFST2 element 11 reports PKA callable services as enabled when neither the DES nor AES master keys are loaded and valid.

**Note:** If your system has CEX3C or CEX4C coprocessors, the PKA callable services control will not be available. The PKA callable services state will be the same as the RSA master key. If the RSA master key is active, the PKA callable services will be enabled in the ICSFSTAT and ICSFST2 reports.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 294. ICSF Query Service required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		
IBM System z9 EC	None.	
IBM System z9 BC		
IBM System z10 EC	None.	
IBM System z10 BC		
z196	None.	
z114		
IBM zEnterprise EC12	None	

## X9.9 Data Editing (CSNB9ED)

Use this utility to edit an ASCII text string according to the editing rules of ANSI X9.9-4. It edits the text that the *source\_text* parameter supplies according to these rules. The rules are listed here in the order in which they are applied. It returns the result in the *target\_text* parameter.

1. This service replaces each carriage-return (CR) character and each line-feed (LF) character with a single-space character.
2. It replaces each lowercase alphabetic character (a through z) with its equivalent uppercase character (A through Z).
3. It deletes all characters other than:
  - Alphabets A...Z
  - Numerics 0...9
  - Space
  - Comma ,
  - Period .
  - Dash -
  - Solidus /
  - Asterisk \*
  - Open parenthesis (
  - Close parenthesis )
4. It deletes all leading space characters.
5. It replaces all sequences of two or more space characters with a single-space character.

This utility does not support invocation in AMODE(64).

### Format

```
CALL CSNB9ED(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    text_length,
    source_text,
    target_text)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

## X9.9 Data Editing

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **text\_length**

Direction: Input/Output

Type: Integer

On input, the *text\_length* contains an integer that is the length of the *source\_text*. The length must be a positive, nonzero value. On output, *text\_length* is updated with an integer that is the length of the edited text.

### **source\_text**

Direction: Input

Type: String

This parameter contains the string to edit.

### **target\_text**

Direction: Output

Type: String

The edited text that the callable service returns.

## Usage Notes

This service is structured differently from the other services. It runs in the caller's address space in the caller's key and mode.

ICSF need not be active for the service to run. There are no pre-processing or post-processing exits that are enabled for this service. While running, this service does not issue any calls to RACF.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 295. X9.9 data editing required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	None.	
IBM @server zSeries 990	None.	
IBM @server zSeries 890		

Table 295. X9.9 data editing required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM System z9 EC IBM System z9 BC	None.	
IBM System z10 EC IBM System z10 BC	None.	
z196 z114	None.	
IBM zEnterprise EC12	None	



---

## Chapter 13. Trusted Key Entry Workstation Interfaces

The Trusted Key Entry (TKE) workstation is an optional feature. It offers an alternative to clear key entry. You can use the TKE workstation to load:

- DES master keys, PKA master keys, and operational keys in a *secure* way. CCF only supports Operational Transport and PIN keys. On the PCIXCC and CEX2C, all operational keys may be loaded with TKE V4.1 or higher. On the CEX3C, all operational keys may be loaded with TKE 6.0 or higher. On the CEX4C, all operational keys may be loaded with TKE 7.2 or higher.
- DES-MK and ASYM-MK master keys on the PCICC, PCIXCC, and CCA Crypto Express coprocessors.
- AES-MK master keys and AES operational keys are supported on the z9 and z10 systems with the Nov. 2008 or later licensed internal code (LIC)
- P11-MK master keys on Enterprise PKCS #11 coprocessors (CEX4P)

This topic describes these callable services:

- “PCI Interface Callable Service (CSFPCI and CSFPCI6)”
- “PKSC Interface Callable Service (CSFPKSC)” on page 682

---

### PCI Interface Callable Service (CSFPCI and CSFPCI6)

TKE uses this callable service to send a request to a specific PCI card queue and remove the corresponding response when complete. This service also allows the TKE workstation to query the list of access control points which may be enabled or disabled by a TKE user. This service is synchronous. The return and reason codes reflect the success or failure of the queue functions rather than the success or failure of the actual PCI request.

The callable service name for AMODE(64) invocation is CSFPCI6.

#### Format

```
CALL CSFPCI(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    target_pci_coprocessor,  
    target_pci_coprocessor_serial_number,  
    request_block_length,  
    request_block,  
    request_data_block_length,  
    request_data_block,  
    reply_block_length,  
    reply_block,  
    reply_data_block_length,  
    reply_data_block,  
    masks_length,  
    masks_data)
```



## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. See Appendix A, "ICSF and TSS Return and Reason Codes," for a list of return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. See Appendix A, "ICSF and TSS Return and Reason Codes" for a list of reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you are supplying in *rule\_array*. The value must be 1.

### **rule\_array**

Direction: Input

Type: String

Keyword that provides control information to callable services. The keyword is left-justified in an 8-byte field and padded on the right with blanks. The keyword must be in contiguous storage. These keywords are mutually exclusive:

*Table 296. Keywords for PCI Interface Callable Service*

Keyword	Meaning
ACPOINTS	Queries the list of CCA access control points which may be enabled or disabled by a TKE user.
XPPOINTS	Queries the list of PKCS #11 access control points which may be enabled or disabled by a TKE user.

Table 296. Keywords for PCI Interface Callable Service (continued)

Keyword	Meaning
ACTIVECP	This keyword is a request to call the PCI card initialization code to revalidate the PCI cards. When the PCI card initialization is completed, both the 64-bit mask indicating which of the PCI cards are online and 64-bit mask indicating which of the PCI cards are active will be returned. This keyword is used by the TKE workstation code after the ACTIVATE portion of the domain zeroize command. This is to ensure that the status of the PCI card is accurately reflected to the users. See the <i>masks_data</i> parameter description for more information.
APNUM	Specifies the <i>target_pci_coprocessor</i> field is to be used to submit a CCA request.
XPNUM	Specifies the <i>target_pci_coprocessor</i> field is to be used to submit a PKCS #11 request.
SERIALNO	Specifies the <i>target_pci_coprocessor_serial_number</i> field is to be used to submit a CCA request
PCIMASKS	This keyword is a request to return both the 64-bit mask indicating which of the PCI cards are online and 64-bit mask indicating which of the PCI cards are active. See the <i>masks_data</i> parameter description for more information.
XCPMASK	This keyword is a request to return both the 64-bit mask indicating which of the PCIXCCs and Crypto Express coprocessors are online and the 64-bit mask indicating which of the PCIXCCs and Crypto Express coprocessors are active. See the <i>masks_data</i> parameter description for more information.
CX2MASK	This keyword is a request to return both the 64-bit mask indicating which of the CEX2Cs are online and the 64-bit mask indicating which of the CEX2Cs are active. See the <i>masks_data</i> parameter description for more information.
CX3MASK	This keyword is a request to return both the 64-bit mask indicating which of the CEX3Cs are online and the 64-bit mask indicating which of the CEX3Cs are active. See the <i>masks_data</i> parameter description for more information.
CX4MASK	This keyword is a request to return both the 64-bit mask indicating which of the Crypto Express4 coprocessors are online and the 64-bit mask indicating which of the Crypto Express4 coprocessors are active. Only the coprocessors configured for CCA are to be examined. See the <i>masks_data</i> parameter description for more information.
XP4MASK	This keyword is a request to return both the 64-bit mask indicating which of the Crypto Express4 coprocessors are online and the 64-bit mask indicating which of the Crypto Express4 coprocessors are active. Only the coprocessors configured for PKCS #11 are to be examined. See the <i>masks_data</i> parameter description for more information.
QUERYDOM	This keyword is a request to return a 256-bit mask indicating the controlled domain information from the AP facility. See the <i>masks_data</i> parameter description for more information.

**Note:** When the PCIMASKS, ACTIVECP, XCPMASK, CX2MASK, CX3MASK, CX4MASK, XP4MASK, or QUERYDOM keyword is specified, the

## PCI Interface

*request\_data\_block\_length*, *request\_data\_block*, *reply\_data\_block\_length*, and the *reply\_data\_block* parameters are ignored.

### **target\_pci\_coprocessor**

Direction: Input

Type: Integer

The index of the coprocessor card to which this request is directed. Valid values are between 0 and 64.

### **target\_pci\_coprocessor\_serial\_number**

Direction: Input/Output

Type: String

The serial number of the coprocessor to which the request is directed. This parameter may be used instead of the *target\_pci\_coprocessor* by specifying the SERIALNO rule. The length is 8 bytes. This parameter is updated with the serial number of the card if the request was successfully processed.

### **request\_block\_length**

Direction: Input

Type: Integer

Length of CPRB and the request block in the *request\_block* field. For the APNUM or SERIALNO rules, the maximum length allowed is 5,500 bytes. For the XPNUM rule, the maximum length allowed is 12,000 bytes.

### **request\_block**

Direction: Input

Type: String

The complete command or query request for the target coprocessor, including the CPRB.

### **request\_data\_block\_length**

Direction: Input

Type: Integer

Length of request data block in the *request\_data\_block* field. The maximum length allowed is 6,400 bytes. The length field must be a multiple of 4. For the XPNUM rule, the length must be zero.

### **request\_data\_block**

Direction: Input

Type: String

The data that accompanies the *request\_block* field.

### **reply\_block\_length**

Direction: Input/Output

Type: Integer

Length of CPRB and the reply block in the *reply\_block* field. For the APNUM or SERIALNO rules, the maximum length allowed is 5,500 bytes. For the XPNUM rule, the maximum length allowed is 12,000 bytes. This field is updated on output with the actual length of the *reply\_block* field.

### **reply\_block**

Direction: Output	Type: String
-------------------	--------------

Reply from the target coprocessor. This is the CPRB and reply block that has been processed by the coprocessor.

reply data block length

Direction: Input/Output	Type: Integer
-------------------------	---------------

Length of reply block in the *reply\_data\_block* field. For the APNUM or SERIALNO rules, the maximum length allowed is 6,400 bytes. This field is updated on output with the actual length of the *reply\_data\_block* field. This length field must be a multiple of 4. For the XPNUM rule, the length must be zero. For the XPPOINTS keyword, the minimum length is 2206 bytes. For the ACPOINTS keyword, the minimum length is 3818 bytes for CCF systems, 17469 bytes for non-CCF systems.

reply data block

Direction: Output	Type: String
-------------------	--------------

The data that accompanies the *reply\_block* field.

**masks\_length**

Direction: Input	Type: Integer
------------------	---------------

Length of the reply data being returned in the masks\_data field. The length must be 32 bytes for all requests.

**masks data**

Direction: Output      Type: String

Masks data is returned only when the input rule\_array keyword is PCIMASKS, ACTIVECP XCPMASK, CX2MASK, CX3MASK, CX4MASK, XP4MASK, or QUERYDOM . For all other rule\_array keywords, hex zeroes are returned.

For the QUERYDOM rule, the returned data indicates a bit mask of the actual Crypto CP domains that may be controlled from this logical partition. For all other rules, the first 8 bytes indicate the count of the PCI cards online. The second 8 bytes indicate a bit mask of the actual PCI cards brought online. The third 8 bytes indicate the count of the PCI cards active. The fourth 8 bytes indicate a bit mask of the actual PCI cards that are active. For the ACTIVECP keyword, if the PCI card initialization failed, the appropriate return code and reason code is issued and the *masks\_data* field will contain zeros.

## Usage Notes

The *target\_pci\_coprocessor*, the *target\_pci\_coprocessor\_serial\_number*, the *request\_block*, the *reply\_block*, the *request\_data\_block*, and the *reply\_data\_block*, are recorded in SMF Record Type 82, subtype 16.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 297. PCI Interface required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	
IBM @server zSeries 990	PCI X Cryptographic Coprocessor	
IBM @server zSeries 890	Crypto Express2 Coprocessor	
IBM System z9 EC IBM System z9 BC	PCI X Cryptographic Coprocessor  Crypto Express2 Coprocessor	
IBM System z10 EC IBM System z10 BC	Crypto Express2 Coprocessor  Crypto Express3 Coprocessor	
z196 z114	Crypto Express3 Coprocessor	
IBM zEnterprise EC12	Crypto Express4 Coprocessor	

## PKSC Interface Callable Service (CSFPKSC)

**Restriction:** This service is only supported on the IBM @server zSeries 900.

TKE uses this callable service to send a request to a specific cryptographic module and receive a corresponding response when processing is complete. The service is synchronous. Note that the return and reason codes reflect the success or failure of CSFPKSC's interaction with the cryptographic module rather than the success or failure of the cryptographic module request. The response block contains the results of the cryptographic module request.

This service does not support invocation in AMODE(64).

### Format

```
CALL CSFPKSC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    target_crypto_module,
    request_length,
    request,
    response)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **target\_crypto\_module**

Direction: Input

Type: Integer

Cryptographic module to which this request is directed. Value is 0 or 1.

### **request\_length**

Direction: Input

Type: Integer

Length of request message in the *request* field. The maximum length allowed is 1024 bytes.

### **request**

Direction: Input

Type: String

PKSC command or query request for the target cryptographic module. This is the complete architected command or query for the cryptographic module to process.

### **response**

Direction: Output

Type: String

Area where the PKSC response from the target cryptographic module is returned to the caller. The area returned can be up to 512 bytes.

## Usage Notes

The format and content of the PKSC request and response areas are proprietary IBM hardware information that may be licensed. Customers interested in this information may contact the IBM Director of Licensing. For the address, refer to Notices.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 298. PKSC Interface required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	PCI Cryptographic Coprocessor	
IBM @server zSeries 990 IBM @server zSeries 890		This callable service is not supported.
IBM System z9 EC IBM System z9 BC		This callable service is not supported.
IBM System z10 EC IBM System z10 BC		This callable service is not supported.
z196 z114		This callable service is not supported.
IBM zEnterprise EC12		This callable service is not supported.

---

## Chapter 14. Managing Keys According to the ANSI X9.17 Standard

This topic describes the callable services that support the ANSI X9.17 key management standard:

- “ANSI X9.17 EDC Generate (CSNAEGN and CSNGEGN)”
- “ANSI X9.17 Key Export (CSNAKEX and CSNGKEX)” on page 687
- “ANSI X9.17 Key Import (CSNAKIM and CSNGKIM)” on page 692
- “ANSI X9.17 Key Translate (CSNAKTR and CSNGKTR)” on page 697
- “ANSI X9.17 Transport Key Partial Notarize (CSNATKN and CSNGTKN)” on page 702

These services are only supported on an IBM @server zSeries 900.

These callable services, that are described in other topics of this publication, also support the ANSI X9.17 key management standard:

- “Key Generate (CSNBKGN and CSNEKGN)” on page 142
- “Key Part Import (CSNBKPI and CSNEKPI)” on page 171
- “Key Token Build (CSNBKTB and CSNEKTB)” on page 192

---

### ANSI X9.17 EDC Generate (CSNAEGN and CSNGEGN)

Use the ANSI X9.17 EDC generate callable service to generate an error detection code (EDC) on a text string. The service calculates the EDC by by using a key value of X'0123456789ABCDEF' to generate a MAC on the specified text string, as defined by the ANSI X9.17 standard.

**Restriction:** This service is only supported on an IBM @server zSeries 900.

The callable service name for AMODE(64) invocation is CSNGEGN.

#### Format

```
CALL CSNAEGN(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    text_length,  
    text,  
    chaining_vector,  
    EDC)
```

#### Parameters

**return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.



## ANSI X9.17 EDC Generate

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value must be 0.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. Currently there are no keywords that are defined for this variable, but you must declare the variable. To do so, declare an area of blanks of any length.

### **text\_length**

Direction: Input

Type: Integer

The length of the user-supplied *text* parameter for which the service should calculate the EDC.

### **text**

Direction: Input

Type: String

The application-supplied text field for which the service is to generate the EDC.

### **chaining\_vector**

Direction: Input/Output

Type: String

An 18-byte string that ICSF uses as a system work area. The chaining vector permits data to be chained from one call to another. ICSF ignores the information in this field, but you must declare an 18-byte string.

**EDC**

Direction: Output

Type: String

A 9-byte field where the callable service returns the EDC generated as two groups of four ASCII-encoded hexadecimal characters that are separated by an ASCII space character.

**Usage Notes**

The ANSI X9.17 standard states that for EDC, prior to the service generating the MAC the caller must first edit the input text according to topic 4.3 of ANSI X9.9-1982. It is the caller's responsibility to do the editing prior to calling the ANSI X9.17 EDC generate service. If the supplied text is not a multiple of 8, the service pads the text with X'00' up to a multiple of 8, as specified in ANSI X9.9-1.

To use this service you must have the ANSI system keys installed in the CKDS.

**Required Hardware**

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 299. ANSI X9.17 EDC generate required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990 IBM @server zSeries 890		This callable service is not supported.
IBM System z9 EC IBM System z9 BC		This callable service is not supported.
IBM System z10 EC IBM System z10 BC		This callable service is not supported.
z196 z114		This callable service is not supported.
IBM zEnterprise EC12		This callable service is not supported.

**ANSI X9.17 Key Export (CSNAKEX and CSNGKEX)**

Use the ANSI X9.17 key export callable service to export a DATA key or a pair of DATA keys, along with an ANSI key-encrypting key (AKEK), using the ANSI X9.17 protocol. This service converts a single DATA key, or combines two DATA keys, into a single MAC key. You can use the MAC key in either, or both, the MAC generation, or MAC verification service to authenticate the service message. In addition, this service also supports the export of a CCA IMPORTER or EXPORTER KEK.

## ANSI X9.17 Key Export

If you export only DATA keys, the DATA keys are exported encrypted under the specified transport AKEK. You have the option of applying the ANSI X9.17 key offset or key notarization process to the transport AKEK.

If you export both DATA keys and an AKEK, the DATA keys are exported encrypted under the key-encrypting key that is also being exported. The AKEK is exported encrypted under the specified transport AKEK. You have the option of applying the ANSI X9.17 key offset or key notarization process to the transport AKEK. The ANSI X9.17 key offset process is applied to the source AKEK. Use the CKT keyword to specify whether to use an offset of 0 or 1. Use an offset of 0 when sending the DATA key to a key translation center along with a transport AKEK.

**Note:** You must create the cryptographic service message and maintain the offset counter value that is associated with the AKEK.

**Restriction:** This service is only supported on an IBM @server zSeries 900.

The callable service name for AMODE(64) invocation is CSNGKEX.

## Format

```
CALL CSNAKEX(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    origin_identifier,  
    destination_identifier,  
    source_data_key_1_identifier,  
    source_data_key_2_identifier,  
    source_key_encrypting_key_identifier,  
    transport_key_identifier,  
    outbound_KEK_count,  
    target_data_key_1,  
    target_data_key_2,  
    target_key_encrypting_key,  
    MAC_key_token)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

**exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value can be 0 to 4. If you specify 0, the callable service does not perform either notarization or offset.

**rule\_array**

Direction: Input

Type: String

Zero to four keywords that provide control information to the callable service. See the list of keywords in Table 300. The keywords must be in 8 to 32 bytes of contiguous storage. Left-justify each keyword in its own 8-byte location and pad on the right with blanks. You must specify this parameter even if you specify no keyword.

*Table 300. Keywords for ANSI X9.17 Key Export Rule Array*

Keyword	Meaning
<i>Notarization and Offset Rule (optional with no defaults)</i>	
CPLT-NOT	Complete ANSI X9.17 notarization using the value obtained from the <i>outbound_KEK_count</i> parameter. The transport key that the <i>transport_key_identifier</i> specifies must be partially notarized.
NOTARIZE	Perform notarization processing using the values obtained from the <i>origin_identifier</i> , <i>destination_identifier</i> , and <i>outbound_KEK_count</i> parameters.
OFFSET	Perform ANSI X9.17 key offset processing using the origin counter value obtained from the <i>outbound_KEK_count</i> parameter.
<i>Parity Rule (optional)</i>	
ENFORCE	Stop processing if any source keys do not have odd parity. This is the default value.
IGNORE	Ignore the parity of the source key.
<i>Source Key Rule (optional)</i>	
CCA-EXP	Export a CCA EXPORTER KEK. Requires NOCV keys to be enabled.
CCA-IMP	Export a CCA IMPORTER KEK. Requires NOCV keys to be enabled.
1-KD	Export one DATA key. This is the default parameter.

## ANSI X9.17 Key Export

Table 300. Keywords for ANSI X9.17 Key Export Rule Array (continued)

Keyword	Meaning
1-KD+KK	Export one DATA key and a single-length AKEK.
1-KD+*KK	Export one DATA key and a double-length AKEK.
2-KD	Export two DATA keys.
2-KD+KK	Export two DATA keys and a single-length AKEK.
2-KD+*KK	Export two DATA keys and a double-length AKEK.
<b>Data Key Offset Value (optional)</b>	
CKT	Valid only when a key-encrypting key is being exported along with a DATA key. If this keyword is specified, any DATA keys being exported are encrypted under the key-encrypting key using an offset value of 0. If this keyword is not specified (this is the default), any DATA keys being exported are encrypted under the key-encrypting key using an offset value of 1. The CKT keyword is not valid with CCA-IMP or CCA-EXP keywords.

### **origin\_identifier**

Direction: Input

Type: String

This parameter is valid if the NOTARIZE keyword is specified. It specifies an area that contains a 16-byte string that contains the origin identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. This parameter must be a minimum of four, non-space characters. ICSF ignores this parameter if you specify the OFFSET or CPLT-NOT keyword in the *rule\_array* parameter.

### **destination\_identifier**

Direction: Input

Type: String

This parameter is valid if the NOTARIZE keyword is specified. It specifies an area that contains a 16-byte string. The 16-byte string contains the destination identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. This parameter must be a minimum of four, non-space characters. ICSF ignores this parameter if you specify the OFFSET or CPLT-NOT keyword in the *rule\_array* parameter.

### **source\_data\_key\_1\_identifier**

Direction: Input/Output

Type: String

A 64-byte area that contains an internal token, or the label of a CKDS entry that contains a DATA key. ICSF ignores this field if you specify CCA-EXP or CCA-IMP in the *rule\_array* parameter.

### **source\_data\_key\_2\_identifier**

Direction: Input/Output

Type: String

A 64-byte area that contains an internal token, or the label of a CKDS entry that contains a DATA key. This parameter is valid only if you specify 2-KD,

2-KD+KK, or 2-KD+\*KK as the source key rule keyword on the *rule\_array* parameter. ICSF ignores this parameter if you specify other source key rule keywords, or if you specify CCA-EXP or CCA-IMP in the *rule\_array* parameter.

#### **source\_key\_encrypting\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte area that contains an internal token, or the label of a CKDS entry that contains either an AKEK, a CCA IMPORTER, or a CCA EXPORTER key. If this parameter contains an AKEK, you must specify 1-KD+KK, 2-KD+KK, 1-KD+\*KK, or 2-KD+\*KK for the source key rule on the *rule\_array* parameter. If this parameter contains a CCA IMPORTER or CCA EXPORTER key, you must specify CCA-IMP or CCA-EXP, respectively, for the source key rule on the *rule\_array* parameter. ICSF ignores this field if you specify any other source key rule keywords.

#### **transport\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte area that contains either an internal token or a label that refers to an internal token for an AKEK.

#### **outbound\_KEK\_count**

Direction: Input

Type: String

An 8-byte area that contains an ASCII count that is used in the notarization process. The count is an ASCII character string, left-justified, and padded on the right by ASCII space characters. ICSF interprets a single ASCII space character as a zero counter. The maximum value is 99999999.

#### **target\_data\_key\_1**

Direction: Output

Type: String

A 16-byte area where the exported data key 1 is returned. The enciphered key is an ASCII-encoded hexadecimal string.

#### **target\_data\_key\_2**

Direction: Output

Type: String

A 16-byte area where the exported data key 2 is returned. The enciphered key is an ASCII-encoded hexadecimal string. This key is returned if 2-KD, 2-KD+KK, or 2-KD+\*KK is specified in the *rule\_array* parameter.

#### **target\_key\_encrypting\_key**

Direction: Output

Type: String

If the *rule\_array* parameter specifies 1-KD+KK, 2-KD+KK, 1-KD+\*KK, or 2-KD+\*KK, this parameter specifies a 32-byte area that contains the exported AKEK. If the *rule\_array* parameter specifies CCA-IMP or CCA-EXP, this parameter specifies a 32-byte area that contains the exported key-encrypting key (KEK). The enciphered key is an ASCII-encoded hexadecimal string. If the

## ANSI X9.17 Key Export

*rule\_array* parameter specifies 1-KD+KK or 2-KD+KK, the 16-byte ASCII-encoded output is left-justified in the field and the rest of the field remains unchanged.

### MAC\_key\_token

Direction: Output

Type: String

A 64-byte area that contains an internal token for a MAC key that is intended for use in the MAC generation or MAC verification process. This field is the EXCLUSIVE OR of the two supplied DATA keys when the source key rule in the *rule\_array* parameter specifies 2-KD, 2-KD+KK, or 2-KD+\*KK. When the source key rule specifies 1-KD, the DATA key is converted to a MAC key and returned as an internal token in this field.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You must install the ANSI system keys in the CKDS to use this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 301. ANSI X9.17 key export required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990		This callable service is not supported.
IBM @server zSeries 890		
IBM System z9 EC		This callable service is not supported.
IBM System z9 BC		
IBM System z10 EC		This callable service is not supported.
IBM System z10 BC		
z196		This callable service is not supported.
z114		
IBM zEnterprise EC12		This callable service is not supported.

---

## ANSI X9.17 Key Import (CSNAKIM and CSNGKIM)

Use the ANSI X9.17 key import callable service to import a DATA key or a pair of DATA keys, along with an ANSI key-encrypting key (AKEK), using the ANSI X9.17 protocol. This service converts a single DATA key, or combines two DATA keys, into a single MAC key. The MAC key can be used in either, or both, the MAC

generation or the MAC verification service to authenticate the service message. In addition, this service also supports the import of the KEK to a CCA IMPORTER or EXPORTER KEK, as well as an AKEK.

If you are importing only DATA keys, this service assumes that the DATA keys are encrypted under the specified transport AKEK. You have the option of applying the ANSI X9.17 key offset or key notarization process to the transport AKEK.

If you are importing both DATA keys and an AKEK, this service assumes that the AKEK is encrypted under the specified transport AKEK. This service also assumes that the DATA keys are encrypted under the source AKEK that is also being imported. You have the option of applying the ANSI X9.17 key offset or key notarization process to the transport AKEK. ICSF applies the ANSI X9.17 key offset process to the source AKEK with an offset of 1.

**Note:** You must create the cryptographic service message and maintain the offset counter value that is associated with the AKEK.

**Restriction:** This service is only supported on an IBM @server zSeries 900.

The callable service name for AMODE(64) invocation is CSNGKIM.

## Format

```
CALL CSNAKIM(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    origin_identifier,
    destination_identifier,
    source_data_key_1,
    source_data_key_2,
    source_key_encrypting_key,
    inbound_KEK_count,
    transport_key_identifier,
    target_data_key_1,
    target_data_key_2,
    target_key_encrypting_key,
    MAC_key_token)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are



## ANSI X9.17 Key Import

assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value can be 0 to 3. If you specify 0, ICSF does not perform either notarization or offset.

### **rule\_array**

Direction: Input

Type: String

Zero to three keywords that provide control information to the callable service. See the list of keywords in Table 302. The keywords must be in 8 to 24 bytes of contiguous storage. Each of the keywords must be left-justified in its own 8-byte location and padded on the right with blanks. You must specify this parameter even if you do not specify a keyword.

*Table 302. Keywords for ANSI X9.17 Key Import Rule Array*

Keyword	Meaning
<b>Notarization and Offset Rule (optional with no defaults)</b>	
CPLT-NOT	Complete ANSI X9.17 notarization using the value obtained from the <i>inbound_KEK_count</i> parameter. The transport key that the <i>transport_key_identifier</i> specifies must be partially notarized.
NOTARIZE	Perform notarization processing using the values obtained from the <i>origin_identifier</i> , <i>destination_identifier</i> , and <i>inbound_KEK_count</i> parameters.
OFFSET	Perform ANSI X9.17 key offset processing using the origin counter value obtained from the <i>inbound_KEK_count</i> parameter.
<b>Parity Rule (optional)</b>	
ENFORCE	Stop processing if any source keys do not have odd parity. This is the default value.
IGNORE	Ignore the parity of the source key.
<b>Source Key Rule (optional)</b>	
CCA-EXP	Import a key-encrypting key as a CCA EXPORTER. Requires NOCV keys to be enabled.

Table 302. Keywords for ANSI X9.17 Key Import Rule Array (continued)

Keyword	Meaning
CCA-IMP	Import a key-encrypting key as a CCA IMPORTER. Requires NOCV keys to be enabled.
1-KD	Import one DATA key. This is the default parameter.
1-KD+KK	Import one DATA key and a single-length AKEK.
1-KD+*KK	Import one DATA key and a double-length AKEK.
2-KD	Import two DATA keys.
2-KD+KK	Import two DATA keys and a single-length AKEK.
2-KD+*KK	Import two DATA keys and a double-length AKEK.

**origin\_identifier**

Direction: Input

Type: String

This parameter is valid if you specify the NOTARIZE keyword in the *rule\_array* parameter. It specifies an area that contains a 16-byte string that contains the origin identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. The string must be a minimum of four, non-space characters. This parameter is ignored if the OFFSET or CPLT-NOT keyword is specified.

**destination\_identifier**

Direction: Input

Type: String

This parameter is valid if you specify the NOTARIZE keyword in the *rule\_array* parameter. It specifies an area that contains a 16-byte string that contains the destination identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. It must be a minimum of four non-space characters. This parameter is ignored if the OFFSET or CPLT-NOT keyword is specified.

**source\_data\_key\_1**

Direction: Input

Type: String

A 16-byte area that contains the enciphered DATA key to be imported. You must supply the DATA key as an ASCII-encoded hexadecimal string. The field is ignored if the *rule\_array* parameter specifies CCA-IMP or CCA-EXP.

**source\_data\_key\_2**

Direction: Input

Type: String

A 16-byte area that contains the second enciphered DATA key to be imported. This parameter is valid only if the *rule\_array* parameter specifies KK, or 2-KD+\*KK. You must supply the key as an ASCII-encoded hexadecimal string. This field is ignored if the *rule\_array* parameter specifies other source key rules.

**source\_key\_encrypting\_key**

Direction: Input

Type: String

## ANSI X9.17 Key Import

A 16- or 32-byte area that contains an enciphered AKEK, if the *rule\_array* parameter specifies either 1-KD+KK, 2-KD+KK, 1-KD+\*KK, or 2-KD+\*KK. This parameter specifies a KEK, if the *rule\_array* parameter specifies either CCA-IMP or CCA-EXP. The area is 16 bytes if the *rule\_array* parameter specifies a single-length AKEK (1-KD+KK or 2-KD+KK). The area is 32 bytes if the *rule\_array* parameter specifies a double-length AKEK (1-KD+\*KK or 2-KD+\*KK). You must supply the key as an ASCII-encoded hexadecimal string. This field is ignored if the *rule\_array* parameter specifies 1-KD or 2-KD.

### **inbound\_KEK\_count**

Direction: Input

Type: String

An 8-byte area that contains an ASCII count for use in the notarization process. The count is an ASCII character string, left-justified, and padded on the right by space characters. ICSF interprets a single space character as a zero counter. The maximum value is 99999999.

### **transport\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte area that contains an internal token or a label that refers to an internal token for an AKEK.

### **target\_data\_key\_1**

Direction: Output

Type: String

A 64-byte area where the imported data key 1 is returned as an ICSF internal key token. ICSF does not support the direct import by label.

### **target\_data\_key\_2**

Direction: Output

Type: String

A 64-byte area where the imported data key 2 is returned as an ICSF internal key token. ICSF does not support the direct import by label. This key is returned if 2-KD, 2-KD+KK, or 2-KD+\*KK is specified in the *rule\_array* parameter.

### **target\_key\_encrypting\_key**

Direction: Output

Type: String

A 64-byte area where the imported key-encrypting key is returned as an ICSF internal key token. If the *rule\_array* parameter specifies 1-KD+KK, 1-KD+\*KK, 2-KD+KK, or 2-KD+\*KK, the internal key token contains an AKEK. If the *rule\_array* parameter specifies either CCA-IMP or CCA-EXP, the internal token contains a CCA IMPORTER or a CCA EXPORTER, respectively.

### **MAC\_key\_token**

Direction: Output

Type: String

A 64-byte area that contains an internal token for a MAC key that is intended for use in the MAC generation or MAC verification function. This field is the EXCLUSIVE OR of the two imported DATA keys if the source key rule in the

*rule\_array* parameter specifies 2-KD, 2-KD+KK, or 2-KD+\*KK. If the source key rule in the *rule\_array* parameter specifies 1-KD, ICSF converts the DATA key to a MAC key and returns it as an internal token in this field.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You must install the ANSI system keys in the CKDS to use this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 303. ANSI X9.17 key import required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990 IBM @server zSeries 890		This callable service is not supported.
IBM System z9 EC IBM System z9 BC		This callable service is not supported.
IBM System z10 EC IBM System z10 BC		This callable service is not supported.
z196 z114		This callable service is not supported.
IBM zEnterprise EC12		This callable service is not supported.

## ANSI X9.17 Key Translate (CSNAKTR and CSNGKTR)

Use the ANSI X9.17 key translate callable service to translate a key from encryption under one AKEK to encryption under another AKEK. In a single service call you can translate either one or two encrypted DATA keys, or a single encrypted key-encrypting key. In addition, this service also imports the supplied DATA keys. If the *rule\_array* parameter specifies 2-KD, this service exclusive-ORs the two imported DATA keys and converts the result into a MAC key, which it returns in the *MAC\_key\_token* field. The MAC key is used to perform MAC processing on the service message. If the *rule\_array* specifies keywords 1-KD and 2-KD, ICSF translates only DATA keys. The service uses the inbound transport key-encrypting key to decrypt the DATA keys, and uses the outbound transport key-encrypting key to reencrypt the DATA keys. The service uses the ANSI X9.17 key offset process during decryption or importing. The service can use the ANSI X9.17 notarization process during reencryption or exporting of the DATA keys.

If the *rule\_array* parameter specifies 1-KD+KK or 1-KD+\*KK, the service translates only the AKEK. The service uses the inbound transport key-encrypting key to

## ANSI X9.17 Key Translate

decrypt or import the input AKEK, applying the ANSI X9.17 offset process. The service uses the outbound transport key-encrypting key to reencrypt or export the AKEK, with or without applying the optional ANSI X9.17 notarization process. ICSF uses the inbound key-encrypting key that is being translated to import the supplied DATA key, applying the ANSI X9.17 offset processing only with an offset of 0. The DATA key is imported as previously discussed then converted to a MAC key token and returned in the *MAC\_key\_token* field.

**Restriction:** This service is only supported on an IBM @server zSeries 900.

The callable service name for AMODE(64) invocation is CSNGKTR.

## Format

```
CALL CSNAKTR(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    inbound_KEK_count,  
    inbound_transport_key_identifier,  
    inbound_data_key_1,  
    inbound_data_key_2,  
    inbound_key_encrypting_key,  
    outbound_origin_identifier,  
    outbound_destination_identifier,  
    outbound_KEK_count,  
    outbound_transport_key_identifier,  
    outbound_data_key_1,  
    outbound_data_key_2,  
    outbound_key_encrypting_key,  
    MAC_key_token)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

**exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. The value can be 0 to 3. If you specify 0, the service does not perform notarization or offset.

**rule\_array**

Direction: Input

Type: String

Zero to three keywords that provide control information to the callable service. See the list of keywords in Table 304. The keywords must be in 8 to 24 bytes of contiguous storage. Each of the keywords must be left-justified in its own 8-byte location and padded on the right with blanks. You must specify this parameter even if do not specify any keywords.

Table 304. Keywords for ANSI X9.17 Key Translate Rule Array

Keyword	Meaning
<i>Notarization Rule (optional with no defaults)</i>	
CPLT-NOT	Complete ANSI X9.17 notarization using the value obtained from the <i>outbound_KEK_count</i> parameter. The outbound transport key specified must be partially notarized.
NOTARIZE	Perform notarization processing using the values obtained from the <i>outbound_origin_identifier</i> , the <i>outbound_destination_identifier</i> , and the <i>outbound_KEK_count</i> .
<i>Parity Rule (optional)</i>	
ENFORCE	Stop processing if any source keys do not have odd parity. This is the default value.
IGNORE	Ignore the parity of the source key.
<i>Source Key Rule (optional)</i>	
1-KD	Import and translate one DATA key. This is the default parameter.
1-KD+KK	Import and translate one DATA key and a single-length AKEK.
1-KD+*KK	Import and translate one DATA key and a double-length AKEK.
2-KD	Import and translate two DATA keys.

**inbound\_KEK\_count**

Direction: Input

Type: String

## ANSI X9.17 Key Translate

An 8-byte area that contains an ASCII count for use in the offset process. The count is an ASCII character string, left-justified, and padded on the right by space characters. ICSF interprets a single space character as a zero counter. The maximum value is 99999999.

### **inbound\_transport\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte area that contains either an internal token, or a label that refers to an internal token for an AKEK.

### **inbound\_data\_key\_1**

Direction: Input

Type: String

A 16-byte area that contains the enciphered DATA key that the service is importing and translating. You must specify the DATA key as an ASCII-encoded hexadecimal string.

### **inbound\_data\_key\_2**

Direction: Input

Type: String

A 16-byte area that contains the second enciphered DATA key that the service is importing and translating. This field is valid if the *rule\_array* parameter specifies 2-KD. You must supply the key as an ASCII-encoded hexadecimal string. This field is ignored if the *rule\_array* parameter specifies other source key rules.

### **inbound\_key\_encrypting\_key**

Direction: Input

Type: String

A 16- or 32-byte area that contains an enciphered AKEK that the service is to translate. The area is 16 bytes if the *rule\_array* parameter specifies a source key rule of single-length AKEK. The area is 32 bytes if the source key rule specifies a double-length AKEK (1-KD+\*KK). You must supply the key as an ASCII-encoded hexadecimal string. ICSF ignores this field if the *rule\_array* specifies either 1-KD or 2-KD.

### **outbound\_origin\_identifier**

Direction: Input

Type: String

This parameter is valid if the *rule\_array* parameter specifies a keyword of NOTARIZE. It specifies an area that contains a 16-byte string that contains the origin identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. The string must be a minimum of four non-space characters. ICSF ignores this field if the *rule\_array* parameter specifies a keyword of CPLT-NOT.

### **outbound\_destination\_identifier**

Direction: Input

Type: String

This parameter is valid if the *rule\_array* parameter specifies a keyword of NOTARIZE. It specifies an area that contains a 16-byte string that contains the

destination identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. The string must be a minimum of four non-space characters. This parameter is ignored if the *rule\_array* parameter specifies a keyword of CPLT-NOT.

#### **outbound\_KEK\_count**

Direction: Input

Type: String

An 8-byte area that contains an ASCII count for use in the notarization process. The count is an ASCII character string, left-justified, and padded on the right by space characters. ICSF interprets a single space character as a zero counter. The maximum value is 99999999.

#### **outbound\_transport\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte area that contains either an internal token, or a label that refers to an internal token for an AKEK.

#### **outbound\_data\_key\_1**

Direction: Output

Type: String

A 16-byte area where the service returns the translated data key 1 as an ASCII-encoded hexadecimal string. The service returns the key only if the *rule\_array* specifies 1-KD or 2-KD. ICSF ignores this field if the *rule\_array* parameter specifies either 1-KD+KK or 1-KD+\*KK.

#### **outbound\_data\_key\_2**

Direction: Output

Type: String

A 16-byte area where the service returns the translated data key 2 as an ASCII-encoded hexadecimal string. The service returns the key only if the *rule\_array* parameter specifies 2-KD. ICSF ignores this field if the *rule\_array* parameter specifies 1-KD, 1-KD+KK, or 1-KD+\*KK.

#### **outbound\_key\_encrypting\_key**

Direction: Output

Type: String

A 16- or 32-byte area that contains the enciphered, translated AKEK. The area is 16 bytes if the *rule\_array* parameter specifies a single-length AKEK (1-KD+KK). The area is 32 bytes if the *rule\_array* parameter specifies a double-length AKEK (1-KD+\*KK). The service returns the key as an ASCII-encoded hexadecimal string. ICSF ignores this field if the *rule\_array* parameter specifies either 1-KD or 2-KD.

#### **MAC\_key\_token**

Direction: Output

Type: String

A 64-byte area that contains an internal token for a MAC key that is intended for use in the MAC generation or MAC verification process. This field is the EXCLUSIVE OR of the two imported DATA keys when the *rule\_array*



## ANSI X9.17 Key Translate

parameter specifies 2-KD for the source key rule. If the *rule\_array* parameter specifies 1-KD, the service returns the imported key in this field as an ICSF internal key token.

### Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You must install the ANSI system keys in the CKDS to use this service.

### Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 305. ANSI X9.17 key translate required hardware

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 900	Cryptographic Coprocessor Feature	
IBM @server zSeries 990 IBM @server zSeries 890		This callable service is not supported.
IBM System z9 EC IBM System z9 BC		This callable service is not supported.
IBM System z10 EC IBM System z10 BC		This callable service is not supported.
z196 z114		This callable service is not supported.
IBM zEnterprise EC12		This callable service is not supported.

## ANSI X9.17 Transport Key Partial Notarize (CSNATKN and CSNGTKN)

Use the ANSI X9.17 transport key partial notarize callable service to preprocess an ANSI X9.17 transport key-encrypting key with origin and destination identifiers. ICSF completes the notarization process when you use the partially notarized key in the ANSI X9.17 key export, ANSI X9.17 key import, or ANSI X9.17 key translate services and specify the CPLT-NOT *rule\_array* keyword.

**Note:** You cannot reverse the partial notarization process. If you want to keep the original value of the AKEK, you must record the value.

**Restriction:** This service is only supported on an IBM @server zSeries 900.

The callable service name for AMODE(64) invocation is CSNGTKN.

## Format

```
CALL CSNATKN(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    origin_identifier,
    destination_identifier,
    source_transport_key_identifier,
    target_transport_key_identifier)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Input/Output

Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit\_data* parameter.

### **exit\_data**

Direction: Input/Output

Type: String

The data that is passed to the installation exit.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. Currently no *rule\_array* keywords are defined; thus, this field must be set to 0.

### **rule\_array**

Direction: Input

Type: String

Currently, no *rule\_array* keywords are defined for this service. You must still specify this parameter for possible future use.

## ANSI X9.17 Transport Key Partial Notarize

### **origin\_identifier**

Direction: Input

Type: String

A 16-byte string that contains the origin identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. The string must be a minimum of four non-space characters.

### **destination\_identifier**

Direction: Input

Type: String

A 16-byte string that contains the destination identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. The string must be a minimum of four non-space characters.

### **source\_transport\_key\_identifier**

Direction: Input/Output

Type: String

A 64-byte area that contains either an internal token, or a label of an internal token for an AKEK that permits notarization.

### **target\_transport\_key\_identifier**

Direction: Output

Type: String

A 64-byte area where the internal token of a partially notarized AKEK will be returned. This AKEK cannot be used directly as a notarizing KEK until the notarization process has been completed. To do this, specify CPLT-NOT as the *rule\_array* keyword in any service in which you intend to use this key as a notarizing KEK.

## Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You must install the ANSI system keys in the CKDS to use this service.

## Required Hardware

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 306. ANSI X9.17 transport key partial notarize required hardware*

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 800	Cryptographic Coprocessor Feature	
IBM @server zSeries 900		

Table 306. ANSI X9.17 transport key partial notarize required hardware (continued)

Server	Required cryptographic hardware	Restrictions
IBM @server zSeries 990 IBM @server zSeries 890		This callable service is not supported.
IBM System z9 EC IBM System z9 BC		This callable service is not supported.
IBM System z10 EC IBM System z10 BC		This callable service is not supported.
z196 z114		This callable service is not supported.
IBM zEnterprise EC12		This callable service is not supported.



---

## **Part 3. PKCS #11 Callable Services**



---

## Chapter 15. Using PKCS #11 Tokens and Objects

This topic describes the callable services for creating and maintaining PKCS #11 tokens and objects. ICSF provides a number of callable services to assist you in managing PKCS #11 tokens and maintaining the token data set (TKDS). Services are also provided for generating, using, and managing key objects.

The following callable services are described:

- “PKCS #11 Derive multiple keys (CSFPDMK and CSFPDMK6)”
- “PKCS #11 Derive key (CSFPDVK and CSFPDVK6)” on page 717
- “PKCS #11 Get attribute value (CSFPGAV and CSFPGAV6)” on page 722
- “PKCS #11 Generate key pair (CSFPGKP and CSFPGKP6)” on page 725
- “PKCS #11 Generate secret key (CSFPGSK and CSFPGSK6)” on page 728
- “PKCS #11 Generate HMAC (CSFPHMG and CSFPHMG6)” on page 730
- “PKCS #11 Verify HMAC (CSFPHMV and CSFPHMV6)” on page 734
- “PKCS #11 One-way hash, sign, or verify (CSFPOWH and CSFPOWH6)” on page 737
- “PKCS #11 Private key sign (CSFPPKS and CSFPPKS6)” on page 742
- “PKCS #11 Public key verify (CSFPPKV and CSFPPKV6)” on page 744
- “PKCS #11 Pseudo-random function (CSFPPRF and CSFPPRF6)” on page 747
- “PKCS #11 Set attribute value (CSFPSAV and CSFPSAV6)” on page 749
- “PKCS #11 Secret key decrypt (CSFPSKD and CSFPSKD6)” on page 752
- “PKCS #11 Secret key encrypt (CSFPSKE and CSFPSKE6)” on page 756
- “PKCS #11 Token record create (CSFPTRC and CSFPTRC6)” on page 762
- “PKCS #11 Token record delete (CSFPTRD and CSFPTRD6)” on page 766
- “PKCS #11 Token record list (CSFPTRL and CSFPTRL6)” on page 768
- “PKCS #11 Unwrap key (CSFPUWK and CSFPUWK6)” on page 772
- “PKCS #11 Wrap key (CSFPWPK and CSFPWPK6)” on page 775

As of ICSF FMID HCR7770, a TKDS is no longer required to use the PKCS #11 services. If ICSF is started without a TKDS, however, only the omnipresent token will be available. The omnipresent token supports session objects only. Session objects are objects that do not persist beyond the life of a PKCS #11 session.

---

### PKCS #11 Derive multiple keys (CSFPDMK and CSFPDMK6)

Use the PKCS #11 Derive Multiple Keys callable service to generate multiple secret key objects and protocol dependent keying material from an existing secret key object. This service does not support any recovery methods.

The key handle must be a handle of a PKCS #11 secret key object. The CKA\_DERIVE attribute for the secret key object must be true. The mechanism keyword specified in the rule array indicates what derivation protocol to use. The derive parms list provides additional input/output data. The format of this list is dependent on the protocol being used.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPDMK6.



### Format

```
CALL CSFPDMK(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    attribute_list_length,  
    attribute_list,  
    base_key_handle,  
    parms_list_length,  
    parms_list)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

#### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

#### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

#### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. This value must be 1.

#### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 307. Keywords for derive multiple keys

Keyword	Meaning
Mechanism (required)	
SSL-KM	Use the SSL 3.0 Key and MAC derivation protocol as defined in the PKCS #11 standard as mechanism CKM_SSL3_KEY_AND_MAC_DERIVE.
TLS-KM	Use the TLS 1.0/1.1 Key and MAC derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_KEY_AND_MAC_DERIVE.
IKE1PHA1	<p>Use the IKEv1 phase 1 protocol to derive multiple keys using a previously derived IKE seed key as the base key and a previously derived secret key as an additional key. 3 keys are derived (one derivation, one authentication, and one encryption key).</p> <p>Using IKE terminology, this mechanism performs <math>\{SKEYID_d \mid SKEYID_a \mid SKEYID_e\} = \text{prf}(SKEYID, g^{xy} \mid CKY-I \mid CKY-R)</math> with key expansion for <math>SKEYID_e</math>, if required. (<math>SKEYID_d, a</math> are always the size of the prf output.)</p> <p>Where:</p> <ul style="list-style-type: none"> <li>• <math>CKY-I \mid CKY-R</math> - is the concatenated initiator/responder cookie string</li> <li>• <math>SKEYID</math> - is the base key</li> <li>• <math>g^{xy}</math> - is the additional key</li> <li>• <math>SKEYID_{d,a,e}</math> - are the to-be-derived derivation, authentication and encryption keys</li> </ul>
IKE2PHA1	<p>Use the IKEv2 phase 1 (SA) protocol to derive multiple keys using a previously derived IKE seed key as the base key. 7 keys are derived (one derivation, two authentication, two encryption, and two peer authentication keys).</p> <p>Using IKE terminology, this mechanism performs <math>\{SK_d \mid SK_{ai} \mid SK_{ar} \mid SK_{ei} \mid SK_{er} \mid SK_{pi} \mid SK_{pr}\} = \text{prf}+(SKEYSEED, Ni \mid Nr \mid SPIi \mid SPIr)</math>.</p> <p>Where:</p> <ul style="list-style-type: none"> <li>• <math>Ni \mid Nr \mid SPIi \mid SPIr</math> - is the concatenated initiator/responder nonce and Security Parameter Index string</li> <li>• <math>SKEYSEED</math> - is the base key</li> <li>• <math>SK_{d,ai,ar,ei,er,pi,pr}</math> - are the to-be-derived derivation, initiator authentication, responder authentication, initiator encryption, responder encryption, initiator peer authentication, and responder peer authentication keys</li> </ul>

## PKCS #11 Derive multiple keys

Table 307. Keywords for derive multiple keys (continued)

Keyword	Meaning
IKE1PHA2	<p>Use the IKEv1 phase 2 (CHILD SA) protocol to derive multiple keys and salt values using a previously derived IKE derivation key as the base key and a previously derived secret key as an additional key (optional). The derivation produces one of the following key sets:</p> <ul style="list-style-type: none"> <li>• One authentication key</li> <li>• One GMAC key plus salt value</li> <li>• One authentication key plus one encryption key</li> <li>• One GCM key plus a salt value</li> </ul> <p>Up to two such sets are produced, one for the sender and one for the receiver.</p> <p>Using IKE terminology, this mechanism performs <math>KEYMAT = \text{prf}(SKEYID\_d, [g^{xy} \mid] \text{protocol} \mid SPI \mid Ni\_b \mid Nr\_b)</math>, done in two passes – once for the sender and once for the receiver.</p> <p>Where:</p> <ul style="list-style-type: none"> <li>• <math>\text{protocol} \mid SPI \mid Ni\_b \mid Nr\_b</math> - is the concatenated Protocol, Security Parameter Index, and initiator/responder nonce string</li> <li>• <math>SKEYID\_d</math> - is the base key</li> <li>• <math>g^{xy}</math> - is the optional additional key</li> <li>• <math>KEYMAT</math> - is the generated key material which is partitioned into the key set</li> </ul>
IKE2PHA2	<p>Use the IKEv2 phase 2 protocol to derive multiple keys and salt values using a previously derived IKE derivation key as the base key and a previously derived secret key as an additional key (optional). The derivation produces one of the following key sets:</p> <ul style="list-style-type: none"> <li>• One authentication key</li> <li>• One GMAC key plus salt value</li> <li>• One authentication key plus one encryption key</li> <li>• One GCM key plus a salt value</li> </ul> <p>Two such sets are produced, one for the initiator and one for the responder.</p> <p>Using IKE terminology, this mechanism performs <math>KEYMAT = \text{prf}(SK\_d, [g^{ir} \mid] Ni \mid Nr)</math>.</p> <p>Where:</p> <ul style="list-style-type: none"> <li>• <math>Ni \mid Nr</math> - is the concatenated initiator/responder nonce string</li> <li>• <math>SK\_d</math> - is the base key</li> <li>• <math>g^{ir}</math> - is the optional additional key</li> <li>• <math>KEYMAT</math> - is the generated key material which is partitioned into the key set</li> </ul>

### attribute\_list\_length

Direction: Input

Type: Integer

The length of the attributes supplied in the *attribute\_list* parameter in bytes.  
The maximum value for this field is 32752.

### attribute\_list

Direction: Input

Type: String

List of attributes for the derived secret key object. See “Attribute List” on page 98 for the format of an *attribute\_list*.

## **base\_key\_handle**

Direction: Input

Type: String

The 44-byte handle of the base key object. See “Handles” on page 99 for the format of a *key\_handle*.

## **parms\_list\_length**

Direction: Input

Type: Integer

The length of the parameters supplied in the *parms\_list* parameter in bytes.

## **parms\_list**

Direction: Input/Output

Type: String

The protocol specific parameters. This field has a varying format depending on the mechanism specified:

Table 308. *parms\_list* parameter format for SSL-KM and TLS-KM mechanisms

Offset	Length in bytes	Direction	Description
0	1	Input	Boolean indicating if “export” processing is required. Any value other than x'00' means yes
1	3	Not applicable	reserved
4	4	Input	length in bytes of the client's random data (x) ), where 1 <= length <= 32
8	4	Input	length in bytes of the server's random data (y) ), where 1 <= length <= 32
12	4	Input	size of MAC to be generated in bits, where 8 <= size <= 384, in multiples of 8
16	4	Input	size of key to be generated in bits, Must match a supported size for the key type specified in the attribute list. Zero if no encryption keys are to be generated.
20	4	Input	size of IV to be generated in bits (v), where 0<= size <= 128, in multiples of 8. Must be zero if no encryption keys are to be generated.
24	44	Output	handle of client MAC secret object created
68	44	Output	handle of server MAC secret object created
112	44	Output	handle of client key object created
156	44	Output	handle of server key object created
200	x	Input	client's random data
200+x	y	Input	server's random data
200+x+y	v/8	Output	client's IV
200+x+y+v/8	v/8	Output	server's IV

## PKCS #11 Derive multiple keys

Table 309. *parms\_list* parameter format for IKE1PHA1 mechanism

Offset	Length in bytes	Direction	Description
0	1	Input	IKE version code. Must be x'01'
1	1	Input	PRF function code x'01' = HMAC_MD5, x'02' = HMAC_SHA1, x'04' = HMAC_SHA256, x'05' = SHA384, and x'06' = SHA512
2	4	Input	reserved
6	2	Input	length of to-be-derived encryption key, SKEYID_e
8	44	Input	Key handle of additional key
52	16	Input	Concatenated cookie string
68	44	Output	SKEYID_d key handle
112	44	Output	SKEYID_a key handle
156	44	Output	SKEYID_e key handle

Table 310. *parms\_list* parameter format for IKE2PHA1 mechanism

Offset	Length in bytes	Direction	Description
0	1	Input	IKE version code. Must be x'02'
1	1	Input	PRF function code x'01' = HMAC_MD5, x'02' = HMAC_SHA1, x'04' = HMAC_SHA256, x'05' = SHA384, and x'06' = SHA512
2	2	Input	length of to-be-derived derivation key, SK_d
4	2	Input	length of a single to-be-derived authentication key, SK_a
6	2	Input	length of a single to-be-derived encryption key, SK_e
8	2	Input	length of a single to-be-derived peer authentication key, SK_p
10	2	Input	Concatenated nonce, SPI string length (n), where 24 <= n <= 520
12	44	Output	SKEYID_d key handle
56	44	Output	Initiator SKEYID_a key handle
100	44	Output	Responder SKEYID_a key handle
144	44	Output	Initiator SKEYID_e key handle
188	44	Output	Responder SKEYID_e key handle
232	44	Output	Initiator SKEYID_p key handle
276	44	Output	Responder SKEYID_p key handle
320	n	Input	Concatenated nonce, SPI string

Table 311. *parms\_list* parameter format for IKE1PHA2 and IKE2PHA2 mechanisms

Offset	Length in bytes	Direction	Description
0	1	Input	IKE version code. Must be x'01' for IKE1PHA2, x'02' for IKE2PHA2
1	1	Input	PRF function code x'01' = HMAC_MD5, x'02' = HMAC_SHA1, x'04' = HMAC_SHA256, x'05' = SHA384, and x'06' = SHA512

Table 311. *parms\_list* parameter format for IKE1PHA2 and IKE2PHA2 mechanisms (continued)

Offset	Length in bytes	Direction	Description
2	2	Input	length of to-be-derived salts (s), where $0 \leq s \leq 4$ . Zero if salts are not to be derived
4	2	Input	length of to-be-derived authentication keys. Zero if authentication keys are not to be derived
6	2	Input	length of to-be-derived encryption, GMAC, or GCM keys. Zero if no such keys are to be derived
8	2	Input	First pass parameter string length (n) <ul style="list-style-type: none"> <li>For IKE1PHA2 – Receiver concatenated Protocol, Security Parameter Index, and initiator/responder nonce string length, where <math>25 \leq n \leq 525</math></li> <li>For IKE2PHA2 – Concatenated initiator/responder nonce string length, where <math>16 \leq n \leq 512</math>.</li> </ul>
10	2	Input	Second pass parameter string length (m) <ul style="list-style-type: none"> <li>For IKE1PHA2 – Sender concatenated Protocol, Security Parameter Index, and initiator/responder nonce string length, where <math>25 \leq m \leq 525</math>. Zero if second pass is to be skipped</li> <li>For IKE2PHA2 – Not used. Must be zero</li> </ul>
12	44	Input	Key handle of additional key. Fill with binary zeros if n/a
56	44	Output	Initiator (sender) authentication key handle
100	44	Output	Responder (receiver) authentication key handle
144	44	Output	Initiator (sender) encryption, GMAC, or GCM key handle
188	44	Output	Responder (receiver) encryption, GMAC, or GCM key handle
232	n	Input	First pass parameter string
232+n	m	Input	Second pass parameter string
232+n+m	s	Output	Initiator (sender) salt
232+n+m+s	s	Output	Responder (receiver) salt

## Authorization

There are multiple keys involved in this service — one or two base keys and the target keys (the new keys created from the base key).

- To use a base key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To use a base key that is a private object, the caller must have USER (READ) authority (user access).
- To derive a target key that is a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.
- To derive a target key that is a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

### Usage Notes

The service does not support secure keys.

Key derivation functions are performed in software.

For the SSL-KM and TLS-KM mechanisms, an attribute list is required if encryption keys are to be generated.

For the IKE1PHA1, IKE2PHA1, IKE1PHA2, and IKE2PHA2 mechanisms, the following attribute rules apply to the derived keys:

- Derivation keys will have the following attributes which may not be overridden by other values in the attribute list:
  - CKA\_CLASS=CKO\_SECRET\_KEY
  - CKA\_KEY\_TYPE=CKK\_GENERIC\_SECRET
  - CKA\_DERIVE=TRUE
  - CKA\_VALUE\_LEN=*as specified in the parms list*
- Authentication keys will have the following attributes which may not be overridden by other values in the attribute list:
  - CKA\_CLASS=CKO\_SECRET\_KEY
  - CKA\_KEY\_TYPE=CKK\_GENERIC\_SECRET
  - CKA\_SIGN=TRUE=TRUE
  - CKA\_VERIFY=TRUE=TRUE
  - CKA\_VALUE\_LEN= *as specified in the parms list*
- Encryption, GMAC, and GCM keys will be typed according to information found in the attribute list. However, they will have the following attributes which may not be overridden by other values in the attribute list:
  - CKA\_CLASS=CKO\_SECRET\_KEY
  - For key types other than CKK\_DES, CKK\_DES2, and CKK\_DES3, CKA\_VALUE\_LEN= *as specified in the parms list*
- All key types will inherit the values of the CKA\_SENSITIVE, CKA\_ALWAYS\_SENSITIVE, CKA\_EXTRACTABLE, and CKA\_NEVER\_EXTRACTABLE attributes from the base key. These may not be overridden by other values in the attribute list. If an additional key is specified, its values will be applied after setting the base key values as follows:
  - If the additional key has CKA\_SENSITIVE=TRUE, so will the derived key(s)
  - If the additional key has CKA\_EXTRACTABLE=FALSE, so will the derived keys(s)
  - If the additional key has CKA\_ALWAYS\_SENSITIVE=FALSE, so will the derived keys(s)
  - If the additional key has CKA\_NEVER\_EXTRACTABLE=FALSE, so will the derived keys(s)
- If encryption, GMAC, or GCM keys are to be derived, an attribute list is required for the key typing information. Otherwise, it is optional. For all keys, other applicable secret key attributes may be specified in the attribute list. Any attribute not specified will be assigned the default value normally assigned to a newly created secret key.

For the IKE1PHA1, IKE1PHA2, and IKE2PHA2 mechanisms, the additional key must be a secret key (CKA\_CLASS=CKO\_SECRET\_KEY) capable of performing key derivation (CKA\_DERIVE=TRUE). It must also be contained in the same PKCS #11 token as the base key.

The IKE1PHA1, IKE2PHA1, IKE1PHA2, and IKE2PHA2 mechanisms have the following limitations if the operation is FIPS 140 restricted:

- The MD5 PRF may not be specified.
- The length of the base key must be at least half the length of the output of the PRF function.

---

## PKCS #11 Derive key (CSFPDVK and CSFPDVK6)

Use the PKCS #11 Derive Key callable service to generate a new secret key object from an existing key object. This service does not support any recovery methods.

The deriving key handle must be a handle of an existing PKCS #11 key object. The CKA\_DERIVE attribute for this object must be true. The mechanism keyword specified in the rule array indicates what derivation protocol to use. The derive parms list provides additional input data. The format of this list is dependent on the protocol being used.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPDVK6.

### Format

```
CALL CSFPDVK(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    attribute_list_length,
    attribute_list,
    base_key_handle,
    parms_list_length,
    parms_list,
    target_key_handle)
```

### Parameters

#### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### reason\_code

Direction: Output

Type: Integer



## PKCS #11 Derive key

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. This value must be 1.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

*Table 312. Keywords for derive key*

Keyword	Meaning
Mechanism (required)	
PKCS-DH	Use the Diffie-Hellman PKCS derivation protocol as defined in the PKCS #11 standard as mechanism CKM_DH_PKCS_DERIVE.
SSL-MS	Use the SSL 3.0 Master Secret derivation protocol as defined in the PKCS #11 standard as mechanism CKM_SSL3_MASTER_KEY_DERIVE. The SSL protocol version is also returned. The base key must have been generated according to the rules for SSL 3.0
SSL-MSDH	Use the SSL 3.0 Master Secret for Diffie-Hellman derivation protocol as defined in the PKCS #11 standard as mechanism CKM_SSL3_MASTER_KEY_DERIVE_DH.
TLS-MS	Use the TLS Master Secret derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_MASTER_KEY_DERIVE. The base key must have been generated according to the rules for TLS 1.0 or TLS 1.1
TLS-MSDH	Use the TLS Master Secret for Diffie-Hellman derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_MASTER_KEY_DERIVE_DH.
EC-DH	Use the Elliptic Curve Diffie-Hellman derivation protocol as defined in the PKCS #11 standard as mechanism CKM_ECDH1_DERIVE

Table 312. Keywords for derive key (continued)

Keyword	Meaning
IKESEED	<p>Use the IKEv1 or IKEv2 initial seeding protocol to derive a seed key using a previously derived secret key as the base key.</p> <p>Using IKE terminology, this mechanism performs either <math>SKEYID = prf(Ni\_b \parallel Nr\_b, g^{xy})</math> for IKEv1 or <math>SKEYSEED = prf(Ni \parallel Nr, g^{ir})</math> for IKEv2.</p> <p>Where:</p> <ul style="list-style-type: none"> <li><math>Ni\_b \parallel Nr\_b</math> or <math>Ni \parallel Nr</math> - is the concatenated initiator/responder nonce string</li> <li><math>g^{xy}</math> or <math>g^{ir}</math> - is the base key</li> </ul>
IKESHARE	<p>Use the IKEv1 initial seeding protocol to derive a seed key using a pre-shared secret key as the base key.</p> <p>Using IKE terminology, this mechanism performs <math>SKEYID = prf(pre-shared-key, Ni\_b \parallel Nr\_b)</math>.</p> <p>Where:</p> <ul style="list-style-type: none"> <li><math>Ni\_b \parallel Nr\_b</math> - is the concatenated initiator/responder nonce string</li> <li><math>pre-shared-key</math> - is the base key</li> </ul>
IKEREKEY	<p>Use the IKEv2 rekeying protocol to derive a new seed key using a previously derived IKE derivation key as the base key and a previously derived secret key as an additional key.</p> <p>Using IKE terminology, this mechanism performs <math>SKEYSEED = prf(SK\_d, g^{ir} \parallel Ni \parallel Nr)</math>.</p> <p>Where:</p> <ul style="list-style-type: none"> <li><math>Ni \parallel Nr</math> - is the concatenated initiator/responder nonce string</li> <li><math>SK\_d</math> - is the base key</li> <li><math>g^{ir}</math> - is the additional key</li> </ul>

### attribute\_list\_length

Direction: Input

Type: Integer

The length of the attributes supplied in the *attribute\_list* parameter in bytes. The maximum value for this field is 32750.

### attribute\_list

Direction: Input

Type: String

List of attributes for the derived secret key object. See “Attribute List” on page 98 for the format of an *attribute\_list*.

### base\_key\_handle

Direction: Input

Type: String

The 44-byte handle of the source key object. See “Handles” on page 99 for the format of a *base\_key\_handle*.

### parms\_list\_length

Direction: Input

Type: Integer

## PKCS #11 Derive key

The length of the parameters supplied in the *parms\_list* parameter in bytes.

### **parms\_list**

Direction: Input/Output

Type: String

The protocol specific parameters. This field has a varying format depending on the mechanism specified:

*Table 313. parms\_list parameter format for PKCS-DH mechanism*

Offset	Length in bytes	Direction	Description
0	4	Input	length in bytes of the other party's public value, where $64 \leq \text{length} \leq 256$
4	$\leq 256$	Input	binary value representing the other party's public value.

*Table 314. parms\_list parameter format for SSL-MS, SSL-MSDH, TLS-MS, and TLS-MSDH mechanisms*

Offset	Length in bytes	Direction	Description
0	2	Output	SSL protocol version returned for SSL-MS and TLS-MS only. For the other protocols, this field is left unchanged.
2	2	not applicable	reserved
4	4	Input	length in bytes of the client's random data (x), where $1 \leq \text{length} \leq 32$
8	4	Input	length in bytes of the server's random data (y), where $1 \leq \text{length} \leq 32$
12	x	Input	client's random data
12+x	y	Input	server's random data

*Table 315. parms\_list parameter format for EC-DH mechanism*

Offset	Length in bytes	Direction	Description
0	1	Input	KDF function code, x'01' = NULL; x'02' = SHA1. x'05' = SHA224, x'06' = SHA256, x'07' = SHA384, and x'08' = SHA512
1	3	not applicable	reserved
4	4	Input	length in bytes of the optional data shared between the two parties. A zero length means no shared data. For the NULL KDF the length must be zero. Otherwise, the maximum shared data length 2147483647.
8	8	Input	64-bit address of the data shared between the two parties. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length is zero.

**Table 315. *parms\_list* parameter format for EC-DH mechanism (continued)**

Offset	Length in bytes	Direction	Description
16	4	Input	length in bytes of the other party's public value (x). This length is dependent on the curve type/size of the base key and on whether the value is DER encoded or not:  secp192r1 – 49 (51 w/DER) secp224r1 – 57 (59 w/DER) secp256r1 – 65 (67 w/DER) secp384r1 – 97 (99 w/DER) secp521r1 – 133 (136 w/DER)  brainpoolP160r1 – 41 (43 w/DER) brainpoolP192r1 – 49 (51 w/DER) brainpoolP224r1 – 57 (59 w/DER) brainpoolP256r1 – 65 (67 w/DER) brainpoolP320r1 – 81 (83 w/DER) brainpoolP384r1 – 97 (99 w/DER) brainpoolP512r1 – 129 (132 w/DER)
20	x<=136	Input	binary value representing the other party's public value with or without DER encoding.

**Table 316. *parms\_list* parameter format for IKESSEED, IKESHARE, and IKEREKEY mechanisms**

Offset	Length in bytes	Direction	Description
0	1	Input	IKE version code. Must be x'01' for IKESHARE, x'02' for IKEREKEY, x'01' or x'02' for IKESSEED
1	1	Input	PRF function code x'01' = HMAC_MD5, x'02' = HMAC_SHA1, x'04' = HMAC_SHA256, x'05' = SHA384, and x'06' = SHA512
2	2	Input	Length of concatenated initiator/responder nonce string (n), where 16 <= n <= 512
4	44	Input	Key handle of additional key - required for IKEREKEY. Ignored for the other mechanisms.
48	n	Input	Concatenated initiator/responder nonce string

### **target\_key\_handle**

Direction: Output

Type: String

Upon successful completion, the 44-byte handle of the secret key object that was derived.

## **Authorization**

There are multiple keys involved in this service — one or two base keys and the target key (the new key created from the base key).

- To use a base key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To use a base key that is a private object, the caller must have USER (READ) authority (user access).
- To derive a target key that is a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.

## PKCS #11 Derive key

- To derive a target key that is a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

### Usage Notes

The service does not support secure keys.

Derivation of the EC-DH shared secret "Z" may be performed in hardware or software. All other key derivation operations are performed in software.

Key derivation functions are performed in software.

For the IKSEED, IKESHARE, and IKEREKEY mechanisms, the following attribute rules apply to the derived key:

- The key will have the following attributes which may not be overridden by other values in the attribute list:
  - CKA\_CLASS=CKO\_SECRET\_KEY
  - CKA\_KEY\_TYPE=CKK\_GENERIC\_SECRET
  - CKA\_DERIVE=TRUE
  - CKA\_VALUE\_LEN=*length of the output of the PRF function*
- Other applicable secret key attributes may be specified in the attribute list. However, an attribute list is not required. Any attribute not specified will be assigned the default value normally assigned to a newly created secret key. In particular, CKA\_SENSITIVE defaults to FALSE and CKA\_EXTRACTABLE defaults to TRUE.
- CKA\_ALWAYS\_SENSITIVE is set to FALSE if the CKA\_ALWAYS\_SENSITIVE attribute from the base key is FALSE. Otherwise it is set equal to the value of the CKA\_SENSITIVE attribute assigned to the derived key.
- CKA\_NEVER\_EXTRACTABLE is set to FALSE if the CKA\_NEVER\_EXTRACTABLE attribute from the base key is FALSE. Otherwise it is set opposite to the value of the CKA\_EXTRACTABLE attribute assigned to the derived key.

For the IKEREKEY mechanism, the additional key must be a secret key (CKA\_CLASS=CKO\_SECRET\_KEY) capable of performing key derivation (CKA\_DERIVE=TRUE). It must also be contained in the same PKCS #11 token as the base key.

For the IKSEED, IKESHARE, and IKEREKEY mechanisms, the MD5 PRF may not be specified if the operation is FIPS 140 restricted.

For the IKESHARE and IKEREKEY mechanisms, the length of the base key must be at least half the length of the output of the PRF function if the operation is FIPS 140 restricted.

For the IKSEED mechanism, the length of the concatenated initiator/responder nonce value must be at least half the length of the output of the PRF function if the operation is FIPS 140 restricted.

---

## PKCS #11 Get attribute value (CSFPGAV and CSFPGAV6)

Use the get attribute value callable service (CSFPGAV) to retrieve the attributes of an object.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPGAV6.

## Format

```
CALL CSFPGAV(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    handle,
    rule_array_count,
    rule_array,
    attribute_list_length,
    attribute_list)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **handle**

Direction: Input

Type: String

The 44-byte handle of the object. See “Handles” on page 99 for the format of a *handle*.

### **rule\_array\_count**

## PKCS #11 Get attribute value

Direction: Input

Type: Integer

The number of keywords supplied in the *rule\_array* parameter. This value must be 0.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

### **attribute\_list\_length**

Direction: Input/Output

Type: Integer

On input, the length of the *attribute\_list* parameter in bytes.

On output, the length of the *attribute\_list* parameter in bytes. If the length supplied on input is insufficient to hold all attributes, the length on output is set to the minimum length required.

### **attribute\_list**

Direction: Output

Type: String

A list of object attributes.

See "Attribute List" on page 98 for the format of an *attribute\_list*.

## Authorization

The token authorization required and the amount of attribute information returned is dependent on the values of the attributes the object possesses.

The authority to retrieve the non-sensitive attributes is as follows:

- For a public object - any authority to the token (USER (READ) or SO (READ))
- For a private object - USER (READ) or SO (CONTROL)

If the caller is not authorized to retrieve the non-sensitive attributes, the service fails.

If the caller is authorized to retrieve the non-sensitive attributes and the object does not possess any sensitive attributes, the service returns all the object's attributes.

If the caller is authorized to retrieve the non-sensitive attributes and the object does possess sensitive attributes, processing is as defined in this table:

*Table 317. Get attribute value processing for objects possessing sensitive attributes*

Object	PKCS #11 role authority	CKA_SENSITIVE	CKA_EXTRACTABLE	Attributes returned
Public	USER (READ) or SO (READ)	True	True or False	Non-sensitive only
Private	USER (READ) or SO (CONTROL)	True	True or False	Non-sensitive only
Public	USER (READ) or SO (READ)	False	False	Non-sensitive only
Private	USER (READ) or SO (CONTROL)	False	False	Non-sensitive only
Public	USER (READ) or SO (READ)	False	True	Sensitive and non-sensitive
Private	SO (CONTROL)	False	True	Non-sensitive only
Private	USER (READ)	False	True	Sensitive and non-sensitive

**Note:**

- Session and token objects require the same authority.
- The sensitive attributes are as follows:
  - CKA\_VALUE for a secret key, Elliptic Curve private key, DSA private key, or Diffie-Hellman private key object.
  - CKA\_PRIVATE\_EXPONENT, CKA\_PRIME\_1, CKA\_PRIME\_2, CKA\_EXPONENT\_1, CKA\_EXPONENT\_2, and CKA\_COEFFICIENT for a private key object.
- See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the SO and User PKCS #11 roles.

## Usage Notes

If the object is marked sensitive or not extractable, the sensitive attributes are not returned.

If the caller is authorized to list the non-sensitive attributes of an object, but not the sensitive ones, the sensitive attributes are not returned.

If the caller is not authorized to list the non-sensitive attributes of the object, the service fails.

---

## PKCS #11 Generate key pair (CSFPGKP and CSFPGKP6)

Use the generate key pair callable service to generate an RSA, DSA, Elliptic Curve, or Diffie-Hellman key pair. New token or session objects are created to hold the key pair.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPGKP6.



## Format

```
CALL CSFPGKP(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    token_handle,
    rule_array_count,
    rule_array,
    public_key_attribute_list_length,
    public_key_attribute_list,
    public_key_object_handle,
    private_key_attribute_list_length,
    private_key_attribute_list,
    private_key_object_handle)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **token\_handle**

Direction: Input

Type: String

The 44-byte handle of the token of the key objects. See “Handles” on page 99 for the format of a *token\_handle*.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array\_parameter*. This value must be 0.

### **rule\_array**

Direction: Input Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage

### **public\_key\_attribute\_list\_length**

Direction: Input Type: Integer

The length of the attributes supplied in the *public\_key\_attribute* list parameter in bytes.

### **public\_key\_attribute\_list**

Direction: Input Type: String

List of attributes for the public key object. The maximum value for this field is 32750. See "Attribute List" on page 98 for the format of a *public\_key\_attribute\_list*.

### **public\_key\_object\_handle**

Direction: Output Type: String

The 44-byte handle of the new public key object.

### **private\_key\_attribute\_list\_length**

Direction: Input Type: Integer

The length of the attributes supplied in the *private\_key\_attribute\_list* parameter in bytes.

### **private\_key\_attribute\_list**

Direction: Input Type: String

List of attributes for the private key object. The maximum value for this field is 32750. See "Attribute List" on page 98 for the format of a *private\_key\_attribute\_list*.

### **private\_key\_object\_handle**

Direction: Output Type: String

The 44-byte handle of the new private key object.

## **Authorization**

To generate a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.

To generate a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

## **Usage Notes**

The type of key pair generated is determined by the key type attributes in the *public\_key\_attributes\_list* and *private\_key\_attributes\_list* parameters.

Key pair generation may be done in hardware or software.

## PKCS #11 Generate secret key (CSFPGSK and CSFPGSK6)

Use the generate secret key callable service to generate a secret key or set of domain parameters. A new token or session object is created to hold the information.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPGSK6.

### Format

```
CALL CSFPGSK(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    handle,
    rule_array_count,
    rule_array,
    attribute_list_length,
    attribute_list,
    parms_list_length,
    parms_list)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

#### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

#### **handle**

Direction: Input/Output

Type: String

On input, the 44-byte handle of the token. On output, the 44-byte handle of the new secret key or domain parameters object. See “Handles” on page 99 for the format of a *handle*.

**rule\_array\_count**

The number of keywords you supplied in the *rule\_array* parameter. This value must be 1.

**rule\_array**

Direction: Input Type: String

Keywords that provide control information to the callable service.

Table 318. Keywords for generate secret key

Keyword	Meaning
Mechanism (One of the following must be specified)	
SSL	Generate a generic secret key object where the client is using SSL (for CKM_SSL3_PRE_MASTER_KEY_GEN)
TLS	Generate a generic secret key object where the client is using TLS (for CKM_TLS_PRE_MASTER_KEY_GEN)
KEY	Generate a secret key object according to the key type attribute in the <i>attribute_list</i> parameter (for CKM_GENERIC_SECRET_KEY_GEN, CKM_DES_KEY_GEN, CKM_DES2_KEY_GEN, CKM_DES3_KEY_GEN, CKM_AES_KEY_GEN, CKM_RC4_KEY_GEN, and CKM_BLOWFISH_KEY_GEN)
PBEKEY	Generate password-based encryption key material and a secret key object according to the key type attribute in the <i>attribute_list</i> parameter (for CKM_PBE_SHA1_DES3_EDE_CBC only)
PARMS	Generate a domain parameters object according to the key type attribute in the <i>attribute_list</i> parameter (for CKM_DSA_PARAMETER_GEN and CKM_DH_PKCS_PARAMETER_GEN)

**attribute\_list\_length**

Direction: Input Type: Integer

The length of the attributes supplied in the *attribute\_list* parameter in bytes. The maximum value for this field is 32750.

**attribute\_list**

Direction: Input Type: String

List of attributes for the secret key object. See “Attribute List” on page 98 for the format of an *attribute\_list* .

**parms\_list\_length**

Direction: Input Type: Integer

The length of the parameters supplied in the *parms\_list* parameter in bytes.

**parms\_list**

Direction: Input/Output Type: String

## PKCS #11 Generate secret key

The protocol specific parameters. This field has a varying format depending on the mechanism specified:

Table 319. *parms\_list* parameter format for SSL and TLS mechanism

Offset	Length in bytes	Direction	Description
0	2	input	SSL or TLS version number in binary, e.g., for version 3.01 this would be x'0301'

Table 320. *parms\_list* parameter format for PBEKEY mechanism

Offset	Length in bytes	Direction	Description
0	2	input	length in bytes of the password (p), where $1 \leq p \leq 128$
2	2	input	length in bytes of the salt (s), where $1 \leq s \leq 128$
4	4	input	number of iterations required (n), where $1 \leq n \leq 65,536$
8	8	output	8-byte IV returned
16	p	input	password
16+p	s	input	salt

For the KEY and PARMS mechanisms, there are no parameters. The *parms\_list\_length* parameter must be set to zero for these mechanisms.

## Authorization

To generate a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.

To generate a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

## Usage Notes

Domain parameters are generated in software.

BLOWFISH, RC4, TLS and SSL key generation is performed in software. All other key generation may be performed in hardware or software.

Rule PBEKEY requires a z890/990 or later machine type.

---

## PKCS #11 Generate HMAC (CSFPHMG and CSFPHMG6)

Use the PKCS #11 Generate HMAC callable service to generate a hashed message authentication code (MAC). This service does not support any recovery methods.

The key handle must be a handle of a PKCS #11 generic secret key object. The mechanism keyword specified in the rule array indicates the hash algorithm to use. The CKA\_SIGN attribute for the secret key object must be true.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPHMG6.

## Format

```
CALL CSFPHMG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    text_length,
    text,
    text_id,
    chain_data_length,
    chain_data,
    key_handle,
    hmac_length,
    hmac )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. This value must be 1 or 2.

### **rule\_array**

Direction: Input

Type: String

## PKCS #11 Generate HMAC

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 321. Keywords for generate HMAC

Keyword	Meaning
Mechanism (required)	
MD5	Generate an HMAC. Use MD5 hashing. Output returned in the <i>hmac</i> parameter is 16 bytes in length.
SHA-1	Generate an HMAC. Use SHA-1 hashing. Output returned in the <i>hmac</i> parameter is 20 bytes in length.
SHA-224	Generate an HMAC. Use SHA-224 hashing. Output returned in the <i>hmac</i> parameter is 28 bytes in length.
SHA-256	Generate an HMAC. Use SHA-256 hashing. Output returned in the <i>hmac</i> parameter is 32 bytes in length.
SHA-384	Generate an HMAC. Use SHA-384 hashing. Output returned in the <i>hmac</i> parameter is 48 bytes in length.
SHA-512	Generate an HMAC. Use SHA-512 hashing. Output returned in the <i>hmac</i> parameter is 64 bytes in length.
SSL3-MD5	Generate a MAC according to the SSL v3 protocol. Use MD5 hashing. Output returned in the <i>hmac</i> parameter is 16 bytes in length.
SSL3-SHA	Generate a MAC according to the SSL v3 protocol. Use SHA1 hashing. Output returned in the <i>hmac</i> parameter is 20 bytes in length.
Chaining Selection (Optional)	
FIRST	Specifies this is the first call in a series of chained calls. Intermediate results are stored in the <i>hash</i> field.
MIDDLE	Specifies this is a middle call in a series of chained calls. Intermediate results are stored in the <i>hash</i> field.
LAST	Specifies this is the last call in a series of chained calls.
ONLY	Specifies this is the only call and the call is not chained. This is the default.

### **text\_length**

Direction: Input

Type: Integer

Length of the *text* parameter in bytes. The length can be from 0 to 2147483647.

### **text**

Direction: Input

Type: String

Value for which an HMAC will be generated.

### **text\_id**

Direction: Input

Type: Integer

The ALET identifying the space where the text resides.

### **chain\_data\_length**

Direction: Input/Output

Type: Integer

The byte length of the *chain\_data* parameter. This must be 128 bytes.

#### chain\_data

Direction: Input/Output

Type: String

This field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on a FIRST call and may change it on subsequent MIDDLE and LAST calls. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message. The chain data has the following format:

Table 322. *chain\_data* parameter format

Offset	Length	Description
0	4	Flag word  <div> <div>Bit</div> <div>Meaning when set on</div> </div> <div> <div>0</div> <div>Cryptographic state object has been allocated</div> </div> <div> <div>1-31</div> <div>Reserved for IBM's use</div> </div>
4	44	Cryptographic state object handle
48	80	Reserved for IBM's use

#### key\_handle

Direction: Input

Type: String

The 44-byte handle of a generic secret key object. This parameter is ignored for MIDDLE and LAST chaining requests. See “Handles” on page 99 for the format of a *key\_handle*.

#### hmac\_length

Direction: Ignored

Type: Integer

Reserved field

#### hmac

Direction: Output

Type: String

Upon successful completion of an ONLY or LAST request, this field contains the generated HMAC value, left justified. The caller must provide an area large enough to hold the generated HMAC as defined by the mechanism specified. This field is ignored for FIRST and MIDDLE requests.

## Authorization

To use this service with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have at least USER (READ) authority (user access).

## Usage Notes

HMAC operations are performed in hardware or software.

If the FIRST rule is used to start a series of chained calls:



## PKCS #11 Generate HMAC

- The key used to initiate the chained calls must not be deleted until the chained calls are complete.
- The application should make a LAST call to free ICSF resources allocated. If processing is to be aborted without making a LAST call and the *chain\_data* parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object's handle.

---

## PKCS #11 Verify HMAC (CSFPHMV and CSFPHMV6)

Use the PKCS #11 Verify HMAC callable service to verify a hash message authentication code (MAC). This service does not support any recovery methods.

The key handle must be a handle of a PKCS #11 generic secret key object. The mechanism keyword specified in the rule array indicates the hash algorithm to use. The CKA\_VERIFY attribute for the secret key object must be true.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPHMV6.

### Format

```
CALL CSFPHMV(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    text_length,  
    text,  
    text_id,  
    chain_data_length,  
    chain_data,  
    key_handle,  
    hmac_length,  
    hmac)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

#### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

#### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. This value must be 1 or 2.

#### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 323. Keywords for verify HMAC

Keyword	Meaning
Mechanism (required)	
MD5	Verify an HMAC. Use MD5 hashing. Data supplied in the <i>hmac</i> parameter must be 16 bytes in length.
SHA-1	Verify an HMAC. Use SHA-1 hashing. Data supplied in the <i>hmac</i> parameter must be 20 bytes in length.
SHA-224	Verify an HMAC. Use SHA-224 hashing. Data supplied in the <i>hmac</i> parameter must be 28 bytes in length.
SHA-256	Verify an HMAC. Use SHA-256 hashing. Data supplied in the <i>hmac</i> parameter must be 32 bytes in length.
SHA-384	Verify an HMAC. Use SHA-384 hashing. Data supplied in the <i>hmac</i> parameter must be 48 bytes in length.
SHA-512	Verify an HMAC. Use SHA-512 hashing. Data supplied in the <i>hmac</i> parameter must be 64 bytes in length.
SSL3-MD5	Verify a MAC according to the SSL v3 protocol. Use MD5 hashing. Data supplied in the <i>hmac</i> parameter must be 16 bytes in length.
SSL3-SHA	Verify a MAC according to the SSL v3 protocol. Use SHA1 hashing. Data supplied in the <i>hmac</i> parameter must be 20 bytes in length.
Chaining Selection (Optional)	
FIRST	Specifies this is the first call in a series of chained calls. Intermediate results are stored in the hash field.
MIDDLE	Specifies this is a middle call in a series of chained calls. Intermediate results are stored in the hash field.
LAST	Specifies this is the last call in a series of chained calls.
ONLY	Specifies this is the only call and the call is not chained. This is the default.

## PKCS #11 Verify HMAC

### **text\_length**

Direction: Input

Type: Integer

Length of the *text* parameter in bytes. The length can be from 0 to 2147483647.

### **text**

Direction: Input

Type: String

Value for which an HMAC will be generated.

### **text\_id**

Direction: Input

Type: Integer

The ALET identifying the space where the text resides.

### **chain\_data\_length**

Direction: Input/Output

Type: Integer

The byte length of the *chain\_data* parameter. This must be 128 bytes.

### **chain\_data**

Direction: Input/Output

Type: String

This field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on a FIRST call and may change it on subsequent MIDDLE and LAST calls. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message. The chain data has the following format:

Table 324. *chain\_data* parameter format

Offset	Length	Description
0	4	Flag word
		<b>Bit</b> <b>Meaning when set on</b>
		0          Cryptographic state object has been allocated
		1-31      Reserved for IBM's use
4	44	Cryptographic state object handle
48	80	Reserved for IBM's use

### **key\_handle**

Direction: Input

Type: String

The 44-byte handle of a generic secret key object. This parameter is ignored for MIDDLE and LAST chaining requests. See "Handles" on page 99 for the format of a *key\_handle*.

### **hmac\_length**

Direction: Ignored

Type: Integer

Reserved field

### **hmac**

Direction: Input

Type: String

This field contains the HMAC value to be verified on ONLY and LAST requests, left justified. The caller must provide an HMAC value of the required length as determined by the mechanism specified. This field is ignored for FIRST and MIDDLE requests.

## Authorization

To use this service with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have at least USER (READ) authority (user access).

## Usage Notes

HMAC operations are performed in hardware or software.

Return code 4, reason code 8000 indicates the HMAC didn't verify.

If the FIRST rule is used to start a series of chained calls:

- The key used to initiate the chained calls must not be deleted until the chained calls are complete.
- The application should make a LAST call to free ICSF resources allocated. If processing is to be aborted without making a LAST call and the *chain\_data* parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object's handle.

---

## PKCS #11 One-way hash, sign, or verify (CSFPOWH and CSFPOWH6)

Use the one-way hash, sign, or verify callable service to generate a one-way hash on specified text, sign specified text, or verify a signature on specified text. For one-way hash, this service supports the following methods:

- MD2 - software only
- MD5 - software only
- SHA-1
- RIPEMD-160 - software only
- SHA-224
- SHA-256
- SHA-384
- SHA-512

For sign and verify, the following methods are supported:

- MD2 with RSA-PKCS 1.5
- MD5 with RSA-PKCS 1.5
- SHA1 with RSA-PKCS 1.5, DSA, or ECDSA
- SHA-224 with RSA-PKCS 1.5, DSA, or ECDSA
- SHA-256 with RSA-PKCS 1.5, DSA, or ECDSA
- SHA-384 with RSA-PKCS 1.5, DSA, or ECDSA
- SHA-512 with RSA-PKCS 1.5, DSA, or ECDSA

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPOWH6.

## Format

```
CALL CSFPOWH(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    text_length,
    text,
    text_id,
    chain_data_length,
    chain_data,
    handle,
    hash_length,
    hash )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. This value must be 1 or 2.

### **rule\_array**

Direction: Input

Type: String

## PKCS #11 One-way hash, sign, or verify (CSFPOWH)

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 325. Keywords for one-way hash generate

Keyword	Meaning
Hash Method (required)	
MD2	Hash algorithm is MD2 algorithm. Length of hash generated is 16 bytes.
MD5	Hash algorithm is MD5 algorithm. Length of hash generated is 16 bytes.
RPMD-160	Hash algorithm is RIPEMD-160. Length of hash generated is 20 bytes.
SHA-1	Hash algorithm is SHA-1. Length of hash generated is 20 bytes.
SHA-224	Hash algorithm is SHA-224. Length of hash generated is 28 bytes.
SHA-256	Hash algorithm is SHA-256. Length of hash generated is 32 bytes.
SHA-384	Hash algorithm is SHA-384. Length of hash generated is 48 bytes.
SHA-512	Hash algorithm is SHA-512. Length of hash generated is 64 bytes.
DETERMIN	For use with non-chained RSA signature verifies only. Hash algorithm is to be determined from the input signature.
Chaining Flag (optional)	
FIRST	Specifies this is the first call in a series of chained calls. Intermediate results are stored in the <i>hash</i> and <i>chain_data</i> fields. Cannot be specified with hash method DETERMIN.
MIDDLE	Specifies this is a middle call in a series of chained calls. Intermediate results are stored in the <i>hash</i> and <i>chain_data</i> fields. Cannot be specified with hash method DETERMIN.
LAST	Specifies this is the last call in a series of chained calls. Cannot be specified with hash method DETERMIN.
ONLY	Specifies this is the only call and the call is not chained. This is the default.
Requested Operation (optional)	
HASH	The specified text is to be hashed only. This is the default. Cannot be specified (either explicitly or by default) with hash method DETERMIN.
SIGN-RSA	The data is to be hashed then signed using RSA-PKCS 1.5 formatting. Any hash method is acceptable except RPMD-160 and DETERMIN.
SIGN-DSA	The data is to be hashed then signed using DSA. The hash method must be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.
SIGN-EC	The data is to be hashed then signed using ECDSA. The hash method must be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.
VER-RSA	The data is to be hashed then signature verified using RSA-PKCS 1.5 formatting. Any hash method is acceptable except RPMD-160. This operation is required for hash method DETERMIN.
VER-DSA	The data is to be hashed then signature verified using DSA. The hash method must be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.

## PKCS #11 One-way hash, sign, or verify (CSFPOWH)

Table 325. Keywords for one-way hash generate (continued)

Keyword	Meaning
VER-EC	The data is to be hashed then signature verified using ECDSA. The hash method must be SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.

### text\_length

Direction: Input

Type: Integer

The length of the text parameter in bytes.

If you specify the FIRST or MIDDLE keyword, then the text length must be a multiple of the block size of the hash method. For MD2, this is a multiple of 16 bytes. For MD5, RPMD-160, SHA-1, SHA-224, and SHA-256, this is a multiple of 64 bytes. For SHA-384 and SHA-512, this is a multiple of 128 bytes. For ONLY and LAST, this service performs the required padding according to the algorithm specified. The length can be from 0 to 2147483647.

### text

Direction: Input

Type: String

Value to be hashed

### text\_id

Direction: Input

Type: Integer

The ALET identifying the space where the text resides.

### chain\_data\_length

Direction: Input/Output

Type: Integer

The byte length of the *chain\_data* parameter. This must be 128 bytes.

### chain\_data

Direction: Input/Output

Type: String

This field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on a FIRST call and may change it on subsequent MIDDLE calls. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message. The chain data has the following format:

Table 326. *chain\_data* parameter format

Offset	Length	Description
0	4	Flag word
		<b>Bit</b> <b>Meaning when set on</b>
		0          Cryptographic state object has been allocated
		1-31      Reserved for IBM's use
4	44	Cryptographic state object handle
48	80	Reserved for IBM's use

### handle

## PKCS #11 One-way hash, sign, or verify (CSFPOWH)

Direction: Input

Type: String

For hash requests, this is the 44-byte name of the token to which this hash operation is related. The first 32 bytes of the handle are meaningful. The remaining 12 bytes are reserved. See “Handles” on page 99 for the format of a *handle*.

For sign and verify requests, this is the 44-byte handle to the key object that is to be used. For FIRST and MIDDLE chaining requests, only the first 32 bytes of the handle are meaningful, to identify the token.

### hash\_length

Direction: Input/Output

Type: Integer

The length of the supplied hash field in bytes.

For hash requests, this field is input only. For SHA-1 and RPMD-160 this must be at least 20 bytes; for MD2 and MD5 this must be at least 16 bytes. For SHA-224 and SHA-256, this must be at least 32 bytes. Even though the length of the SHA-224 hash is less than SHA-256, the extra bytes are used as a work area during the generation of the hash value. The SHA-224 value is left-justified and padded with 4 bytes of binary zeroes. For SHA-384 and SHA-512, this must be at least 64 bytes. Even though the length of the SHA-384 hash is less than SHA-512, the extra bytes are used as a work area during the generation of the hash value. The SHA-384 value is left-justified and padded with 16 bytes of binary zeroes.

For FIRST and MIDDLE sign and verify requests, this field is ignored.

For LAST and ONLY sign requests, this field is input/output. If the signature generation is successful, ICSF will update this field with the length of the generated signature. If the signature generation is unsuccessful because the supplied hash field is too small, ICSF will update this field with the required length.

For LAST and ONLY verify requests, this field is input only.

### hash

Direction: Input/Output

Type: String

This field contains the hash or signature, left-justified. The processing of the rest of the field depends on the implementation.

For hash requests, this field is the generated hash. If you specify the FIRST or MIDDLE keyword, this field contains the intermediate hash value. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message.

For FIRST and MIDDLE sign and verify requests, this field is ignored.

For LAST and ONLY sign requests, this field is the generated signature.

For LAST and ONLY verify requests, this field is input signature to be verified.

## Authorization

To use this service to sign or verify with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service to sign or verify with a private object, the caller must have at least USER (READ) authority (user access).



### Usage Notes

If the FIRST rule is used to start a series of chained calls, the application must not change the Hash Method or Requested Operation rules between the calls. The behavior of the service is undefined if the rules are changed.

If the FIRST rule is used to start a series of chained calls, the application should make a LAST call to free ICSF resources allocated. If processing is to be aborted without making a LAST call and the *chain\_data* parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object's handle.

The CSFSERV resource name that protects this service is CSFOWH, the same resource name used to protect the non-PKCS #11 One Way Hash service.

For hash method DETERMIN, ICSF determines the hashing method by RSA decrypting the input signature using the specified public key and examining the result. ICSF will return the "signature did not verify" error (return code 4, reason code X'2AF8') if this process is unsuccessful for any of the following reasons:

1. ICSF cannot successfully perform the decryption because the public key is the wrong size.
2. The resulting clear text block is not properly RSA-PKCS 1.5 formatted.
3. The resulting clear text block indicates a hashing algorithm not supported by this service was used.

---

## PKCS #11 Private key sign (CSFPPKS and CSFPPKS6)

Use the PKCS #11 private key sign callable service to:

- Decrypt or sign data using an RSA private key using zero-pad or PKCS #1 v1.5 formatting
- Sign data using a DSA private key
- Sign data using an Elliptic Curve private key in combination with DSA

The key handle must be a handle of a PKCS #11 private key object. When the request type keyword DECRYPT is specified in the rule array, CKA\_DECRYPT attribute must be true. When no request type is specified, the CKA\_SIGN attribute must be true.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPPKS6.

### Format

```
CALL CSFPPKS(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    cipher_value_length,  
    cipher_value,  
    key_handle,  
    clear_value_length,  
    clear_value )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array\_parameter*. This value may be 1 or 2.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service.

Table 327. Keywords for private key sign

Keyword	Meaning
Mechanism (One of the following must be specified)	
RSA-ZERO	Mechanism is RSA decryption or signature generation using zero-pad formatting
RSA-PKCS	Mechanism is RSA decryption or signature generation using PKCS #1 v1.5 formatting
DSA	Mechanism is DSA signature generation
ECDSA	Mechanism is Elliptic Curve with DSA signature generation
Request type (optional)	
DECRYPT	The request is to decrypt data. This type of request requires the CKA_DECRYPT attribute to be true. If DECRYPT is not specified, the CKA_SIGN attribute must be true. Valid with RSA only.

## PKCS #11 Private key sign

### **cipher\_value\_length**

Direction: Input

Type: Integer

Length of the *cipher\_value* parameter in bytes.

### **cipher\_value**

Direction: Input

Type: String

For decrypt, this is the value to be decrypted. Otherwise this is the value to be signed. For RSA-PKCS signature requests, the data to be signed is expected to be a DER encoded DigestInfo structure. For DSA and ECDSA signature requests, the data to be signed is expected to be a SHA1, SHA224, SHA256, SHA384 or SHA512 digest.

### **key\_handle**

Direction: Input

Type: String

The 44-byte handle of a private key object. See “Handles” on page 99 for the format of a *key\_handle*.

### **clear\_value\_length**

Direction: Input/Output

Type: Integer

Length of the *clear\_value* parameter in bytes. On output, this is updated to be the actual length of the decrypted value or the generated signature.

### **clear\_value**

Direction: Output

Type: String

For decrypt, this field will contain the decrypted value. Otherwise this field will contain the generated signature.

## Authorization

To use this service with a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have USER (READ) authority (user access).

## Usage Notes

DSA, ECDSA and RSA operations may be done in hardware or software.

Request type DECRYPT is not supported for an Elliptic Curve or DSA private key.

---

## PKCS #11 Public key verify (CSFPPKV and CSFPPKV6)

Use the PKCS #11 public key verify callable service to:

- Encrypt or verify data using an RSA public key using zero-pad or PKCS #1 v1.5 formatting. For encryption, the encrypted data is returned
- Verify a signature using a DSA public key. No data is returned
- Verify a signature using an Elliptic Curve public key in combination with DSA. No data is returned

The key handle must be a handle of a PKCS #11 public key object. When the request type keyword ENCRYPT is specified in the rule array, CKA\_ENCRYPT attribute must be true. When no request type is specified, the CKA\_VERIFY attribute must be true.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPPKV6.

## Format

```
CALL CSFPPKV(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    clear_value_length,
    clear_value,
    key_handle,
    cipher_value_length,
    cipher_value )
```

## Parameters

### return\_code

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### reason\_code

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### exit\_data\_length

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### exit\_data

Direction: Ignored

Type: String

This field is ignored.

### rule\_array\_count

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. This value must be 1 or 2.

## PKCS #11 Public key verify

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service.

*Table 328. Keywords for public key verify*

Keyword	Meaning
Mechanism (One of the following must be specified)	
RSA-ZERO	Mechanism is RSA encryption or signature verification using zero-pad formatting
RSA-PKCS	Mechanism is RSA encryption or signature verification using PKCS #1 v1.5 formatting
DSA	Mechanism is DSA signature verification
ECDSA	Mechanism is Elliptic Curve with DSA signature verification
Request type (optional)	
ENCRYPT	The request is to encrypt data. This type of request requires the CKA_ENCRYPT attribute to be true. If ENCRYPT is not specified, the CKA_VERIFY attribute must be true. Valid with RSA only.

### **clear\_value\_length**

Direction: Input

Type: Integer

The length of the `clear_value` parameter

### **clear\_value**

Direction: Input

Type: String

For encrypt, this is the value to be encrypted. Otherwise this is the signature is be verified.

### **key\_handle**

Direction: Input

Type: String

The 44-byte handle of public key object. See “Handles” on page 99 for the format of a *key\_handle*.

### **cipher\_value\_length**

Direction: Input/Output

Type: Integer

For encrypt, on input, this is the length of the *cipher\_value* parameter in bytes. On output, this is updated to be the actual length of the text encrypted into the *cipher\_value* parameter. For signature verification, this is the length of the data to be verified (input only).

### **cipher\_value**

Direction: Input/Output

Type: String

For encrypt, this is the encrypted value (output only). For signature verification, this is the data to be verified (input only). For RSA-PKCS signature verification requests, the data to be verified is expected to be a DER encoded DigestInfo structure. For DSA and ECDSA signature verification requests, the data to be verified is expected to be a SHA1, SHA224, SHA256, SHA384 or SHA512 digest.

## Authorization

To use this service with a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have USER (READ) authority (user access).

## Usage Notes

DSA, ECDSA, and RSA operations may be done in hardware or software.

Request type ENCRYPT is not supported for an Elliptic Curve or DSA public key.

## PKCS #11 Pseudo-random function (CSFPPRF and CSFPPRF6)

Use the PKCS #11 Pseudo-random callable service to generate pseudo-random output of arbitrary length. This service does not support any recovery methods.

The mechanism keyword specified in the rule array indicates what derivation protocol to use. The derive parms list provides additional input/output data. The format of this list is dependent on the protocol being used.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPPRF6.

## Format

```
CALL CSFPPRF(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    handle,  
    parms_list_length,  
    parms_list,  
    prf_output_length,  
    prf_output)
```

## Parameters

**return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

## reason code

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

## PKCS #11 Pseudo-random function

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. This value must be 1.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 329. Keywords for PKCS #11 Pseudo-random function

Keyword	Meaning
Mechanism (required)	
TLS-PRF	Use the TLS Pseudo-Random Function derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_PRF. This mechanism derives deterministic random bytes from a caller supplied secret key object and other parameters.
PRNG	Generate pseudo-random bytes using the best source available. Possible sources are: CCA coprocessors, Enterprise PKCS #11 coprocessors, or a pseudo (deterministic) random algorithm. CCA coprocessors are only used for entropy seeding when ICSF is running in FIPS standard mode or FIPS compatibility mode.
PRNGFIPS	Generate pseudo-random bytes using the best source available. Possible sources are: Enterprise PKCS #11 coprocessors or a pseudo (deterministic) random algorithm, consistent with NIST SP 800-90. PRNGFIPS allows the caller to demand FIPS processing, in which case CCA coprocessors are only used for entropy seeding.

### **handle**

Direction: Input

Type: String

For mechanism TLS-PRF, this is the 44-byte handle of the source secret key object. The CKA\_DERIVE attribute for the secret key object must be true. If no key is to be used, set the handle to all blanks.

For mechanisms PRNG and PRNGFIPS, this is the 44-byte name of the token to which this operation is related. The first 32 bytes of the handle are meaningful. The remaining 12 bytes are reserved and must be blanks.

See “Handles” on page 99 for the format of a *handle*.

### **parms\_list\_length**

Direction: Input

Type: Integer

The length of the parameters supplied in the *parms\_list* parameter in bytes.

### parms\_list

Direction: Input/Output

Type: String

The protocol specific parameters. This field has a varying format depending on the mechanism specified:

Table 330. *parms\_list* parameter format for TLS-PRF mechanism

Offset	Length in bytes	Direction	Description
0	1	input	PRF function code – x'00', use combined MD5/SHA1 digest algorithm as defined in TLS 1.0/1.1, otherwise use the following single digest algorithm as defined in TLS 1.2: x'01' = SHA256, x'02' = SHA384, and x'03' = SHA512
1	3	not applicable	reserved
4	4	input	length in bytes of the label (x). where 1 <= length <= 256
8	4	input	length in bytes of the seed (y), where 1 <= length <= 256
12	x	input	label
12+x	y	input	seed

For the PRNG and PRNGFIPS mechanisms, there are no parameters. The *parms\_list\_length* parameter must be set to zero for this mechanism

### prf\_output\_length

Direction: Input

Type: Integer

The length in bytes of pseudo-random data to be generated and returned in the *prf\_output* parameter. The maximum length is 2147483647 bytes.

### prf\_output

Direction: Output

Type: String

The pre-allocated area in which the pseudo-random data is returned.

## Authorization

To use this service with a public object for mechanism TLS-PRF, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object for mechanism TLS-PRF, the caller must have at least USER (READ) authority (user access).

## Usage Notes

Pseudo-random functions operations are performed in hardware or software.

The CSFSERV resource name that protects this service is CSFRNG, the same resource name used to protect the non-PKCS #11 Random Number Generation service.

## PKCS #11 Set attribute value (CSFPSAV and CSFPSAV6)

Use the set attribute value callable service (CSFPSAV) to update the attributes of an object.



## PKCS #11 Set attribute value

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPSAV6.

### Format

```
CALL CSFPSAV(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    handle,  
    rule_array_count,  
    rule_array,  
    attribute_list_length,  
    attribute_list)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

#### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

#### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

#### **handle**

Direction: Input

Type: String

The 44-byte handle of the object. See “Handles” on page 99 for the format of a *handle*.

#### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords supplied in the *rule\_array* parameter. This value must be 0.

### rule\_array

Direction: Input

Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

### attribute\_list\_length

Direction: Input

Type: Integer

The length of the *attribute\_list* parameter in bytes.

The maximum size in bytes is 32752.

### attribute\_list

Direction: Input

Type: String

A list of object attributes.

**Note:** Lengths in the attribute list and attribute structures are unsigned integers.

See “Attribute List” on page 98 for the format of an *attribute\_list*.

## Authorization

Table 331. Authorization requirements for the set attribute value callable service

Action	Object	Authority required
Set	Public object, except a CA certificate	USER (UPDATE) or SO (READ)
Set	Private object, except a CA certificate	USER (UPDATE) or SO (CONTROL)
Set	Public CA certificate object	USER (CONTROL) or SO (READ)
Set	Private CA certificate object	USER (CONTROL) or SO (CONTROL)

### Note:

- Session and token objects require the same authority.
- See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the SO and User PKCS #11 roles and how ICSF determines that a certificate is a CA certificate.

## Usage Notes

When updating the attributes of an object, all attributes in the template will be processed and the value used is that of the last instance processed.

Key pair generation may be done in hardware or software.

### PKCS #11 Secret key decrypt (CSFPSKD and CSFPSKD6)

Use the PKCS #11 secret key decrypt callable service to decipher data using a clear symmetric key. AES, DES, BLOWFISH, and RC4 are supported. This service supports CBC, ECB, Galois/Counter, and stream modes and PKCS #7 padding. The key handle must be a handle of a PKCS #11 secret key object. The CKA\_DECRYPT attribute must be true.

If the length of output field is too short to hold the output, the service will fail and return the required length of the output field in the `clear_text_length` parameter.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPSKD6.

### Format

```
CALL CSFPSKD(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_handle,
    initialization_vector_length,
    initialization_vector,
    chain_data_length,
    chain_data,
    cipher_text_length,
    cipher_text,
    cipher_text_id,
    clear_text_length,
    clear_text,
    clear_text_id )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### exit\_data

Direction: Ignored

Type: String

This field is ignored.

### rule\_array\_count

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. This value must be 0, 1, 2, or 3.

### rule\_array

Direction: Input

Type: String

Keywords that provide control information to the callable service.

Table 332. Keywords for secret key decrypt

Keyword	Meaning
Encryption Mechanism (Optional. No default. If not specified, mechanism will be taken from key type of secret key. If specified , must match key type)	
AES	AES algorithm will be used.
DES	DES algorithm will be used. This is only single-key encryption.
DES3	DES3 algorithm will be used, This includes double- and triple-key encryption.
BLOWFISH	BLOWFISH algorithm will be used.
RC4	RC4 algorithm will be used. This is a stream cipher.
Processing Rule (optional)	
CBC	Performs cipher block chaining. The cipher text length must be a multiple of the block size for the specified algorithm (8 bytes for DES, DES3, and BLOWFISH, 16 bytes for AES). CBC is the default value for DES, DES3, AES, and BLOWFISH. CBC cannot be specified for RC4.
CBC-PAD	Performs cipher block chaining. The cipher text length must be greater than zero and a multiple of the block size for the specified algorithm. For FINAL and ONLY calls, PKCS #7 padding is performed. For this reason, the clear text will always be shorter than the cipher text and may even be zero length. CBC-PAD cannot be specified for BLOWFISH or RC4.
ECB	Performs electronic code book encryption. The cipher text length must be a multiple of the block size for the specified algorithm. ECB cannot be specified for BLOWFISH or RC4.
GCM	Performs Galois/Counter mode encryption. The cipher text length must be greater than zero. The clear text will be shorter than the cipher text and may even be zero length due to the truncation of the authentication tag. GCM may only be specified with AES. GMAC is a specialized form of GCM where no plain text is returned.
STREAM	Performs a stream cipher. STREAM cannot be specified for BLOWFISH, DES, DES3, or AES. STREAM is the default value for RC4.
Chaining Selection (optional)	

Table 332. Keywords for secret key decrypt (continued)

Keyword	Meaning
INITIAL	Specifies this is the first call in a series of chained calls. For cipher block chaining, the initialization vector is taken from the <i>initialization_vector</i> parameter. Cannot be specified with processing rule ECB or GCM.
CONTINUE	Specifies this is a middle call in a series of chained calls. Intermediate results are read from and stored in the <i>chain_data</i> field. Cannot be specified with processing rule ECB or GCM.
FINAL	Specifies this is the last call in a series of chained calls. Intermediate results are read from the <i>chain_data</i> field. Cannot be specified with processing rule ECB or GCM.
ONLY	Specifies this is the only call and the call is not chained. For cipher block chaining, the initialization vector is taken from the <i>initialization_vector</i> parameter. For Galois Counter mode, the initialization parameters are taken from the <i>initialization_vector</i> parameter. ONLY is the default chaining.

## key\_handle

Direction: Input

Type: String

The 44-byte handle of secret key object. See “Handles” on page 99 for the format of a *key\_handle*.

## initialization\_vector\_length

Direction: Input

Type: Integer

Length of the *initialization\_vector* in bytes. For CBC and CBC-PAD, this must be 8 bytes for DES and BLOWFISH and 16 bytes for AES. For GCM, this must be the size of the *initialization\_vector* field (28 bytes).

## initialization\_vector

Direction: Input

Type: String

This field has a varying format depending on the mechanism specified. For CBC and CBC-PAD this is the 8 or 16 byte initial chaining value. The format for GCM is shown in the following table.

Table 333. *initialization\_vector* parameter format for GCM mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	length in bytes of the initialization vector. The minimum value is 1. The maximum value is 128. 12 is recommended.
4	8	Input	64-bit address of the initialization vector. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers.
12	4	Input	length in bytes of the additional authentication data. The minimum value is 0. The maximum value is 1048576.
16	8	Input	64-bit address of the additional authentication data. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length of the additional authentication data is zero.
24	4	Input	Length in bytes of the desired authentication tag. This value must be one of 4, 8, 12, 13, 14, 15, or 16.

### chain\_data\_length

Direction: Input/Output Type: Integer

The byte length of the chain\_data parameter. This must be 128 bytes.

### chain\_data

Direction: Input/Output Type: String

This field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on an INITIAL call, and may change it on subsequent CONTINUE calls. Your application must not change the data in this field between the sequence of INITIAL, CONTINUE, and FINAL calls for a specific message. The chain data has the following format:

Table 334. chain\_data parameter format

Offset	Length	Description
0	4	Flag word
		<b>Bit</b> <b>Meaning when set on</b>
		<b>0</b> Cryptographic state object has been allocated
		<b>1-31</b> Reserved for IBM's use
4	44	Cryptographic state object handle
48	80	Reserved for IBM's use

### cipher\_text\_length

Direction: Input Type: Integer

Length of the cipher\_text parameter in bytes. Except for processing rule GCM, the length can be up to 2147483647. For processing rule GCM, the length cannot exceed 1048576 plus the length of the tag.

### cipher\_text

Direction: Input Type: String

Text to be decrypted.

### cipher\_text\_id

Direction: Input Type: Integer

The ALET identifying the space where the cipher text resides.

### clear\_text\_length

Direction: Input/Output Type: Integer

On input, the length in bytes of the clear\_text parameter. On output, the length of the text decrypted into the clear\_text parameter

### clear\_text

Direction: Output Type: String

Decrypted text

### clear\_text\_id

## PKCS #11 Secret key decrypt

Direction: Input

Type: Integer

The ALET identifying the space where the clear text resides.

### Authorization

To use this service with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have at least USER (READ) authority (user access).

### Usage Notes

If the INITIAL rule is used to start a series of chained calls:

- The same key\_handle, Encryption Mechanism and Processing Rule must be used on the subsequent CONTINUE and FINAL calls.
- The key used to initiate the chained calls must not be deleted until the chained calls are complete.
- The application should make a FINAL call to free ICSF resources allocated. If processing is to be aborted without making a FINAL call and the *chain\_data* parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object's handle.

GCM decryption may be used to verify a GMAC on some authentication data. To do this request AES decryption with processing rule. The *cipher\_text\_length* and *cipher\_text* fields must be set to the length and value of the GMAC to be verified. A *return\_code* of zero and no *clear\_text* data returned means the GMAC verification was successful.

A secure key may not be used for Processing Rule GCM.

---

## PKCS #11 Secret key encrypt (CSFPSKE and CSFPSKE6)

Use the PKCS #11 secret key encrypt callable service to encipher data using a clear symmetric key. AES, DES, BLOWFISH, and RC4 are supported. This service supports CBC, ECB, Galois/Counter, and stream modes and PKCS #7 padding. The key handle must be a handle of a PKCS #11 secret key object. The CKA\_ENCRYPT attribute must be true.

If the length of output field is too short to hold the output, the service will fail and return the required length of the output field in the *cipher\_text\_length* parameter.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPSKE6.

## Format

```
CALL CSFPSKE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_handle,
    initialization_vector_length,
    initialization_vector,
    chain_data_length,
    chain_data,
    clear_text_length,
    clear_text,
    clear_text_id,
    cipher_text_length,
    cipher_text,
    cipher_text_id )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. This value must be 0, 1, 2, or 3.

### **rule\_array**

Direction: Input

Type: String



## PKCS #11 Secret key encrypt (CSFPSKE)

Keywords that provide control information to the callable service.

Table 335. Keywords for secret key encrypt

Keyword	Meaning
Encryption Mechanism (Optional. No default. If not specified, mechanism will be taken from key type of secret key. If specified , must match key type)	
AES	AES algorithm will be used.
DES	DES algorithm will be used. This is only single-key encryption.
DES3	DES3 algorithm will be used, This includes double- and triple-key encryption.
BLOWFISH	BLOWFISH algorithm will be used.
RC4	RC4 algorithm will be used. This is a stream cipher.
Processing Rule (optional)	
CBC	Performs cipher block chaining. The text length must be a multiple of the block size for the specified algorithm (8 bytes for DES, DES3, and BLOWFISH, 16 bytes for AES). CBC is the default value for DES, DES3, AES, and BLOWFISH. CBC cannot be specified for RC4.
CBC-PAD	Performs cipher block chaining. Except for FINAL and ONLY chaining calls, the clear text length must be a multiple of the block size for the specified algorithm. For FINAL and ONLY calls: <ul style="list-style-type: none"><li>• The clear text length may be shorter than the block size and may even be zero.</li><li>• PKCS #7 padding is performed. Thus, the cipher text will always be longer than the clear text.</li></ul> CBC-PAD cannot be specified for BLOWFISH or RC4.
ECB	Performs electronic code book encryption. The text length must be a multiple of the block size for the specified algorithm. ECB cannot be specified for BLOWFISH or RC4.
GCM	Performs Galois/Counter mode encryption. The clear text length may be shorter than the block size and may even be zero. The authentication tag is returned appended to the cipher text. GCM may only be specified with AES. GMAC is a specialized form of GCM where no plain text is specified.
GCMIVGEN	Performs similarly to the GCM processing rule except that ICSF will generate part of the initialization vector and return it in the <i>initialization_vector</i> parameter. Having ICSF generate the initialization vector ensures that initialization vectors are never repeated for a given key object.
STREAM	Performs a stream cipher. STREAM cannot be specified for BLOWFISH, DES, DES3, or AES. STREAM is the default value for RC4.
Chaining Selection (optional)	
INITIAL	Specifies this is the first call in a series of chained calls. For cipher block chaining, the initialization vector is taken from the <i>initialization_vector</i> parameter. Intermediate results are stored in the <i>chain_data</i> field. Cannot be specified with processing rule ECB, GCM, or GCMIVGEN.

Table 335. Keywords for secret key encrypt (continued)

Keyword	Meaning
CONTINUE	Specifies this is a middle call in a series of chained calls. Intermediate results are read from and stored in the <i>chain_data</i> field. Cannot be specified with processing rule ECB, GCM, or GCMIVGEN.
FINAL	Specifies this is the last call in a series of chained calls. Intermediate results are read from the <i>chain_data</i> field. Cannot be specified with processing rule ECB, GCM, or GCMIVGEN.
ONLY	Specifies this is the only call and the call is not chained. For cipher block chaining, the initialization vector is taken from the <i>initialization_vector</i> parameter. For Galois Counter mode, the initialization parameters are taken from the <i>initialization_vector</i> parameter. ONLY is the default chaining.

### key\_handle

Direction: Input

Type: String

The 44-byte handle of secret key object. See “Handles” on page 99 for the format of a *key\_handle*.

### Initialization\_vector\_length

Direction: Input

Type: Integer

Length of the *initialization\_vector* in bytes. For CBC and CBC-PAD, this must be 8 bytes for DES and BLOWFISH and 16 bytes for AES. For GCM and GCMVGEN, this must be the size of the *initialization\_vector* field (28 bytes).

### initialization\_vector

Direction: Input

Type: String

This field has a varying format depending on the mechanism specified. For CBC and CBC-PAD this is the 8 or 16 byte initial chaining value. The format for GCM and GCMIVGEN are shown in the following tables.

Table 336. *initialization\_vector* parameter format for GCM mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	length in bytes of the initialization vector area. The minimum value is 1. The maximum value is 128. 12 is recommended.
4	8	Input	64-bit address of the initialization vector area. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers.
12	4	Input	length in bytes of the additional authentication data. The minimum value is 0. The maximum value is 1048576.
16	8	Input	64-bit address of the additional authentication data. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length of the additional authentication data is zero.
24	4	Input	Length in bytes of the desired authentication tag. This value must be one of 4, 8, 12, 13, 14, 15, or 16.

## PKCS #11 Secret key encrypt (CSFPSKE)

Table 337. *initialization\_vector* parameter format for GCMIVGEN mechanism

Offset	Length in bytes	Direction	Description
0	4	Input	Nonce value which ICSF is to use as the first 4 bytes of the initialization vector. The remaining 8 bytes will be generated and returned to the caller in the initialization vector area.
4	8	Input	64-bit address of the initialization vector area into which ICSF will store the 8 bytes it generates. The area must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers.  The complete initialization vector to be used for decryption is the 4-byte nonce concatenated with the 8 bytes stored in the area
12	4	Input	length in bytes of the additional authentication data. The minimum value is 0. The maximum value is 1048576.
16	8	Input	64-bit address of the additional authentication data. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length of the additional authentication data is zero.
24	4	Input	Length in bytes of the desired authentication tag. This value must be one of 4, 8, 12, 13, 14, 15, or 16.

### **chain\_data\_length**

Direction: Input/Output

Type: Integer

The byte length of the *chain\_data* parameter. This must be 128 bytes.

### **chain\_data**

Direction: Input/Output

Type: String

This field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on an INITIAL call, and may change it on subsequent CONTINUE calls. Your application must not change the data in this field between the sequence of INITIAL, CONTINUE, and FINAL calls for a specific message. The chain data has the following format:

Table 338. *chain\_data* parameter format

Offset	Length	Description
0	4	Flag word
		<b>Bit</b> <b>Meaning when set on</b>
		<b>0</b> Cryptographic state object has been allocated
		<b>1-31</b> Reserved for IBM's use
4	44	Cryptographic state object handle
48	80	Reserved for IBM's use

### **clear\_text\_length**

Direction: Input

Type: Integer

Length of the *clear\_text* parameter in bytes. Except for processing rules GCM and GCMIVGEN, the length can be up to 2147483647. For processing rules GCM and GCMIVGEN, the length cannot exceed 1048576.

### **clear\_text**

Direction: Input

Type: String

Text to be encrypted

**clear\_text\_id**

Direction: Input

Type: Integer

The ALET identifying the space where the clear text resides.

**cipher\_text\_length**

Direction: Input/Output

Type: Integer

On input, the length in bytes of the *cipher\_text* parameter. On output, the length of the text encrypted into the *cipher\_text* parameter.

**cipher\_text**

Direction: Output

Type: String

Encrypted text

**cipher\_text\_id**

Direction: Output

Type: Integer

The ALET identifying the space where the cipher text resides.

## Authorization

To use this service with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have at least USER (READ) authority (user access).

## Usage Notes

If the INITIAL rule is used to start a series of chained calls:

- The same key\_handle, Encryption Mechanism and Processing Rule must be used on the subsequent CONTINUE and FINAL calls.
- The key used to initiate the chained calls must not be deleted until the chained calls are complete.
- The application should make a FINAL call to free ICSF resources allocated. If processing is to be aborted without making a FINAL call and the *chain\_data* parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object's handle.

GCM encryption may be used to produce a GMAC on some authentication data. To do this, request AES encryption with processing rule GCM or GCMVGEN. The *clear\_text\_length* field must be set to zero. The authentication tag (the GMAC) is returned in the *cipher\_text* field.

For Processing Rule GCMVGEN, the total number of initialization vector generations for a token key object is limited to 4294967295. Once this number is exceeded, the key object will no longer be eligible for Processing Rule GCMVGEN and is considered "retired". This usage counter is maintained in the TKDS as part

## PKCS #11 Secret key encrypt (CSFPKSKE)

of the key object. For keys that are copied using CSFPTRC (C\_CopyObject), the existing counter value is copied to the new key object, but not synchronized after that.

For Processing Rule GCMIVGEN, session key objects have no maximum lifetime. They may be retired at any time. Once retired, the key object will no longer be eligible for Processing Rule GCMIVGEN.

For Processing Rule GCMIVGEN, the nonce value portion of the initialization vector is predetermined by the caller. It is used to ensure that initialization vector values are not repeated for any given key value. The caller should provide a random value and change the value as often as practical. It must be changed whenever:

- a given key value is replicated as a new persistent key object
- a given persistent key object is replicated as a new session key object
- a given session key value is re-instantiated after system IPL
- a given key value is re-instantiated after ICSF indicates it has been retired

Use of Processing Rule GCMIVGEN with token key objects requires that the first 4 bytes of ECVTSPLX or CVTSNAME be set to a unique value with respect to other systems. See *z/OS Cryptographic Services ICSF System Programmer's Guide*, SA22-7520 for information on how to set these fields.

A session key object should never be used for Processing Rule GCMIVGEN if the key value is distributed to multiple systems outside the current sysplex where new initialization vectors may be generated. Use only token key objects in such cases. If session key objects are used, the other systems must use different nonces.

For Processing Rule GCMIVGEN, the 8 bytes of generated initialization vector are stored back into the initialization vector area before the GCM operation is performed. This allows the generated initialization vector to be part of the additional authentication data, if desired.

A secure key may not be used for Processing Rule GCM or GCMIVGEN.

---

## PKCS #11 Token record create (CSFPTRC and CSFPTRC6)

Use the token record create callable service (CSFPTRC) to do these tasks:

- Initialize or re-initialize a z/OS PKCS #11 token
- Create or copy a token object in the token data set
- Create or copy a session object for the current PKCS #11 session

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPTRC6.

## Format

```
CALL CSFPTRC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    handle,
    rule_array_count,
    rule_array,
    attribute_list_length,
    attribute_list)
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **handle**

Direction: Input/Output

Type: String

On input, the 44-byte name of the z/OS PKCS #11 token to be initialized, or the token handle of the object to be created or copied. For the create or re-create functions, the first 32 bytes of the handle are meaningful on input. The remaining 12 bytes are filled in by the token record create service. For the copy function, all 44 bytes of the handle are significant on input.

On output, the 44-byte handle of the z/OS PKCS #11 token or object created.

See “Handles” on page 99 for the format of a *handle*.

### **rule\_array\_count**

Direction: Input

Type: Integer

## PKCS #11 Token record create

The number of keywords supplied in the *rule\_array* parameter. The value must be 1 or 2.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Keyword	Meaning
<b>One of these two keywords must be specified:</b>	
TOKEN	Specifies that a token is to be initialized. If the token exists in the token data set, the RECREATE keyword must be specified.
OBJECT	Specifies that an object (token object or session object) is to be created. If the object is to be a copy of an existing object, the COPY keyword must be specified.
<b>This keyword is optional, and valid only with TOKEN:</b>	
RECREATE	Specifies that the token exists and is to be re-initialized. All objects of the existing token will be deleted.
<b>This keyword is optional, and valid only with OBJECT:</b>	
COPY	Specifies that the object specified by the handle is to be copied into a new object.

### **attribute\_list\_length**

Direction: Input

Type: Integer

Length of the *attribute\_list* parameter in bytes.

The maximum size in bytes is 32752.

### **attribute\_list**

Direction: Input

Type: String

List of token or object attributes.

When creating or re-creating a token, the *attribute\_list* parameter has this format:

Bytes	Description
0 - 31	Manufacturer ID
32 - 47	Model
48 - 63	Serial number
64 - 67	Reserved for IBM's use. Must be hexadecimal zeros.

**Note:** The strings supplied for Manufacturer ID, Model, and Serial number are assumed to be from code page IBM1047.

For objects, see "Attribute List" on page 98 for the format of an *attribute\_list*.

## Authorization

**Note:** Session and token objects require the same SAF authority.

Table 339. Authorization requirements for the token record create callable service

Action	Source object (Copy only)	Token / Object being created	PKCS #11 role Authority required
Create or recreate token	N/A	Token	SO (UPDATE)
Create object	N/A	Public object, except a CA certificate	USER (UPDATE) or SO (READ)
Create object	N/A	Private object, except a CA certificate	USER (UPDATE) or SO (CONTROL)
Create object	N/A	Public CA certificate object	USER (CONTROL) or SO (READ)
Create object	N/A	Private CA certificate object	USER (CONTROL) or SO (CONTROL)
Copy object	Public object, except a CA certificate	Public object, except a CA certificate	USER (UPDATE) or SO (READ)
Copy object	Public object or private object, except a CA certificate	Private object, except a CA certificate	USER (UPDATE) or SO (CONTROL)
Copy object	Private object, except a CA certificate	Public object, except a CA certificate	USER (UPDATE)
Copy object	Public object, where source or target or both are CA certificate objects	Public object, where source or target or both are CA certificate objects	USER (CONTROL) or SO (READ)
Copy object	Public object or private object, where source or target or both are CA certificate objects	Private object, where source or target or both are CA certificate objects	USER (CONTROL) or SO (CONTROL) or both USER (UPDATE) and SO (READ)
Copy object	Private object, where source or target or both are CA certificate objects	Public object, where source or target or both are CA certificate objects	USER (CONTROL) or both USER (UPDATE) and SO (READ)

**Note:**

- Session and token objects require the same authority.
- See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the SO and User PKCS #11 roles and on how ICSF determines that a certificate is a CA certificate.

## Usage Notes

HMAC operations are performed in hardware or software.

When creating an object, these attribute processing rules will be in effect:

- All attributes will be processed and the value of the last instance of an attribute in the template will be saved.

When copying an object, these attribute processing rules will be in effect:



## PKCS #11 Token record create

- All attributes will be processed and the value of the last instance of an attribute in the template will be saved except for CKA\_EXTRACTABLE and CKA\_SENSITIVE. CKA\_EXTRACTABLE will be copied from the source object and may be set to False if the value in the source object is True. CKA\_SENSITIVE will be copied from the source object and may be set to True if the value in the source object is False.

---

## PKCS #11 Token record delete (CSFPTRD and CSFPTRD6)

Use the token record delete callable service (CSFPTRD) to delete a z/OS PKCS #11 token, token object, session object, or state object. When a token is deleted, all associated objects are deleted as well. The deletions occur in the token data set (TKDS), and all session memory areas in the ICSF address space.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPTRD6.

### Format

```
CALL CSFPTRD(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    handle,  
    rule_array_count,  
    rule_array)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

#### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

#### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

**handle**

Direction: Input

Type: String

44-byte name of the token or object to be deleted. See “Handles” on page 99 for the format of a *handle*.

**rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords supplied in the *rule\_array* parameter. This value must be 1.

**rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Keyword	Meaning
<b>One of these two keywords must be specified:</b>	
TOKEN	Specifies that a token and all associated objects are to be deleted.
OBJECT	Specifies that an object is to be deleted.

## Authorization

Table 340. Authorization requirements for the token record delete callable service

Token / Object Type	PKCS #11 Role Authority Required
Token	SO (UPDATE)
Public object, except CA certificate	USER (UPDATE) or SO (READ)
Private object, except CA certificate	USER (UPDATE) or SO (CONTROL)
Public CA certificate object	USER (CONTROL) or SO (READ)
Private CA certificate object	USER (CONTROL) or SO (CONTROL)
State object	None

**Note:**

- Session and token objects require the same authority.
- See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the SO and User PKCS #11 roles and how ICSF determines that a certificate is a CA certificate.

## Usage Notes

An application can free state objects allocated by certain PKCS #11 callable services by calling this service. To do so, specify the handle of the state object in the *handle* parameter and “OBJECT ” in the *rule\_array* parameter. For more information on the PKCS #11 callable services that can allocate state objects, refer to:

- “PKCS #11 Secret key decrypt (CSFPSKD and CSFPSKD6)” on page 752 CSFPSKD
- “PKCS #11 Secret key encrypt (CSFPSKE and CSFPSKE6)” on page 756 CSFPSKE

## PKCS #11 Token record delete

- “PKCS #11 One-way hash, sign, or verify (CSFPOWH and CSFPOWH6)” on page 737 CSFPOWH
- “PKCS #11 Generate HMAC (CSFPHMG and CSFPHMG6)” on page 730 CSFPHMG
- “PKCS #11 Verify HMAC (CSFPHMV and CSFPHMV6)” on page 734 CSFPHMV

---

## PKCS #11 Token record list (CSFPTRL and CSFPTRL6)

Use the token record list callable service (CSFPTRL) to:

- Obtain a list of z/OS PKCS #11 tokens. The caller must have SAF authority to the token for a particular token to be listed.
- Obtain a list of token and session objects for a token. Use a search template to restrict the search for specific attributes. The caller must have SAF authority to the token.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPTRL6.

### Format

```
CALL CSFPTRL(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    handle,  
    rule_array_count,  
    rule_array,  
    search_template_length,  
    search_template,  
    list_length,  
    handle_count,  
    output_list)
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

#### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

#### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### exit\_data

Direction: Ignored

Type: String

This field is ignored.

### handle

Direction: Input

Type: String

For tokens, an empty string (blanks) for the first call, or the 44-byte handle of the last token found for subsequent calls.

For objects, the 44-byte handle of the token for the first call, or the 44-byte handle of the last object found for subsequent calls.

See Usage Notes for more information. See "Handles" on page 99 for the format of a *handle*.

### rule\_array\_count

Direction: Input

Type: Integer

The number of keywords supplied in the *rule\_array* parameter. This value must be 1 or 2.

### rule\_array

Direction: Input

Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Keyword	Meaning
<b>Processing entity (required)</b>	
TOKEN	Specifies that the list will contain all tokens to which the caller has SAF access. The <i>search_template</i> parameter is ignored.
OBJECT	Specifies that the list will contain the handles of all objects that match the attributes specified in the <i>search_template</i> parameter and to which the caller has SAF access.
<b>List options (optional, valid only with OBJECT)</b>	
ALL	Specifies that when listing objects, both public and private objects that meet the search criteria should be listed if the caller has SAF authority for the token. There may be no sensitive attributes in the search template. See the Authorization topic for details.

### search\_template\_length

Direction: Input

Type: Integer

The length of the *search\_template* parameter in bytes. The value must be 0 when the TOKEN keyword is specified.

The maximum size in bytes is 32752.

### search\_template

## PKCS #11 Token record list

Direction: Input

Type: String

A list of criteria (attribute values) that an object must meet to be added to the list. If the *search\_template\_length* parameter is 0, no criteria are checked.

See “Attribute List” on page 98 for the format of an *attribute\_list*.

### **list\_length**

Direction: Input/Output

Type: Integer

On input, the length in bytes of the *output\_list* parameter. On output, the number of bytes used for the *output\_list* parameter. If the supplied length is insufficient to hold one record, the *list\_length* parameter is set to the minimum length required for a record.

### **handle\_count**

Direction: Input/Output

Type: Integer

On input, the maximum number of tokens or object handles to return in the list. On output from a successful call (*return\_code* < 8), the actual number of tokens or object handles in the list.

### **output\_list**

Direction: Output

Type: String

A list of token names and descriptions or a list of object handles meeting the search criteria.

## Authorization

To list tokens, the caller must have at least USER (READ) or SO (READ) authority.

Authority to list objects depends on the object's attributes and the search criteria as follows:

- To list secret key or private key objects where sensitive key attributes are specified in the search template, this must be true:
  - The object must be marked CKA\_SENSITIVE=F and CKA\_EXTRACTABLE=T and
  - The caller must have USER (READ) authority
- Otherwise (no sensitive attributes in the search criteria)
  - To list public objects, the caller must have at least USER (READ) or SO (READ) authority
  - To list private objects when the ALL rule array keyword is specified, the caller must have at least USER (READ) or SO (READ) authority
  - To list private objects when the ALL rule array keyword is not specified, the caller must have USER (READ) or SO (CONTROL) authority

Token / Object Type	Sensitive Attributes in search criteria	ALL Rule Specified	PKCS #11 Role Authority Required
Token	N/A	N/A	USER (READ) or SO (READ)
Public object	No	N/A	USER (READ) or SO (READ)
Private object	No	No	USER (READ) or SO (CONTROL)

Token / Object Type	Sensitive Attributes in search criteria	ALL Rule Specified	PKCS #11 Role Authority Required
Private object	No	Yes	USER (READ) or SO (READ)
Secret key or Private key object (public or private object class) CKA_SENSITIVE=F and CKA_EXTRACTABLE=T	Yes	N/A	USER (READ)
Secret key or Private key object (public or private object class) CKA_SENSITIVE=T or CKA_EXTRACTABLE=F	Yes	N/A	N/A (object is not listed)

**Note:**

- Session and token objects require the same authority.
- When the caller does not possess sufficient authority to list a given token or object, that record is skipped. (No information for the token or object is returned.) Processing continues with the next token or object.
- The sensitive attributes are as follows:
  - CKA\_VALUE for a secret key object, Elliptic Curve private key, DSA private key, or Diffie-Hellman private key object.
  - CKA\_PRIVATE\_EXPONENT, CKA\_PRIME\_1, CKA\_PRIME\_2, CKA\_EXPONENT\_1, CKA\_EXPONENT\_2, and CKA\_COEFFICIENT for an RSA private key object.
- See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information on the SO and USER PKCS #11 roles.

## Usage Notes

**For tokens:** On the initial call to get a list of tokens, the *handle* parameter should be all blanks. On subsequent calls, the *handle* parameter should be the last token handle from the *output\_list* returned in the previous call.

The output records are in this format:

Bytes	Description				
0 - 31	Token name				
32 - 63	Manufacturer ID				
64 - 79	Model				
80 - 95	Serial number				
96 - 103	Date that the token information or any token object was last updated, expressed as Coordinated Universal Time (UCT) in the format <i>yyyymmdd</i>				
104 - 111	Time that the token information or any token object was last updated, expressed as Coordinated Universal Time (UCT) in the format <i>hhmmssst</i>				
112 - 115	Flags <table> <tr> <th>Bit</th><th>Meaning when set on</th></tr> <tr> <td>0</td><td>Token is write protected.</td></tr> </table>	Bit	Meaning when set on	0	Token is write protected.
Bit	Meaning when set on				
0	Token is write protected.				

## PKCS #11 Token record list

**For objects:** On the initial call to get a list of object handles matching the search template, the *handle* parameter contains the token handle. On subsequent calls, the *handle* parameter should contain the last object handle from the *output\_list* returned in the previous call. The output records are the 44-byte handles of the objects.

---

## PKCS #11 Unwrap key (CSFPUWK and CSFPUWK6)

Use unwrap key callable service to unwrap and create a key object using another key. The following formatting is supported:

- PKCS 1.2 formatting is supported for a DES, DES3, AES, BLOWFISH, RC4, or GENERIC secret wrapped by an RSA public key.
  - A new secret key object is created with the decrypted key value
  - The unwrapping key must be a private key object
  - The CKA\_UNWRAP attribute must be true
- PKCS 8 formatting (CBC mode with padding) is supported for an RSA, DSA, Elliptic Curve, and Diffie-Hellman private key wrapped by a secret key.
  - A new private key object is created with the decrypted key values
  - The unwrapping key must be a secret key object
  - The CKA\_UNWRAP attribute must be true
  - The encryption mechanism must be specified in the rule array and must match the key type of the secret key object
- IBM Proprietary Attribute Bound format – where the key's usage flags are restored from the wrapped key data

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPUWK6.

### Format

```
CALL CSFPUWK(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    wrapped_key_length,  
    wrapped_key,  
    initialization_vector_length,  
    initialization_vector,  
    unwrapping_key_handle,  
    attribute_list_length,  
    attribute_list,  
    target_key_handle )
```

### Parameters

#### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

#### **reason\_code**

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

This field is ignored. It is recommended to specify 0 for this parameter.

This field is ignored.

The number of keywords you supplied in the *rule\_array* parameter. This value must be 1 or 2.

Keywords that provide control information to the callable service.

Keyword	Meaning
Formatting Method (required)	
ATTRBND	The <i>wrapped_key</i> is an IBM proprietary format. The private or secret key and its usage flags are to be unwrapped together. A signature verification public key handle must be supplied through the <i>initialization_vector</i> parameter.
PKCS-1.2	RSA PKCS #1 block type 02 will be used to recover the key value.
PKCS-8	The private key values are DER encoded as specified by PKCS-8. The encryption mechanism rule array keyword must be specified.
Encryption Mechanism (required when PKCS-8 specified, ignored otherwise)	
AES	For PKCS-8 processing, the unwrapping key must be an AES secret key object.
DES	For PKCS-8 processing, the unwrapping key must be a DES secret key object.
DES3	For PKCS-8 processing, the unwrapping key must be a DES2 or DES3 secret key object.

Length of the wrapped key in the *wrapped\_key* parameter.



## PKCS #11 Unwrap key

### **wrapped\_key**

Direction: Input

Type: String

The key to be unwrapped.

### **initialization\_vector\_length**

Direction: Input

Type: Integer

The length of the *initialization\_vector* parameter. The initial value can only be used with PKCS-8 or ATTRBND. This parameter is ignored for PKCS-1.2. For PKCS-8, the length must match the key type of the wrapping key (8 for DES, DES2, DES3 and 16 for AES). If the length is zero, the *initialization\_vector* parameter is ignored and an initial value of zero is used. For ATTRBND, the length must be 44.

### **initialization\_vector**

Direction: Input

Type: String

For formatting method PKCS-8, this is the initial chaining value for symmetric encryption. The length must match the key type of the wrapping key.

For formatting method ATTRBND, this is the 44-byte handle of the public or secret key object to be used to verify the signature on the key data.

For formatting method PKCS-1.2, this parameter is ignored.

### **unwrapping\_key\_handle**

Direction: Input

Type: String

The 44-byte handle of the private key or secret key object to unwrap the key. See "Handles" on page 99 for the format of a *unwrapping\_key\_handle*.

### **attribute\_list\_length**

Direction: Input

Type: Integer

Length of the *attribute\_list* parameter in bytes. The maximum value for this field is 32750.

### **attribute\_list**

Direction: Input

Type: String

List of token or object attributes for the target key. The attributes must be consistent with the class of the object. See "Attribute List" on page 98 for the format of an *attribute\_list*.

### **target\_key\_handle**

Direction: Output

Type: String

The 44-byte handle of the secret key or private key object created for the unwrapped key. The object will use to token name of the unwrapping key object.

## Authorization

There are two or three keys involved in this service: the unwrapping key and the target key (the new key created from the wrapped key), and (optionally) a signature verification key.

- To use an unwrapping or verification key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To use an unwrapping or verification key that is a private object, the caller must have USER (READ) authority (user access).
- To unwrap a target key that is a public object, the caller must have SO (READ) authority or USER (UPDATE) authority
- To unwrap a target key that is a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority

## Usage Notes

For Attribute Bound unwrapping:

- All keys involved (target, unwrapping, and verification) must have the CKA\_IBM\_ATTRBOUND attribute set TRUE
- The unwrap template is restricted to the following attributes:
  - CKA\_TOKEN
  - CKA\_LABEL
  - CKA\_SUBJECT – For private keys only
  - CKA\_ID
  - CKA\_START\_DATE
  - CKA\_END\_DATE
  - CKA\_APPLICATION
  - CKA\_IBM\_FIPS140
  - CKA\_PRIVATE

---

## PKCS #11 Wrap key (CSFPWPK and CSFPWPK6)

Use wrap key callable service to wrap a key with another key. The following formatting is supported:

- PKCS 1.2 is supported for wrapping a DES, DES3, AES, BLOWFISH, RC4, or GENERIC secret key with an RSA public key.
  - The wrapping key must be a public key object.
  - The CKA\_WRAP attribute must be true.
- PKCS 8 formatting (CBC mode with padding) is supported for wrapping an RSA, DSA, Elliptic Curve, or Diffie-Hellman private key with a secret key.
  - The wrapping key must be a secret key object.
  - The CKA\_WRAP attribute must be true
  - The encryption mechanism must be specified in the rule array and must match the key type of the secret key object
- IBM Proprietary Attribute Bound format – where the key's usage flags are to be included in the wrapped key data

If the length of output field is too short to hold the output, the service will fail and return the required length of the output field in the *wrapped\_key\_length* parameter.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPWPK6.

## Format

```
CALL CSFPWPK(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    source_key_handle,
    wrapping_key_handle,
    initialization_vector_length,
    initialization_vector,
    wrapped_key_length,
    wrapped_key )
```

## Parameters

### **return\_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

### **reason\_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

### **exit\_data\_length**

Direction: Ignored

Type: Integer

This field is ignored. It is recommended to specify 0 for this parameter.

### **exit\_data**

Direction: Ignored

Type: String

This field is ignored.

### **rule\_array\_count**

Direction: Input

Type: Integer

The number of keywords you supplied in the *rule\_array* parameter. This value must be 1 or 2.

### **rule\_array**

Direction: Input

Type: String

Keywords that provide control information to the callable service.

Table 342. Keywords for wrap key

Keyword	Meaning
Formatting Method (required)	
ATTRBND	The private or secret key and its usage flags are to be wrapped together in an IBM proprietary format. A signing key handle must be supplied through the <i>initialization_vector</i> parameter
PKCS-1.2	RSA PKCS #1 block type 02 will be used to format the key value.
PKCS-8	The private key values are DER encoded as specified by PKCS-8. The encryption mechanism rule array keyword must be specified.
Encryption Mechanism (required when PKCS-8 specified, ignored otherwise)	
AES	For PKCS-8 processing, the wrapping key must be an AES secret key object.
DES	For PKCS-8 processing, the wrapping key must be a DES secret key object.
DES3	For PKCS-8 processing, the wrapping key must be a DES2 or DES3 secret key object.

### source\_key\_handle

Direction: Input Type: String

The 44-byte handle of the secret key or private key object to be wrapped.

### wrapping\_key\_handle

Direction: Input Type: String

The 44-byte handle of the public key or secret key object to wrap the secret key. See “Handles” on page 99 for the format of a *wrapping\_key\_handle*.

### Initialization\_vector\_length

Direction: Input Type: Integer

The length of the *initialization\_vector* parameter. The initial value can only be used with PKCS-8 or ATTRBND. This parameter is ignored for PKCS-1.2. For PKCS-8, the length must match the key type of the wrapping key (8 for DES, DES2, DES3 and 16 for AES). If the length is zero, the initialization vector parameter is ignored and a value of zero is used. For ATTRBND the length must be 44.

### Initialization\_vector

Direction: Input Type: String

For formatting method PKCS-8, this is the initial chaining value for symmetric encryption. The length must match the key type of the wrapping key.

For formatting method ATTRBND, this is the 44-byte handle of the private or secret key object to be used to sign the key data.

For formatting method PKCS-1.2, this parameter is ignored.

### wrapped\_key\_length

Direction: Input/Output Type: Integer

On input, the length of the *wrapped\_key* parameter. On output, the actual length

## PKCS #11 wrap key

of the wrapped key returned in the *wrapped\_key* parameter.

### **wrapped\_key**

Direction: Output

Type: String

The wrapped key

## Authorization

There are two or three key objects used by this service, the source key (the key to be wrapped), the wrapping key, and (optionally) a signing key.

- To wrap a source key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To wrap a source key that is a private object, the caller must have USER (READ) authority (user access)
- To use a wrapping or signing key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To use a wrapping or signing key that is a private object, the caller must have USER (READ) authority (user access).

## Usage Notes

Clear keys may not be used to wrap secure keys and vice versa. (See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for the more information on clear keys and secure keys.)

- One exception, clear RSA public keys may be used to perform non-attribute bound wrap of secure secret keys .

For Attribute Bound wrapping, all keys involved (source, wrapping, and signing) must have the CKA\_IBM\_ATTRBOUND attribute set TRUE

---

## Part 4. Appendixes



---

## Appendix A. ICSF and TSS Return and Reason Codes

This topic includes this information:

- Return codes and reason codes issued on the completion of a call to an ICSF callable service
- Return codes and reason codes issued on the completion of a process on a coprocessor or accelerator.
- ICSF return and reason codes can be specified in the installation options data set on the REASONCODES parameter. If the REASONCODES option is not specified, the default of REASONCODES(ICSF) is used. A REASONCODES line in the description indicates a conversion was done as a result of the REASONCODES option in the installation options data set.

If you specified REASONCODES(ICSF) and your service was processed on a PCICC, PCIXCC, or CCA Crypto Express coprocessor, a TSS reason code may be returned if there is no 1–1 corresponding ICSF reason code.

---

### Return Codes and Reason Codes

This topic describes return codes and reason codes.

The TSS return and reason codes have been merged with the ICSF codes in this release. If there is a REASONCODES line in the description, it will indicate an alternate reason code you should investigate.

Each return code returns unique reason codes to your application program. The reason codes associated with each return code are described in these topics. The reason code tables present the hexadecimal code followed by the decimal code in parenthesis.

### Return Codes

Table 343 lists return codes from the ICSF callable services.

*Table 343. Return Codes*

Return Code Hex (Decimal)	Description
Return Code 0 (0)	The call to the service was successfully processed. See the reason code for more information.
Return Code 4 (4)	The call to the service was successfully processed, but some minor event occurred during processing. See the reason code for more information. <b>User action:</b> Review the reason code.
Return Code 8 (8)	The call to the service was unsuccessful. The parameters passed into the call are unchanged, except for the return code and reason code. There are rare examples where output areas are filled, but their contents are not guaranteed to be accurate. These are described under the appropriate reason code descriptions. The reason code identifies which error was found. <b>User action:</b> Review the reason code, correct the problem, and retry the call.



Table 343. Return Codes (continued)

Return Code Hex (Decimal)	Description
Return Code C (12)	<p>The call to the service could not be processed because ICSF was not active, ICSF found something wrong in its environment, a TSS security product is not available, or a processing error occurred in a TSS product. The parameters passed into the call are unchanged, except for the return code and reason code.</p> <p><b>User action:</b> Review the reason code and take the appropriate action.</p>
Return Code 10 (16)	<p>The call to the service could not be processed because ICSF found something seriously wrong in its environment or a processing error occurred in the coprocessor. The parameters passed into the call are unchanged, except for the return code and reason code.</p> <p><b>User action:</b> Review the reason code and contact your system programmer.</p>
Return Code 14 (20)	<p>The call to the service could not be processed because an unexpected error occurred in ICSF's cryptographic software element. The reason codes for this error are not documented.</p> <p><b>User action:</b> Contact your IBM support center.</p>
Return Code 18 (24)	<p>The call to the service could not be processed because an unexpected error occurred in the Crypto Express Enterprise PKCS #11 coprocessor. The reason codes for this error are not documented.</p> <p><b>User action:</b> Contact your IBM support center.</p>
Return Code 19 (25)	<p>The call to the service could not be processed because a vendor specific error occurred in the Crypto Express Enterprise PKCS #11 coprocessor. The reason codes for this error are not documented.</p> <p><b>User action:</b> Contact your IBM support center.</p>

## Reason Codes for Return Code 0 (0)

Table 344 lists reason codes returned from callable services that give return code 0.

Table 344. Reason Codes for Return Code 0 (0)

Reason Code Hex (Decimal)	Description
0 (0)	<p>The call to the ICSF callable service was successfully processed. No error was encountered.</p> <p><b>User action:</b> None.</p>
2 (2)	<p>The call to the ICSF callable service was successfully processed. A minor error was detected. A key used in the service did not have odd parity. This key could be one provided by you as a parameter or be one (perhaps of many) that was retrieved from the in-storage CKDS.</p> <p><b>User action:</b> Refer to the reason code obtained when the key passed to this service was transformed into operational form using clear key import, multiple clear key import, key import, secure key import, or multiple secure key import callable services. Check if any of the services prepared an even parity key. If one of these service reported an even parity key, you need to know which key is affected. If none of these services identified an even parity key, then the even parity key detected was found on the CKDS. Report this to your administrator.</p> <p><b>REASONCODES:</b> ICSF 4 (4)</p>

Table 344. Reason Codes for Return Code 0 (0) (continued)

Reason Code Hex (Decimal)	Description
4 (4)	<p>The call to the ICSF callable service was successfully processed. A minor error was detected. A key used in the service did not have odd parity. This key could be one provided by you as a parameter or be one (perhaps of many) that was retrieved from the in-storage CKDS.</p> <p><b>User action:</b> Refer to the reason code obtained when the key passed to this service was transformed into operational form using clear key import, multiple clear key import, key import, secure key import, or multiple secure key import callable services. Check if any of the services prepared an even parity key. If one of these service reported an even parity key, you need to know which key is affected. If none of these services identified an even parity key, then the even parity key detected was found on the CKDS. Report this to your administrator.</p> <p><b>REASONCODES:</b>TSS 2 (2)</p>
8 (8)	<p>The CKDS key record read callable service attempted to read a NULL key record. The returned key token contains only binary zeros.</p> <p><b>User action:</b> None required.</p>
862 (2146)	<p>The call to the callable service was successfully processed. A key was wrapped by a weaker key. This reason code is returned when either the "Warn when weak wrap - Transport keys" or "Warn when weak wrap - Master keys" access control point is enabled.</p> <p><b>User action:</b> None required. If you wish to prohibit weak key wrapping, enable the access control point "Prohibit weak wrapping - Transport keys" and "Prohibit weak wrapping - Master keys" access control points using the TKE workstation.</p>
BC2 (3010)	<p>The call to CSFIQF was successful. Additionally, the coprocessor adapter is disabled by TKE.</p>
2710 (10000)	<p>The call to the callable service was successfully processed. The keys in one or more key identifiers have been reenciphered from encipherment under the old master key to encipherment under the current master key.</p> <p><b>User action:</b> If you obtained your operational token from a file, replace the token in the file with the token just returned from ICSF.</p> <p>Management of internal tokens is a user responsibility. Consider the possible case where the token for this call was fetched from a file, and where this reason code is ignored. For the next invocation of the service, the token will be fetched from the file again, and the service will give this reason code again. If this continues until the master key is changed again, then the next use of the internal token will fail.</p>
2711 (10001)	<p>The call to the callable service was successfully processed. The keys in one or more key identifiers were encrypted under the old master key. The callable service was unable to reencipher the key.</p>
2713 (10003)	<p>The call to the callable service was successfully processed. Weak key used. The strength of the KEK key is less than the strength of the key to be wrapped.</p> <p>If Access Control Point 'Prohibit weak wrapping - Transport keys' is not enabled, this informational Reason Code will be returned. If Access Control Point 'Prohibit weak wrapping - Transport keys' is enabled you will receive an error from the callable service.</p> <p><b>User action:</b> None.</p>

## Reason Codes for Return Code 4 (4)

Table 345 on page 784 lists reason codes returned from callable services that give return code 4.

Table 345. Reason Codes for Return Code 4 (4)

Reason Code Hex (Decimal)	Description
1 (1)	<p>The verification test failed.</p> <p><b>REASONCODES:</b> This reason code also corresponds to these ICSF reason codes: FA0 (4000), 1F40 (8000), 1F44 (8004), 2328 (9000), 232C (9004), 2AF8 (11000), or 36B8 (14008).</p>
13 (19)	<p>This is a combination reason code value. The call to the Encrypted PIN verify (PINVER) callable service was successfully processed. However, the trial PIN that was supplied does not match the PIN in the PIN block.</p> <p><b>User action:</b> The PIN is incorrect. If you expected the reason code to be zero, check that you are using the correct key.</p> <p><b>REASONCODES:</b> ICSF BD4 (3028)</p> <p>In addition, a key in a key identifier token has been reenciphered.</p> <p><b>User action:</b> See reason code 10000 (return code 0) for more detail about the key reencipherment.</p>
14 (20)	<p>The input text length was odd rather than even. The right nibble of the last byte is padded with X'00'.</p> <p><b>User action:</b> None</p> <p><b>REASONCODES:</b> ICSF 7D0 (2000)</p>
A6 (166)	<p>The control vector is not valid because of parity bits, anti-variant bits, inconsistent KEK bits, or because bits 59 to 62 are not zero.</p>
B3 (179)	<p>The control vector keywords that are in the rule array are ignored.</p>
1AD (429)	<p>The digital signature verify ICSF callable service completed successfully but the supplied digital signature failed verification.</p> <p><b>User action:</b> None</p> <p><b>REASONCODES:</b> ICSF 2AF8 (11000)</p>
7D0 (2000)	<p>The input text length was odd rather than even. The right nibble of the last byte is padded with X'00'.</p> <p><b>User action:</b> None</p> <p><b>REASONCODES:</b> TSS 14 (20)</p>
81E (2078)	<p>The call to CKDS Key Record Read was successful. The key label exists in the CKDS. The key label contains a clear DES or AES key token and is not returned to the caller.</p>
872 (2162)	<p>A weak master key was detected when the final key part was loaded for the DES or RSA master key. A key is weak if any of the three parts are the same as another part. For example, when the first and third key parts are the same, the key is weak (effectively a double-length key).</p> <p><b>User action:</b> Create new key values for the new master key and retry master key entry.</p>
BBA (3002)	<p>The call to the CVV Verify callable service was successfully processed. However, the trial CVV that was supplied does not match the generated CVV. In addition, a key in the key identifier has been reenciphered.</p> <p><b>REASONCODES:</b> See reason code 4000 (return code 4) for more details about the incorrect CVV. See reason code 10000 (return code 0) for more details about the key reencipherment.</p>
BC9 (3017)	<p>The call to create a list of information completed successfully, however the storage supplied for the list was insufficient to hold the complete list.</p>

Table 345. Reason Codes for Return Code 4 (4) (continued)

Reason Code Hex (Decimal)	Description
BD4 (3028)	<p>The call to the Encrypted PIN verify (PINVER) callable service was successfully processed. However, the trial PIN that was supplied does not match the PIN in the PIN block.</p> <p><b>User action:</b> The PIN is incorrect. If you expected the reason code to be zero, check that you are using the correct key.</p> <p><b>REASONCODES:</b> TSS 13 (19)</p>
BD8 (3032)	<p>This is a combination reason code value. The call to the Encrypted PIN verify (PINVER) callable service was successfully processed. However, the trial PIN that was supplied does not match the PIN in the PIN block.</p> <p>In addition, a key in a key identifier token has been reenciphered.</p> <p><b>REASONCODES:</b> See reason code 3028 (return code 4) for more detail about the incorrect PIN. See reason code 10000 (return code 0) for more detail about the key reencipherment.</p>
BFC (3068)	<p>The verification pattern of an encrypted CPACF key block doesn't match the current wrapping key's verification pattern.</p>
FA0 (4000)	<p>The CVV did not verify.</p> <p><b>User action:</b> Regenerate the CVV.</p> <p><b>REASONCODES:</b> TSS 1 (1)</p>
FA4 (4004)	<p>Rewrapping is not allowed for one or more keys.</p>
1F40 (8000)	<p>The call to the MAC verification (MACVER) callable service was successfully processed. However, the trial MAC that you supplied does not match that of the message text.</p> <p><b>User action:</b> The message text may have been modified, such that its contents cannot be trusted. If you expected the reason code to be zero, check that you are using the correct key. Check that all segments of the message were presented and in the correct sequence. Also check that the trial MAC corresponds to the message being authenticated.</p> <p><b>REASONCODES:</b> TSS 1 (1)</p>
1F44 (8004)	<p>This is a combination reason code value. The call to the MAC verification (MACVER) callable service was successfully processed. However, the trial MAC that was supplied does not match the message text provided.</p> <p>In addition, a key in a key identifier token has been reenciphered.</p> <p><b>User action:</b> See reason code 8000 (return code 4) for more detail about the incorrect MAC. See reason code 10000 (return code 0) for more detail about the key reencipherment.</p> <p><b>REASONCODES:</b> TSS 1 (1)</p>
2328 (9000)	<p>The call to the key test service processed successfully, but the key test pattern was not verified.</p> <p><b>User action:</b> Investigate why the key failed. When determining this, you can reinstall or regenerate the key.</p> <p><b>REASONCODES:</b> TSS 1 (1)</p>

Table 345. Reason Codes for Return Code 4 (4) (continued)

Reason Code Hex (Decimal)	Description
232C (9004)	<p>This is a combination reason code value. The call to the key test service processed successfully, but the key test pattern was not verified. Also, the key token has been reenciphered.</p> <p><b>User action:</b> Investigate why the key failed. When determining this, you can reinstall or regenerate the key.</p> <p><b>REASONCODES:</b> TSS 1 (1)</p>
2AF8 (11000)	<p>The digital signature verify ICSF callable service completed successfully but the supplied digital signature failed verification.</p> <p><b>User action:</b> None</p> <p><b>REASONCODES:</b> TSS 1AD (429)</p>
36B8 (14008)	<p>The PKDS record failed the authentication test.</p> <p><b>User action:</b> The record has changed since ICSF wrote it to the PKDS. The user action is application dependent.</p> <p><b>REASONCODES:</b> TSS 1 (1)</p>
8D10 (36112)	<p>CKDS conversion completed successfully but some tokens could not be rewrapped because the control vector prohibited rewrapping from the enhanced wrapping method.</p>

## Reason Codes for Return Code 8 (8)

Table 346 lists reason codes returned from callable services that give return code 8.

Most of these reason codes indicate that the call to the service was unsuccessful. No cryptographic processing took place. Therefore, no output parameters were filled. Exceptions to this are noted in the descriptions.

Table 346. Reason Codes for Return Code 8 (8)

Reason Code Hex (Decimal)	Description
00C (12)	<p>A key identifier was passed to a service or token. It is checked in detail to ensure that it is a valid token, and that the fields within it are valid values. There is a token validation value (TVV) in the token, which is a non-cryptographic value. This value was again computed from the rest of the token, and compared to the stored TVV. If these two values are not the same, this reason code is returned.</p> <p><b>User action:</b> The contents of the token have been altered because it was created by ICSF or TSS. Review your program to see how this could have been caused.</p>
016 (22)	The ID number in the request field is not valid. The PAN data is incorrect for VISA CVV.
017 (23)	Offset length not correct for data to be inserted.
018 (24)	<p>A key identifier was passed to a service. The master key verification pattern in the token shows that the key was created with a master key that is neither the current master key nor the old master key. Therefore, it cannot be reenciphered to the current master key.</p> <p><b>User action:</b> Re-import the key from its importable form (if you have it in this form), or repeat the process you used to create the operational key form. If you cannot do one of these, you cannot repeat any previous cryptographic process that you performed with this token.</p> <p><b>REASONCODES:</b> ICSF 2714 (10004)</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
019 (025)	<p>A length parameter has an incorrect value. The value in the length parameter could have been zero (when a positive value was required) or a negative value. If the supplied value was positive, it could have been larger than your installation's defined maximum, or for MDC generation with no padding, it could have been less than 16 or not an even multiple of 8.</p> <p><b>User action:</b> Check the length you specified. If necessary, check your installation's maximum length with your ICSF administrator. Correct the error.</p>
01D (29)	<p>A key identifier was passed to a service or token. It is checked in detail to ensure that it is a valid token, and that the fields within it are valid values. There is a token validation value (TVV) in the token, which is a non-cryptographic value. This value was again computed from the rest of the token, and compared to the stored TVV. If these two values are not the same, this reason code is returned.</p> <p><b>User action:</b> The contents of the token have been altered because it was created by ICSF or TSS. Review your program to see how this could have been caused.</p> <p><b>REASONCODES:</b> ICSF 2710 (10000)</p>
01E (30)	<p>A key label was supplied for a key identifier parameter. This label is the label of a key in the in-storage CKDS or the PKDS. Either the key could not be found, or a key record with that label and the specific type required by the ICSF callable service could not be found. For a retained key label, this error code is also returned if the key is not found in the PCICC, PCIXCC, or CCA Crypto Express coprocessor specified in the PKDS record.</p> <p><b>User action:</b> Check with your administrator if you believe that this key should be in the in-storage CKDS or the PKDS. The administrator may be able to bring it into storage. If this key cannot be in storage, use a different label.</p> <p><b>REASONCODES:</b> ICSF 271C (10012)</p>
01F (31)	<p>The control vector did not specify a DATA key.</p> <p><b>REASONCODES:</b> ICSF 272C (10028)</p>
020 (32)	<p>You called the CKDS key record create callable service, but the <i>key_label</i> parameter syntax was incorrect.</p> <p><b>User action:</b> Correct <i>key_label</i> syntax.</p> <p><b>REASONCODES:</b> ICSF 3EA0 (16032)</p>
021 (33)	<p>The <i>rule_array</i> parameter contents or a parameter value is not correct.</p> <p><b>User action:</b> Refer to the <i>rule_array</i> parameter described in this publication under the appropriate callable service for the correct value.</p> <p><b>REASONCODES:</b> ICSF 7E0 (2016)</p>
022 (34)	<p>A <i>rule_array</i> keyword combination is not valid.</p> <p><b>REASONCODES:</b> ICSF 7E0 (2016)</p>
023 (35)	<p>The <i>rule_array_count</i> parameter contains a number that is not valid.</p> <p><b>User action:</b> Refer to the <i>rule_array_count</i> parameter described in this publication under the appropriate callable service for the correct value.</p> <p><b>REASONCODES:</b> ICSF 7DC (2012)</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
027 (39)	<p>A control vector violation occurred.</p> <p><b>REASONCODES:</b> This reason code also corresponds to these ICSF reason codes: 272C (10028), 2730 (10032), 2734 (10036), 2744 (10052), 2768 (10088), 278C (10124), 3E90 (16016), 2724 (10020).</p>
028 (40)	<p>The service code does not contain numerical data.</p> <p><b>REASONCODES:</b> ICSF BE0 (3040)</p>
029 (41)	<p>The <i>key_form</i> parameter is neither IM nor OP. Most constants, these included, can be supplied in lower or uppercase. Note that this parameter is 4 bytes long, so the value IM or OP is not valid. They must be padded on the right with blanks.</p> <p><b>User action:</b> Review the value provided and change it to IM or OP, as required.</p>
02A (42)	<p>The expiration date is not numeric (X'F0' through X'F9'). The parameter must be character representations of numerics or hexadecimal data.</p> <p><b>User action:</b> Review the numeric parameters or fields required in the service that you called and change to the format and values required.</p> <p><b>REASONCODES:</b> ICSF BE0 (3040)</p>
02B (43)	<p>The value specified for the <i>key_length</i> parameter of the key generate callable service is not valid.</p> <p><b>User action:</b> Review the value provided and change it as appropriate.</p> <p><b>REASONCODES:</b> See also the ICSF reason code 80C (2060) or 2710 (10000) for additional information.</p>
02C (44)	<p>The CKDS key record create callable service requires that the key created not already exist in the CKDS. A key of the same label was found.</p> <p><b>User action:</b> Make sure the application specifies the correct label. If the label is correct, contact your ICSF security administrator or system programmer.</p>
02D (45)	<p>An input character is not in the code table.</p> <p><b>User action:</b> Correct the code table or the source text.</p>
02F (47)	<p>A source key token is unusable because it contains data that is not valid or undefined.</p> <p><b>REASONCODES:</b> This reason code also corresponds to these ICSF reason codes: 83C (2108), 2754 (10068), 2758 (10072), 275C (10076), 2AFC (11004), 2B04 (11012), 2B08 (11016), 2B10 (11024). Please see those reason codes for additional information.</p>
030 (48)	<p>One or more keys has a master key verification pattern that is not valid.</p> <p>This reason code also corresponds to these ICSF reason codes: 2714 (10004) and 2B0C (11020). Please see those reason codes for additional information.</p>
031 (49)	<p>Key identifiers contain a version number. The version number in a supplied key identifier (internal or external) is inconsistent with one or more fields in the key identifier, making the key identifier unusable.</p> <p><b>User action:</b> Use a token containing the required version number.</p> <p><b>REASONCODES:</b> ICSF 2738 (10040)</p>



Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
033 (51)	<p>The encipher and decipher callable services sometime require text (plaintext or ciphertext) to have a length that is an exact multiple of 8 bytes. Padding schemes always create ciphertext with a length that is an exact multiple of 8. If you want to decipher ciphertext that was produced by a padding scheme, and the text length is not an exact multiple of 8, then an error has occurred. The CBC mode of enciphering requires a text length that is an exact multiple of 8.</p> <p>The ciphertext translate callable service cannot process ciphertext whose length is not an exact multiple of 8.</p> <p>The value that the <i>text_length</i> parameter specifies is not a multiple of the cryptographic algorithm block length.</p> <p><b>User action:</b> Review the requirements of the service you are using. Either adjust the text you are processing or use another process rule.</p>
038 (56)	The master key verification pattern in the OCV is not valid.
03D (61)	<p>The keyword supplied with the <i>key_type</i> parameter is not valid.</p> <p><b>REASONCODES:</b> This reason code also corresponds to these ICSF reason codes: 2720 (10016), 2740 (10048), 274C (10060). Please see those reason codes for additional information.</p>
03E (62)	<p>The source key was not found.</p> <p><b>REASONCODES:</b> ICSF 271C (10012)</p>
03F (63)	<p>This check is based on the first byte in the key identifier parameter. The key identifier provided is either an internal token, where an external or null token was required; or an external or null token, where an internal token was required. The token provided may be none of these, and, therefore, the parameter is not a key identifier at all. Another cause is specifying a <i>key_type</i> of IMP-PKA for a key in importable form.</p> <p><b>User action:</b> Check the type of key identifier required and review what you have provided. Also check that your parameters are in the required sequence.</p> <p><b>REASONCODES:</b> ICSF 7F8 (2040)</p>
040 (64)	<p>The supplied private key can be used only for digital signature. Key management services are disallowed.</p> <p><b>User action:</b> Supply a key with key management enabled.</p> <p>OR</p> <p>This service requires an RSA private key that is for signature use. The specified key may be used for key management purposes only.</p> <p><b>User action:</b> Re-invoke the service with a supported private key.</p> <p>OR</p> <p>This service requires an RSA private key that is translatable. The specified key may not be used in the PKA Key Translate callable service.</p> <p><b>User action:</b> Re-invoke the service with a supported private key. To make a key translatable, XLATE-OK must be turned on.</p>
041 (65)	<p>The RSA public or private key specified a modulus length that is incorrect for this service.</p> <p><b>User action:</b> Re-invoke the service with an RSA key with the proper modulus length.</p> <p><b>REASONCODES:</b> ICSF 2B18 (11032) and 2B58 (11096)</p>



Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
042 (66)	<p>The recovered encryption block was not a valid PKCS-1.2 or zero-pad format. (The format is verified according to the recovery method specified in the rule-array.) If the recovery method specified was PKCS-1.2, refer to PKCS-1.2 for the possible error in parsing the encryption block.</p> <p><b>User action:</b> Ensure that the parameters passed to CSNDSYI or CSNFSYI are correct. Possible causes for this error are incorrect values for the RSA private key or incorrect values in the <i>RSA_enciphered_key</i> parameter, which must be formatted according to PKCS-1.2 or zero-pad rules when created.</p> <p><b>REASONCODES:</b> ICSF 2B20 (11040)</p>
043 (67)	DES or RSA encryption failed.
044 (68)	DES or RSA decryption failed.
046 (70)	<p>Identifier tag for optional block is invalid: conflicts with IBM reserved tag, is a duplicate to a tag already found, is bad in combination with a tag already found when parsing a section of optional blocks, or is otherwise invalid.</p> <p><b>User action:</b> Check the TR-31 key block header for correctness.</p>
048 (72)	<p>The value specified for length parameter for a key token, key, or text field is not valid.</p> <p><b>User action:</b> Correct the appropriate length field parameter.</p> <p><b>REASONCODES:</b> This reason code also corresponds to these ICSF reason codes: 2AF8 (11000) and 2B14 (11028). Please see those reason codes for additional information.</p>
05A (90)	<p>Access is denied for this request. This is due to an access control point in the ICSF role either being disabled or an access control point being enabled that restricts the use of a parameter such as a rule array keyword.</p> <p><b>User action:</b> Check the reference information for the callable service to determine which access control points are involved in the request. Contact the ICSF administrator to determine if the access control points are in the correct state. The access control points can be enabled/disabled using the TKE workstation.</p>
064 (100)	<p>A request was made to the Clear PIN generate or Encrypted PIN verify callable service, and the <i>PIN_length</i> parameter has a value outside the valid range. The valid range is from 4 to 16, inclusive.</p> <p><b>User action:</b> Correct the value in the <i>PIN_length</i> parameter to be within the valid range from 4 to 16.</p> <p><b>REASONCODES:</b> ICSF BBC (3004)</p>
065 (101)	<p>A request was made to the Clear PIN generate callable service, and the <i>PIN_check_length</i> parameter has a value outside the valid range. The valid range is from 4 to 16, inclusive.</p> <p><b>User action:</b> Correct the value in the <i>PIN_check_length</i> parameter to be within the valid range from 4 to 16.</p> <p><b>REASONCODES:</b> ICSF BC0 (3008)</p>
066 (102)	<p>The value of the decimalization table is not valid.</p> <p><b>REASONCODES:</b> ICSF BE0 (3040)</p>
067 (103)	<p>The value of the validation date is not valid.</p> <p><b>REASONCODES:</b> ICSF BE0 (3040)</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
068 (104)	<p>The value of the customer-selected PIN is not valid or the PIN length does not match the value specified.</p> <p><b>REASONCODES:</b> ICSF BE0 (3040)</p>
069 (105)	<p>A request was made to the Clear PIN generate callable service, and the <i>PIN_check_length</i> parameter has a value outside the valid range. The valid range is from 4 to 16, inclusive.</p> <p><b>User action:</b> Correct the value in the <i>PIN_check_length</i> parameter to be within the valid range from 4 to 16.</p> <p><b>REASONCODES:</b> ICSF BE0 (3040)</p>
06A (106)	<p>A request was made to the Encrypted PIN Translate or the Encrypted PIN verify callable service, and the PIN block value in the <i>input_PIN_profile</i> or <i>output_PIN_profile</i> parameter has a value that is not valid.</p> <p><b>User action:</b> Correct the PIN block value.</p>
06B (107)	<p>A request was made to the Encrypted PIN Translate callable service and the format control value in the <i>input_PIN_profile</i> or <i>output_PIN_profile</i> parameter has a value that is not valid. The valid values are NONE or PBVC.</p> <p><b>User action:</b> Correct the format control value to either NONE or PBVC.</p>
06C (108)	<p>The value of the PAD data is not valid.</p> <p><b>REASONCODES:</b> ICSF B08 (3016)</p>
06D (109)	<p>The extraction method keyword is not valid.</p>
06E (110)	<p>The value of the PAD data is not numeric character date.</p> <p><b>REASONCODES:</b> ICSF BE0 (3040)</p>
06F (111)	<p>A request was made to the Encrypted PIN Translate callable service. The <i>sequence_number</i> parameter was required, but was not the integer value 99999.</p> <p><b>User action:</b> Specify the integer value 99999.</p>
074 (116)	<p>The supplied PIN value is incorrect.</p> <p><b>User action:</b> Correct the PIN value.</p> <p><b>REASONCODES:</b> ICSF BBC (3004)</p>
079 (121)	<p>The <i>source_key_identifier</i> or <i>inbound_key_identifier</i> you supplied is not a valid string.</p> <p><b>User action:</b> In an ANSI X9.17 service, check that you specified a valid ASCII string for the <i>source_key_identifier</i> or <i>inbound_key_identifier</i> parameter. In the PKA key generate service, an invalid exponent or modulus length was specified.</p>
07A (122)	<p>The <i>outbound_KEK_count</i> or <i>inbound_KEK_count</i> you supplied is not a valid ASCII hexadecimal string.</p> <p><b>User action:</b> Check that you specified a valid ASCII hexadecimal string for the <i>outbound_KEK_count</i> or <i>inbound_KEK_count</i> parameter.</p>
081 (129)	<p>A Required Rule Array keyword was not specified.</p> <p><b>User action:</b> Refer to the <i>rule_array</i> parameter described in this publication under the appropriate callable service for the correct value.</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
09A (154)	<p>This check is based on the first byte in the key identifier parameter. The key identifier provided is either an internal token, where an external or null token was required; or an external or null token, where an internal token was required. The token provided may be none of these, and, therefore, the parameter is not a key identifier at all. Another cause is specifying a <i>key_type</i> of IMP-PKA for a key in importable form.</p> <p><b>User action:</b> Check the type of key identifier required and review what you have provided. Also check that your parameters are in the required sequence.</p> <p><b>REASONCODES:</b> ICSF 7F8 (2040)</p>
09B (155)	<p>The value that the <i>generated_key_identifier</i> parameter specifies is not valid, or it is not consistent with the value that the <i>key_form</i> parameter specifies.</p>
09C (156)	<p>A keyword is not valid with the specified parameters.</p> <p><b>REASONCODES:</b> ICSF 2790 (10128)</p>
09D (157)	<p>The <i>rule_array</i> parameter contents are incorrect.</p> <p><b>User action:</b> Refer to the <i>rule_array</i> parameter described in this publication under the appropriate callable service for the correct value.</p> <p><b>REASONCODES:</b> ICSF 7E0 (2016)</p>
09F (159)	<p>A parameter requires Rule Array keyword that is not specified.</p> <p><b>User action:</b> Refer to the <i>rule_array</i> parameter described in this publication under the appropriate callable service for the correct value.</p>
0A0 (160)	<p>The <i>key_type</i> and the <i>key_length</i> are not consistent.</p> <p><b>User action:</b> Review the <i>key_type</i> parameter provided and match it with the <i>key_length</i> parameter.</p>
A2 (162)	<p>A request was made to the Remote Key Export callable service, and the <i>certificate_parms</i> parameter contains incorrect values. One or more of the offsets and/or lengths for the modulus, public exponent, and/or digital signature would indicate overlap between two or all three of the fields within the <i>certificate</i> parameter.</p> <p><b>User Action:</b> Correct the values in the <i>certificate_parms</i> parameter to indicate the actual offsets and lengths of the modulus, public exponent, and digital signature within the <i>certificate</i> parameter.</p>
A4 (164)	<p>Two parameters (perhaps the plaintext and ciphertext areas, or <i>text_in</i> and <i>text_out</i> areas) overlap each other. That is, some part of these two areas occupy the same address in memory. This condition cannot be processed.</p> <p><b>User action:</b> Determine which two areas are responsible, and redefine their positions in memory.</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
0A5 (165)	<p>The contents of a chaining vector passed to a callable service are not valid. If you called the MAC generation callable service, or the MDC generation callable service with a MIDDLE or LAST segmenting rule, the count field has a number that is not valid. If you called the MAC verification callable service, then this will have been a MIDDLE or LAST segmenting rule.</p> <p><b>User action:</b> Check to ensure that the chaining vector is not modified by your program. The chaining vector returned by ICSF should only be used to process one message set, and not intermixed between alternating message sets. This means that if you receive and process two or more independent message streams, each should have its own chaining vector. Similarly, each message stream should have its own key identifier.</p> <p>If you use the same chaining vector and key identifier for alternating message streams, you will <b>not</b> get the correct processing performed.</p> <p><b>REASONCODES:</b> ICSF 7F4 (2036)</p>
0B4 (180)	<p>A null key token was passed in the key identifier parameter. When the key type is TOKEN, a valid token is required.</p> <p><b>User action:</b> Supply a valid token to the key identifier parameter.</p>
0B5 (181)	<p>This check is based on the first byte in the key identifier parameter. The key identifier provided is either an internal token, where an external or null token was required; or an external or null token, where an internal token was required. The token provided may be none of these, and, therefore, the parameter is not a key identifier at all. Another cause is specifying a <i>key_type</i> of IMP-PKA for a key in importable form.</p> <p><b>User action:</b> Check the type of key identifier required and review what you have provided. Also check that your parameters are in the required sequence.</p> <p>This reason code also corresponds to these ICSF reason codes: 7F8 (2040), 2B24 (11044) and 3E98 (16024). Please see those reason codes for additional information.</p>
0B7 (183)	<p>A cross-check of the control vector the key type implies has shown that it does not correspond with the control vector present in the supplied internal key identifier.</p> <p><b>User action:</b> Change either the key type or key identifier.</p> <p><b>REASONCODES:</b> ICSF 273C (10044)</p>
0B8 (184)	An input pointer is null.
0CC (204)	A memory allocation failed.
14F (335)	The requested function is not implemented on the coprocessor.
154 (340)	One of the input control vectors has odd parity.
157 (343)	Either the data block or the buffer for the block is too small.
159 (345)	Insufficient storage space exists for the data in the data block buffer.
15A (346)	The requested command is not valid in the current state of the cryptographic hardware component.
176 (374)	<p>Less data was supplied than expected or less data exists than was requested.</p> <p><b>REASONCODES:</b> ICSF 7D4 (2004) and ICSF 7E0 (2016)</p>
181 (385)	The cryptographic hardware component reported that the data passed as part of the command is not valid for that command.

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
197 (407)	A PIN block consistency check error occurred.  <b>REASONCODES:</b> ICSF BC8 (3016)
1B9 (441)	One or more input parameters indicates the key to be processed should be partial, but the key is not partial according to the CV or other control bits of the key.  <b>User action:</b> Check that the partial key option of any input parameters is consistent with the partial key setting of any key tokens being used.
1BA (442)	A DES key supplied in a key identifier parameter has replicated key values (the left and right key values are the same). The key cannot be used in the service called.  <b>User action:</b> Supply a key that doesn't have replicated key values.
25D (605)	The number of output bytes is greater than the number that is permitted.
2BF (703)	A new master key value was found to be one of the weak DES keys.
2C0 (704)	The new master key would have the same master key verification pattern as the current master key.
2C1 (705)	The same key-encrypting key was specified for both exporter keys.
2C2 (706)	While deciphering ciphertext that had been created using a padding technique, it was found that the last byte of the plaintext did not contain a valid count of pad characters.  Note that some cryptographic processing has taken place, and the <i>clear_text</i> parameter may contain some or all of the deciphered text.  <b>User action:</b> The <i>text_length</i> parameter was not reduced. Therefore, it contains the length of the base message, plus the length of the padding bytes and the count byte. Review how the message was padded prior to being enciphered. The count byte that is not valid was created prior to the message's encipherment.  You may need to check whether the ciphertext was not created using a padding scheme. Otherwise, check with the creator of the ciphertext on the method used to create it. You could also look at the plaintext to review the padding scheme used, if any.  <b>REASONCODES:</b> ICSF 7EC (2028)
2C3 (707)	The master key registers are not in the state required for the requested function.  <b>User action:</b> Contact your ICSF administrator.
2CA (714)	A reserved parameter was not a null pointer or an expected value.  <b>REASONCODES:</b> ICSF 844 (2116)
2CB (715)	You supplied a <i>pad_character</i> that is not valid for a Transaction Security System compatibility parameter for which ICSF supports only one value; or, you supplied a KEY keyword and a non-zero <i>master_key_version_number</i> in the Key Token Build service; or, you supplied a non-zero regeneration data length for a DSS key in the PKA Generate service.  <b>User action:</b> Check that you specified the valid value for the TSS compatibility parameter.  <b>REASONCODES:</b> ICSF 834 (2100)
2CF (719)	The RSA-OAEP block did not verify when it decomposed. The block type is incorrect (must be X'03').  <b>User action:</b> Recreate the RSA-OAEP block.  <b>REASONCODES:</b> ICSF 2B38 (11064)

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
2D0 (720)	<p>The RSA-OAEP block did not verify when it decomposed. The random number I is not correct (must be non-zero with the high-order bit equal to zero).</p> <p><b>User action:</b> Recreate the RSA-OAEP block.</p> <p><b>REASONCODES:</b> ICSF 2B40 (11072)</p>
2D1 (721)	<p>The RSA-OAEP block did not verify when it decomposed. The verification code is not correct (must be all zeros).</p> <p><b>User action:</b> Recreate the RSA-OAEP block.</p> <p><b>REASONCODES:</b> ICSF 2BC3 (11068)</p>
2F8 (760)	<p>The RSA public or private key specified a modulus length that is incorrect for this service.</p> <p><b>User action:</b> Re-invoke the service with an RSA key with the proper modulus length.</p> <p><b>REASONCODES:</b> ICSF 2B48 (11080)</p>
302 (770)	<p>A reserved field in a parameter, probably a key identifier, has a value other than zero.</p> <p><b>User action:</b> Key identifiers should not be changed by application programs for other uses. Review any processing you are performing on key identifiers and leave the reserved fields in them at zero.</p> <p>This reason code also corresponds to these ICSF reason codes: 7E8 (2024) and 2B00 (11008). Please see those reason codes for additional information.</p> <p><b>REASONCODES:</b> ICSF 2B00 (11008)</p>
30F (783)	<p>The command is not permitted by the Function Control Vector value.</p> <p><b>REASONCODES:</b> ICSF Return code 12, reason code 2B0C (11020)</p>
401 (1025)	Registered public key or retained private key name already exists.
402 (1026)	Registered public key or retained private key name does not exist.
405 (1029)	There is an error in the Environment Identification data.
40B (1035)	<p>The signature does not match the certificate signature during an RKX call.</p> <p><b>User Action:</b> Check that the key used to check the signatures is the correct.</p>
41A (1050)	A KEK RSA-enciphered at this node (EID) cannot be imported at this same node.
41C (1052)	<p>Token identifier of the trusted block's header section is in the range 0x20 and 0xFF.</p> <p><b>User Action:</b> Check the token identifier of the trusted block.</p>
41D (1053)	<p>The Active flag in the trusted block's trusted block section 0x14 is not disabled.</p> <p><b>User Action:</b> Use the trusted block create callable service to create an inactive/external trusted block.</p>
41E (1054)	<p>Token identifier of the trusted block's header section is not 0x1E (external).</p> <p><b>User Action:</b> Use the trusted block create callable service to create an inactive/external trusted block.</p>
41F (1055)	<p>The Active flag of the trusted block's trusted block section 0x14 is not enabled.</p> <p><b>User Action:</b> Use the trusted block create callable service to create an active/external trusted block.</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
420 (1056)	Token identifier of the trusted block's header section is not 0x1F (internal). <b>User Action:</b> Use the PKA public key import callable service to import the trusted block.
421 (1057)	Trusted block rule section 0x12 Rule ID does not match input parameter rule ID. <b>User Action:</b> Verify the trusted block used has the rule section specified.
422 (1058)	Trusted block contains a value that is too small/too large.
423 (1059)	A trusted block parameter that must have a value of zero (or a grouping of bits set to zero) is invalid.
424 (1060)	Trusted block public key section failed consistency checking.
425 (1061)	Trusted block contains extraneous sections or subsections (TLVs). <b>User Action:</b> Check the trusted block for undefined sections of subsections.
426 (1062)	Trusted block contains missing sections or subsections (TLVs). <b>User Action:</b> Check the trusted block for required sections and subsections applicable to the callable service invoked.
427 (1063)	Trusted block contains duplicate sections or subsections (TLVs). <b>User Action:</b> Check the trusted block's sections and subsections for duplicates. Multiple rule sections are allowed.
428 (1064)	Trusted block expiration date has expired (as compared to the 4764 clock). <b>User Action:</b> Validate the expiration date in the trusted block's trusted information section's Activation and Expiration Date TLV Object.
429 (1065)	Trusted block expiration date is at a date prior to the activation date. <b>User Action:</b> Validate the expiration date in the trusted block's trusted information section's Activation and Expiration Date TLV Object.
42A (1066)	Trusted Block Public Key Modulus bit length is not consistent with the byte length. The bit length must be less than or equal to byte length * 8 and greater than (byte length - 1) * 8.
42B (1067)	Trusted block Public Key Modulus Length in bits exceeds the maximum allowed bit length as defined by the Function Control Vector.
42C (1068)	One or more trusted block sections or TLV Objects contained data which is invalid (an example would be invalid label data in label section 0x13).
42D (1069)	Trusted block verification was attempted by a function other than CSNDDSV, CSNDKTC, CSNDKPI, CSNDRKX, or CSNDTBC.
42E (1070)	Trusted block rule ID contained within a Rule section contains invalid characters.
42F (1071)	The source key's length or CV does not match what is expected by the rule section in the trusted block that was selected by the rule ID input parameter.
430 (1072)	The activation data is not valid. <b>User Action:</b> Validate the activation data in the trusted block's trusted information section's Activation and Expiration Date TLV Object.
431 (1073)	The source-key label does not match the template in the export key DES token parameters TLV object of the selected trusted block rule section.
432 (1074)	The control-vector value specified in the common export key parameters TLV object in the selected rule section of the trusted block contains a control vector that is not valid.



Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
433 (1075)	The source-key label template in the export key DES token parameters TLV object in the selected rule section of the trusted block contains a label template that is not valid.
7D1 (2001)	TKE: DH generator is greater than the modulus.
7D2 (2002)	TKE: DH registers are not in a valid state for the requested operation.
7D3 (2003)	TKE: TSN does not match TSN in pending change buffer.
7D4 (2004)	<p>A length parameter has an incorrect value. The value in the length parameter could have been zero (when a positive value was required) or a negative value. If the supplied value was positive, it could have been larger than your installation's defined maximum, or for MDC generation with no padding, it could have been less than 16 or not an even multiple of 8.</p> <p><b>User action:</b> Check the length you specified. If necessary, check your installation's maximum length with your ICSF administrator. Correct the error.</p> <p><b>REASONCODES:</b> TSS 019 (025)</p>
7D5 (2005)	TKE: PCB data exceeds maximum data length.
7D8 (2008)	<p>Two parameters (perhaps the plaintext and ciphertext areas, or <i>text_in</i> and <i>text_out</i> areas) overlap each other. That is, some part of these two areas occupy the same address in memory. This condition cannot be processed.</p> <p><b>User action:</b> Determine which two areas are responsible, and redefine their positions in memory.</p> <p><b>REASONCODES:</b> TSS 0A4 (164)</p>
7D9 (2009)	TKE: ACI can not load both loads and profiles in one call.
7DA (2010)	TKE: ACI can only load one role or one profile at a time.
7DB (2011)	TKE: DH transport key algorithm match.
7DC (2012)	<p>The <i>rule_array_count</i> parameter contains a number that is not valid.</p> <p><b>User action:</b> Refer to the <i>rule_array_count</i> parameter described in this publication under the appropriate callable service for the correct value.</p> <p><b>REASONCODES:</b> TSS 023 (035)</p>
7DD (2013)	TKE: Length of hash pattern for keypart is not valid for DH transport key algorithm specified.
7DE (2014)	TKE: PCB buffer is empty.
7DF (2015)	An error occurred in the Domain Manager.
7E0 (2016)	<p>The <i>rule_array</i> parameter contents are incorrect. One or more of the rules specified are not valid for this service OR some of the rules specified together may not be combined.</p> <p><b>User action:</b> Refer to the <i>rule_array</i> parameter described in this publication under the appropriate callable service for the correct value.</p>
7E2 (2018)	<p>The <i>form</i> parameter specified in the random number generate callable service should be ODD, EVEN, or RANDOM. One of these values was not supplied.</p> <p><b>User action:</b> Change your parameter to use one of the required values for the <i>form</i> parameter.</p> <p><b>REASONCODES:</b> TSS 021 (033)</p>
7E3 (2019)	TKE: Signature in request CPRB did not verify.



Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
7E4 (2020)	TKE: TSN in request CPRB is not valid.
7E8 (2024)	<p>A reserved field in a parameter, probably a key identifier, has a value other than zero.</p> <p><b>User action:</b> Key identifiers should not be changed by application programs for other uses. Review any processing you are performing on key identifiers and leave the reserved fields in them at zero.</p>
7EB (2027)	TKE: DH transport key hash pattern doesn't match.
7EC (2028)	<p>While deciphering ciphertext that had been created using a padding technique, it was found that the last byte of the plaintext did not contain a valid count of pad characters. Note that all cryptographic processing has taken place, and the <i>clear_text</i> parameter contains the deciphered text.</p> <p>When deciphering ciphertext that had been created using Galois/Counter Mode (GCM) either through PKCS #11 Secret key decrypt (CSFPSKD or CSFPSKD6) or Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD, or CSNESYD1), the GCM tag provided did not match the data provided. No cleartext was returned.</p> <p><b>User action:</b> The <i>text_length</i> parameter was not reduced. Therefore, it contains the length of the base message, plus the length of the padding bytes and the count byte. Review how the message was padded prior to it being enciphered. The count byte that is not valid was created prior to the message's encipherment.</p> <p>You may need to check whether the ciphertext was not created using a padding scheme. Otherwise, check with the creator of the ciphertext on the method used to create it. You could also look at the plaintext to review the padding scheme used, if any.</p> <p>If using GCM, verify that the parameters provided (ciphertext, additional authenticated data, and tag) match those provided to, or returned from, the corresponding call to PKCS #11 Secret key encrypt (CSFPSKE or CSFPSKE6) or Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE, or CSNESYE1).</p> <p><b>REASONCODES:</b> TSS 2C2 (706)</p>
7ED (2029)	TKE: Request data block hash does not match hash in CPRB.
7EE (2030)	TKE: DH supplied hash length is not correct.
7EF (2031)	Reply data block too large.
7F0 (2032)	<p>The <i>key_form</i>, <i>key_type_1</i>, and <i>key_type_2</i> parameters for the key generate callable service form a combination, a three-element string. This combination is checked against all valid combinations. Your combination was not found among this list.</p> <p><b>User action:</b> Check the allowable combinations described for each parameter in Key Generate callable service and correct the appropriate parameter(s).</p>
7F1 (2033)	TKE: Change type does not match PCB change type.

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
7F4 (2036)	<p>The contents of a chaining vector or the chaining data passed to a callable service are not valid. If you called the MAC generation callable service, or the MDC generation callable service with a MIDDLE or LAST segmenting rule, the count field has a number that is not valid. If you called the MAC verification callable service, then this will have been a MIDDLE or LAST segmenting rule. If you called the Symmetric Key Encipher, Symmetric Key Decipher, PKCS#11 Secret Key Encrypt or PKCS #11 Secret Key Decrypt, the chaining data passed is unusable, either because a CONTINUE or FINAL was not preceded by an INITIAL or CONTINUE, or because an attempt was made to continue chaining calls after a partial block has been processed.</p> <p><b>User action:</b> Check to ensure that the chaining vector or chaining data is not modified by your program. The chaining vector or chaining data returned by ICSF should only be used to process one message set, and not intermixed between alternating message sets. This means that if you receive and process two or more independent message streams, each should have its own chaining vector. Similarly, each message stream should have its own key identifier.</p> <p>If you use the same chaining vector and key identifier for alternating message streams, you will <b>not</b> get the correct processing performed.</p> <p><b>REASONCODES:</b> TSS 0A5 (165)</p>
7F6 (2038)	<p>No RSA private key information was provided in the supplied token.</p> <p><b>User action:</b> Check that the token supplied was of the correct type for the service.</p>
7F8 (2040)	<p>This check is based on the first byte in the key identifier parameter. The key identifier provided is either an internal token, where an external or null token was required; or an external or null token, where an internal token was required. The token provided may be none of these, and, therefore, the parameter is not a key identifier at all. Another cause is specifying a <i>key_type</i> of IMP-PKA for a key in importable form.</p> <p><b>User action:</b> Check the type of key identifier required and review what you have provided. Also check that your parameters are in the required sequence.</p> <p><b>REASONCODES:</b> TSS 03F (063) and TSS 09A (154)</p>
7FC (2044)	The caller must be in task mode, not SRB mode.
800 (2048)	<p>The <i>key_form</i> is not valid for the <i>key_type</i></p> <p><b>User action:</b> Review the <i>key_form</i> and <i>key_type</i> parameters. For a <i>key_type</i> of IMP-PKA, the secure key import callable service supports only a <i>key_form</i> of OP.</p>
802 (2050)	<p>A UKPT keyword was specified, but there is an error in the <i>PIN_profile</i> key serial number.</p> <p><b>User action:</b> Correct the PIN profile key serial number.</p>
803 (2051)	Invalid message length in OAEP-decoded information.
804 (2052)	<p>A single-length key, passed to the secure key import callable service in the <i>clear_key</i> parameter, must be padded on the right with binary zeros. The fact that it is a single-length key is identified by the <i>key_form</i> parameter, which identifies the key as being DATA, MACGEN, MACVER, and so on.</p> <p><b>User action:</b> If you are providing a single-length key, pad the parameter on the right with zeros. Alternatively, if you meant to pass a double-length key, correct the <i>key_form</i> parameter to a valid double-length key type.</p>
805 (2053)	No message found in OAEP-decoded information.
806 (2054)	Invalid RSA enciphered key cryptogram; OAEP optional encoding parameters failed validation.

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
807 (2055)	The RSA public key is too small to encrypt the DES key.
808 (2056)	<p>The <i>key_form</i> parameter is neither IM nor OP. Most constants, these included, can be supplied in lower or uppercase. Note that this parameter is 4 bytes long, so the value IM or OP is not valid. They must be padded on the right with blanks.</p> <p><b>User action:</b> Review the value provided and change it to IM or OP, as required.</p> <p><b>REASONCODES:</b> TSS 029 (041)</p>
80C (2060)	<p>The value specified for the <i>key_length</i> parameter of the key generate callable service is not valid.</p> <p><b>User action:</b> Review the value provided and change it as appropriate.</p> <p><b>REASONCODES:</b> TSS 02B (043)</p>
810 (2064)	<p>The <i>key_type</i> and the <i>key_length</i> are not consistent.</p> <p><b>User action:</b> Review the <i>key_type</i> parameter provided and match it with the <i>key_length</i> parameter.</p> <p><b>REASONCODES:</b> TSS 0A0 (160)</p>
811 (2065)	<p>A null key token was not specified for a key identifier parameter.</p> <p><b>User action:</b> Check the service description and determine which key identifier parameter must be a null token.</p>
813 (2067)	<p>TKE: A key part register is in an invalid state. This includes the case where an attempt is made to load a FIRST key part, but a register already contains a key or key part with the same key name.</p> <p><b>User action:</b> Supply a different label name for the key part register or clear the existing key part register with the same label name.</p>
814 (2068)	<p>You supplied a key identifier or token to the key generate, key import, multiple secure key import, key export, or CKDS key record write callable service. This key identifier holds an importer or exporter key, and the NOCV bit is on in the token. Only programs running in supervisor state or in a system key (key 0–7) may provide a key identifier with this bit set on. Your program was not running in supervisor state or a system key.</p> <p><b>User action:</b> Either use a different key identifier, or else run in supervisor state or a system key.</p>
815 (2069)	TKE: The control vector in the key part register does not match the control vector in the key structure.
816 (2070)	<p>TKE: All key part registers are already in use.</p> <p><b>User action:</b> Either free existing key part registers by loading keys from ICSF or clearing selected key part registers from TKE or select another PCIXCC or CCA Crypto Express coprocessor for loading the key part register.</p>
817 (2071)	TKE: The key part hash pattern supplied does not match the hash pattern of the key part currently in the register.

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
818 (2072)	<p>A request was made to the key generate callable service to generate double-length keys of SINGLE effective length, in the IMEX form. This request is valid only if the <i>KEK_key_identifier_1</i> parameter identifies a NOCV importer, and the caller (wrongly) supplies a CV importer. The combination of IMEX for the <i>key_form</i> parameter and a CV importer key-encrypting key can only be used for single-length keys.</p> <p><b>User action:</b> Either use a key identifier that holds (or identifies) a NOCV importer, or specify a single-length key in the <i>key_type</i> parameter.</p>
81B (2075)	<p>TKE: The length of the key part received is different from the length of the accumulated value already in the key part register.</p>
81C (2076)	<p>A request was made to the key import callable service to import a single-length key. However, the right half of the key in the <i>source_key_identifier</i> parameter is not zeros. Therefore, it appears to identify the right half of a double-length key. This combination is not valid. This error does not occur if you are using the word TOKEN in the <i>key_type</i> parameter.</p> <p><b>User action:</b> Check that you specified the value in the <i>key_type</i> parameter correctly, and that you are using the correct or corresponding <i>source_key_identifier</i> parameter.</p>
81D (2077)	<p>TKE: An error occurred storing or retrieving the key part register data.</p> <p><b>User action:</b> Verify that the selected PCIXCC or CCA Crypto Express coprocessor is functioning correctly and retry the operation.</p>
81F (2079)	<p>An encrypted symmetric key token was passed to the service. Either an encrypted key token is not supported for this service (CSNDPKE) or the required hardware is not present (CSNBSYD and CSNBSYE).</p>
824 (2084)	<p>The key token is not valid for the CSNBTK service. If the <i>source_key_identifier</i> is an external token, then the <i>KEK_key_identifier</i> cannot be marked as CDMF.</p> <p><b>User action:</b> Correct the appropriate key identifiers.</p>
828 (2088)	<p>The <i>origin_identifier</i> or <i>destination_identifier</i> you supplied is not a valid ASCII hexadecimal string.</p> <p><b>User action:</b> Check that you specified a valid ASCII string for the <i>origin_identifier</i> or <i>destination_identifier</i> parameter.</p>
829 (2089)	<p>The algorithm does not match the algorithm of the key identifier.</p> <p><b>User action:</b> Make sure the <i>rule_array</i> keywords specified are valid for the type of key specified. Refer to the <i>rule_array</i> parameter described in this publication under the appropriate callable service for the valid values.</p>
82C (2092)	<p>The <i>source_key_identifier</i> or <i>inbound_key_identifier</i> you supplied in an ANSI X9.17 service is not a valid ASCII hexadecimal string.</p> <p><b>User action:</b> Check that you specified a valid ASCII string for the <i>source_key_identifier</i> or <i>inbound_key_identifier</i> parameter.</p> <p><b>REASONCODES:</b> TSS 079 (121)</p>
82D (2093)	<p>Key identifiers contain a version number. The version number in a supplied key identifier (internal or external) is inconsistent with one or more fields in the key identifier, making the key identifier unusable.</p> <p><b>User action:</b> Use a token containing the required version number.</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
82F (2095)	<p>The value in the <i>key_form</i> parameter is incompatible with the value in the <i>key_type</i> parameter.</p> <p><b>User action:</b> Ensure compatibility of the selected parameters.</p>
830 (2096)	<p>The <i>outbound_KEK_count</i> or <i>inbound_KEK_count</i> you supplied is not a valid ASCII hexadecimal string.</p> <p><b>User action:</b> Check that you specified a valid ASCII hexadecimal string for the <i>outbound_KEK_count</i> or <i>inbound_KEK_count</i> parameter.</p> <p><b>REASONCODES:</b> TSS 07A (122)</p>
831 (2097)	<p>The value in the <i>key_identifier_length</i> parameter is incompatible with the value in the <i>key_type</i> parameter.</p> <p><b>User action:</b> Ensure compatibility of the selected parameters.</p>
832 (2098)	<p>Either a key bit length that was not valid was found in an AES key token (length not 128, 192, or 256 bits) or a version X'01' DES token had a token-marks field that was not valid.</p>
833 (2099)	<p>Encrypted key length in an AES key token was not valid when an encrypted key is present in the token.</p>
834 (2100)	<p>You supplied a <i>pad_character</i> that is not valid for a Transaction Security System compatibility parameter for which ICSF supports only one value; or, you supplied a KEY keyword and a non-zero <i>master_key_version_number</i> in the Key Token Build service; or, you supplied a non-zero regeneration data length for a DSS key in the PKA Generate service.</p> <p><b>User action:</b> Check that you specified the valid value for the TSS compatibility parameter.</p> <p><b>REASONCODES:</b> TSS 2CB (715)</p>
838 (2104)	<p>An input character is not in the code table.</p> <p><b>User action:</b> Correct the code table or the source text.</p> <p><b>REASONCODES:</b> TSS 02D (045)</p>
83C (2108)	<p>An unused field must be binary zeros, and an unused key identifier field generally must be zeros.</p> <p><b>User action:</b> Correct the parameter list.</p> <p><b>REASONCODES:</b> TSS 02F (047)</p>
83F (2111)	<p>There is an inconsistency between the wrapping information in the key token and the request to wrap a key.</p>
840 (2112)	<p>The length is incorrect for the key type.</p> <p><b>User action:</b> Check the key length parameter. DATA keys may have a length of 8, 16, or 24. DATA LAT and MAC keys must have a length of 8. All other keys should have a length of 16. Also check that the parameters are in the required sequence.</p>
841 (2113)	<p>A key token contains invalid payload.</p> <p><b>User action:</b> Recreate the key token.</p>
844 (2116)	<p>Parameter contents or a parameter value is not correct.</p> <p><b>User action:</b> Specify a valid value for the parameter.</p> <p><b>REASONCODES:</b> TSS 021 (033)</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
846 (2118)	Invalid value(s) in TR-31 key block header.  <b>User action:</b> Check the TR-31 key block header for correctness. Also check that the PADDING optional block is the last optional block in a set of optional blocks.
847 (2119)	“Mode” value in the TR-31 header is invalid or is not acceptable in the chosen operation.  <b>User action:</b> Check the TR-31 key block header for correctness.
849 (2121)	“Algorithm” value in the TR-31 header is invalid or is not acceptable in the chosen operation.  <b>User action:</b> Check the TR-31 key block header for correctness.
84A (2122)	If importing a TR-31 key block, the exportability byte in the TR-31 header contains a value that is not supported. If exporting a TR-31 key block, the requested exportability is inconsistent with the key block. For example a ‘B’ Key Block Version ID key can only be wrapped by a KEK that is wrapped in CBC mode, the ECB mode KEK violates ANSI X9.24.  <b>User action:</b> Check the TR-31 key block header for correctness.
84B (2123)	The length of the cleartext key in the TR-31 block is invalid, for example the algorithm is “D” for single-DES but the key length is not 64 bits.  <b>User action:</b> Check that the values in the TR-31 header are consistent with the key fields.
84D (2125)	The Key Block Version ID in the TR-31 header contains an invalid value.  <b>User action:</b> Check the TR-31 key block header for correctness.
84E (2126)	The key usage field in the TR-31 header contains a value that is not supported for import of the key into CCA.  <b>User action:</b> Check the TR-31 key block header for correctness.
84F (2127)	The key usage field in the TR-31 header contains a value that is not valid with the other parameters in the header.  <b>User action:</b> Check the TR-31 key block header for correctness
851 (2129)	A parameter to a TR-31 service such as a TR-31 key block, a set of optional blocks, or a single optional block contains invalid characters. It may be that the parameter contains EBCDIC characters when ASCII is expected or vice-versa, or the wrong characters were found in a field which only accepts a limited range of characters. For example some length fields can be populated by characters '0' - '9' and 'A' - 'F', while other length fields can only contain characters '0' - '9'.  <b>User action:</b> Check the TR-31 parameters for correctness
852 (2130)	The CV carried in the TR-31 key block optional blocks is inconsistent with other attributes of the key  <b>User action:</b> Check the TR-31 key block header for correctness.
853 (2131)	The MAC validate step failed for a parameter. This may result from tampering, corruption, or attempting to use a different key to validate the MAC from the one used to generate it.  <b>User action:</b> Check each parameter which includes a MAC for correctness. If the parameter is wrapped by a key-encrypting-key (KEK), ensure that the correct KEK is supplied.
856 (2134)	The requested PIN decimalization table does not exist or no PIN decimalization tables have been stored in the coprocessor.
857 (2135)	The supplied PIN decimalization table is not in the list of active tables stored in the coprocessor.

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
85D (2141)	The key verification pattern for the key-encrypting key is not valid.
85E (2142)	<p>A key-usage field setting in a supplied key token prevents operation. This may be due to:</p> <ul style="list-style-type: none"> <li>• a conflict in key-usage setting between two key tokens</li> <li>• a key-usage bit required for the operation is not enabled</li> <li>• an enabled key-usage bit is disallowed for the key for the operation.</li> </ul> <p><b>User action:</b> Supply key tokens with allowed key-usage settings.</p>
85F (2143)	<p>On a call to Key Translate2 using the REFORMAT Encipherment rule and providing a variable-length AES token, the key management fields for input_key_token contain disallowed values or prohibit the operation.</p> <p><b>User action:</b> Call Key Translate2 using a key token whose key-management fields contain allowed values.</p>
861 (2145)	<p>The service failed because a key would have been wrapped by a weaker key (transport or master key). This is disallowed by the "Prohibit weak wrapping - Transport keys" and "Prohibit weak wrapping - Master keys" access control points.</p> <p><b>User action:</b> If weak key wrapping is to be allowed, disable access control point "Prohibit weak wrapping - Transport keys" and "Prohibit weak wrapping - Master keys" using the TKE workstation.</p>
863 (2147)	<p>The key type that was to be generated by this callable service is not valid.</p> <p><b>User action:</b> Refer to the parameters described in this publication under the appropriate callable service for the correct parameter values.</p>
865 (2149)	<p>The key that was to be generated by this callable service is stronger than the input material.</p> <p><b>User action:</b> Validate the key material is at least as strong as the key to be generated.</p>
86A (2154)	<p>At least one key token passed to this callable service does not have the required key type for the specified function.</p> <p><b>User action:</b> Refer to the parameters described in this publication under the appropriate callable service for the correct parameter values.</p>
86E (2156)	<p>Multiple ECC tokens were passed to this callable service. The curve types of the all the token parameters do not match.</p> <p><b>User action:</b> Check that the curve types of the input ECC tokens are the same.</p>
871 (2161)	<p>The requested or default wrapping method conflicts with one or both input tokens.</p> <p><b>User action:</b> On the call to the CVV Key Combine service, make sure that the desired wrapping method (either specified as a <i>rule_array</i> keyword or the default wrapping method) is consistent with the wrapping method of the input token(s). For example, an input token that can only be wrapped in the enhanced method (ENH-ONLY flag on in the CV) cannot produce an output token wrapped in the original method (ECB mode).</p>
873 (2163)	<p>A weak master key was detected when the final key part was loaded for the DES or RSA master key. A key is weak if any of the three parts are the same as another part. For example, when the first and third key parts are the same, the key is weak (effectively a double-length key).</p> <p><b>User action:</b> Create new key values for the new master key and retry master key entry.</p>
875 (2165)	The RSA key token contains a private section that is not valid with the service.



Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
BB9 (3001)	<p>SET block decompose service was called with an encrypted OAEP block with a block contents identifier that indicates a PIN block is present. No PIN encrypting key was supplied to process the PIN block. The block contents identifier is returned in the <i>block_contents_identifier</i> parameter.</p> <p><b>User action:</b> Supply a PIN encrypting key and resubmit the job.</p>
BBB (3003)	<p>An output parameter is too short to hold the output of the request. The length parameter for the output parameter has been updated with the required length for the request.</p> <p><b>User action:</b> Update the size of the output parameter and length specified in the length field and resubmit the request.</p>
BBC (3004)	<p>A request was made to the Clear PIN generate or Encrypted PIN verify callable service, and the <i>PIN_length</i> parameter has a value outside the valid range. The valid range is from 4 to 16, inclusive.</p> <p><b>User action:</b> Correct the value in the <i>PIN_length</i> parameter to be within the valid range from 4 to 16.</p> <p><b>REASONCODES:</b> TSS 064 (100)</p>
BBE (3006)	<p>The UDX verb in the PCICC, PCIXCC, or CCA Crypto Express coprocessor is not authorized to be executed.</p>
BC0 (3008)	<p>A request was made to the Clear PIN generate callable service, and the <i>PIN_check_length</i> parameter has a value outside the valid range. The valid range is from 4 to 16, inclusive.</p> <p><b>User action:</b> Correct the value in the <i>PIN_check_length</i> parameter to be within the valid range from 4 to 16.</p> <p><b>REASONCODES:</b> TSS 065 (101)</p>
BC1 (3009)	<p>For PKCS #11 attribute processing, an attribute has been specified in the template that is not consistent with another attribute of the object being created or updated.</p> <p><b>User action:</b> Correct the template for the object.</p>
BC3 (3011)	<p>The CRT value (p, q, Dp, Dq or U) is longer than the length allowed by the parameter block for clear key processing on an accelerator. A modulus whose length is less than or equal to 1024 bits is 64 bytes in length. A modulus whose length is greater than 1024 bits but less than or equal to 2048 bits is 128 bytes in length.</p> <p><b>User action:</b> Reconfigure the accelerator as a CCA Crypto Express coprocessor to make use of the key (if the CRT value is not in error and there is no CCA Crypto Express coprocessor installed).</p> <p><b>REASONCODES:</b> TSS 065 (101)</p>
BC4 (3012)	<p>A request was made to the Clear PIN generate callable service to generate a VISA-PVV PIN, and the <i>trans_sec_parm</i> field has a value outside the valid range. The field being checked in the <i>trans_sec_parm</i> is the key index, in the 12th byte. This <i>trans_sec_parm</i> field is part of the <i>data_array</i> parameter.</p> <p><b>User action:</b> Correct the value in the key index, held within the <i>trans_sec_parm</i> field in the <i>data_array</i> parameter, to hold a number from the valid range.</p> <p><b>REASONCODES:</b> TSS 069 (105)</p>



Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
BC5 (3013)	<p>The AES clear key value LRC in the token failed validation.</p> <p><b>User action:</b> Correct the AES clear key value.</p> <p><b>REASONCODES:</b> TSS 06A (106)</p>
BC8 (3016)	<p>A request was made to the Encrypted PIN Translate or the Encrypted PIN verify callable service, and the PIN block value or PADDIGIT value in the <i>input_PIN_profile</i> or <i>output_PIN_profile</i> parameter has a value that is not valid.</p> <p><b>User action:</b> Correct the PIN block value.</p> <p><b>REASONCODES:</b> TSS 06A (106)</p>
BCB (3019)	<p>The call to insert or delete a z/OS PKCS #11 token object failed because the token was not found in the TKDS data space or a request to delete a PKCS #11 session object failed because the token was not found in the session data space.</p>
BCC (3020)	<p>For a PKCS #11 callable service, the PKCS #11 object specified is the incorrect class for the request.</p> <p><b>User action:</b> Specify the correct class of object for the service.</p>
BCD (3021)	<p>The call to add a z/OS PKCS #11 token failed because the token already exists in the TKDS data space or a request to add a z/OS PKCS #11 token object failed because an object with the same handle already exists.</p>
BCE (3022)	<p>The call to add or update a z/OS PKCS #11 tokens object failed because the supplied attributes are too large to be stored in the TKDS.</p>
BD0 (3024)	<p>A request was made to the Encrypted PIN Translate callable service and the format control value in the <i>input_PIN_profile</i> or <i>output_PIN_profile</i> parameter has a value that is not valid. The valid values are NONE or PBVC.</p> <p><b>User action:</b> Correct the format control value to either NONE or PBVC.</p> <p><b>REASONCODES:</b> TSS 06B (107)</p>
BD1 (3025)	<p>The call to create a list of z/OS PKCS #11 tokens, a list of objects of a z/OS PKCS #11 token, the information for a z/OS PKCS #11 token or the attributes of a PKCS #11 object failed because the length of the output field was insufficient to hold the data. The length field has been updated with the length of a single list or entry, token information or object attributes.</p>
BD2 (3026)	<p>The z/OS PKCS #11 token or object handle syntax is invalid.</p>
BD3 (3027)	<p>The call to read or update a z/OS PKCS #11 token or token object failed because the token or object was not found in the TKDS data space, or if the call to read or update a PKCS #11 session object failed because the object was not found.</p>
BD4 (3028)	<p>A request was made to the Clear PIN generate callable service. The clear_PIN supplied as part of the <i>data_array</i> parameter for an GBP-PINO request begins with a zero (0). This value is not valid.</p> <p><b>User action:</b> Correct the clear_PIN value.</p> <p><b>REASONCODES:</b> TSS 074 (116)</p>
BD5 (3029)	<p>For PKCS #11 attribute processing, an invalid attribute was specified in the template. The attribute is neither a PKCS #11 or vendor-specified attribute supported by this implementation of PKCS #11.</p> <p><b>User action:</b> Correct the template by removing the invalid attribute or changing the attribute to a valid attribute.</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
BD6 (3030)	An invalid value was specified for a particular PKCS #11 attribute in a template when creating or updating an object.
BD7 (3031)	The certificate specified in creating a PKCS #11 certificate object was not properly encoded.
BD9 (3033)	The attribute template for creating or updating a PKCS #11 object was incomplete. Required attributes for the object class were not specified in the template.
BDA (3034)	The call to modify PKCS #11 object attributes failed because the CKA_MODIFIABLE attribute was set to false when the object was recreated.
BDB (3035)	<p>For PKCS #11 attribute processing, an attribute was specified in the template which can not be set or updated by the application. See <i>z/OS Cryptographic Services ICSF Writing PKCS #11 Applications</i> for a definition of attributes that can be set or updated by the application.</p> <p><b>User action:</b> Remove the offending attribute from the template.</p>
BDC (3036)	<p>A request was made to the Encrypted PIN Translate callable service. The <i>sequence_number</i> parameter was required, but was not the integer value 99999.</p> <p><b>User action:</b> Specify the integer value 99999.</p> <p><b>REASONCODES:</b> TSS 06F (111)</p>
BDE (3038)	<p>For a PKCS #11 callable service, the attributes of the PKCS #11 object specified do not permit the requested function.</p> <p><b>User action:</b> Specify an object that permits the requested function.</p>
BDF (3039)	<p>For a PKCS #11 callable service, where a PKCS #11 key object is required, the specified object is not of the correct key type for the requested function.</p> <p><b>User action:</b> Specify an object that is the correct class of key.</p>
BE0 (3040)	<p>The PAN, expiration date, service code, decimalization table data, validation data, or pad data is not numeric (X'F0' through X'F9'). The parameter must be character representations of numerics or hexadecimal data.</p> <p><b>User action:</b> Review the numeric parameters or fields required in the service that you called and change to the format and values required.</p> <p><b>REASONCODES:</b> TSS 028 (040), TSS 02A (042), TSS 066 (102), TSS 067 (103), TSS 068 (104), TSS 069 (105), TSS 06E (110)</p>
BE1 (3041)	<p>PKCS #11 wrap key callable service failed because the wrapping key object is not of the correct class to wrap the key specified to be wrapped.</p> <p><b>User action:</b> Specify a wrapping key object of the correct class to wrap the key object.</p>
BE3 (3043)	<p>PKCS #11 wrap key callable service failed because the key object to be wrapped does not exist or the key class does not match the wrapping mechanism.</p> <p><b>User action:</b> Specify an existing key object that is correct for the wrapping mechanism.</p>
BE4 (3044)	<p>A PKCS #11 session data space is full. The request to create or update an object failed and the object was not created or updated.</p> <p><b>User action:</b> Delete unused session objects and cryptographic state objects from incomplete chained operations to create space for new or updated objects.</p>
BE5 (3045)	<p>PKCS #11 wrap key callable service failed because the key object to be wrapped has CKA_EXTRACTABLE set to false.</p> <p><b>User action:</b> Specify another key object that can be extracted.</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
BE7 (3047)	<p>A clear key was provided when a secure key was required.</p> <p><b>User action:</b> Correct the appropriate key identifier.</p>
BEA (3050)	<p>A caller is attempting to overwrite one token type with another (for example, AES over DES).</p>
BEC (3052)	<p>A clear key token was supplied to a service where a secure token is required.</p>
BED (3053)	<p>A service was called with no parameter list, but a parameter list was expected.</p> <p><b>User action:</b> Call the service with a parameter list.</p>
BEE (3054)	<p>A request was made to a callable service with a key token wrapped with the enhanced X9.24 CBC method. Tokens wrapped with the enhanced method are not supported by this release of ICSF.</p> <p><b>User action:</b> Contact your ICSF administrator to resolve which key token is to be used.</p>
BF5 (3061)	<p>The provided asymmetric key identifier can not be used for the requested function. PKA Key Management Extensions have been enabled by a CSF.PKAEXTNS.ENABLE profile in the XFACILIT class. A CSFKEYS profile covering the key includes an ICSF segment, and the ASYMUSAGE field of that segment restricts the key from being used for the specified function.</p> <p>An SMF type 82 subtype 27 record is logged in the SMF database.</p>
BF6 (3062)	<p>The provided symmetric key identifier can not be exported using the provided asymmetric key identifier. PKA Key Management Extensions have been enabled by a CSF.PKAEXTNS.ENABLE profile in the XFACILIT class. A CSFKEYS or XCSFKEY profile covering the symmetric key includes an ICSF segment and the SYMEXPORTABLE field of that segment places restrictions on how the key can be exported. The SYMEXPORTABLE field either specifies BYNONE, or else specifies BYLIST but the provided asymmetric key identifier is not one of those permitted to export the symmetric key (as identified by the SYMEXPORTCERTS or SYMEXPORTKEYS fields).</p> <p>An SMF type 82 subtype 27 record is logged to the SMF database.</p>
BF7 (3063)	<p>ICSF key store policy checking is active. The request failed the ICSF token policy check because the caller is not authorized to the label for the token in the key data set (CKDS or PKDS). The request is not allowed to continue because the token check policy is in FAIL mode.</p> <p>SMF type 82 subtype 25 records are logged in the SMF dataset. An SMF type 80 with event code qualifier of ACCESS is logged.</p> <p>The policy is defined by the CSF.CKDS.TOKEN.CHECK.LABEL.FAIL resource or the CSF.PKDS.TOKEN.CHECK.LABEL.FAIL resource in the XFACILIT class.</p>
BF8 (3064)	<p>ICSF key store policy checking is active. The specified token does not exist in the key data set (CKDS or PKDS as appropriate). The CSF-CKDS-DEFAULT or CSF-PKDS-DEFAULT resource in the CSFKEYS class is either not defined or the caller is not authorized to the CSF-CKDS-DEFAULT or CSF-PKDS-DEFAULT resource. The resource is not in WARNING mode, so the request is not allowed to continue.</p> <p>An SMF type 80 record with event qualifier ACCESS is logged indicating the request failed.</p> <p>The policy is defined by the CSF.CKDS.TOKEN.CHECK.DEFAULT.LABEL or the CSF.PKDS.TOKEN.CHECK.DEFAULT.LABEL resource in the XFACILIT class.</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
BF9 (3065)	<p>ICSF token policy checking is active. The caller is requesting to add a token to the key data set (CKDS or PKDS as appropriate) that already exists within the key data set. The request fails.</p> <p>The policy is defined by the CSF.CKDS.TOKEN.NODUPLICATES resource or the CSF.PKDS.TOKEN.NODUPLICATES resource in the XFACILIT class.</p>
BFB (3067)	<p>The provided symmetric key label refers to an encrypted CCA key token, and the CSFKEYS profile covering it does not allow its use in high performance encrypted key operations.</p> <p><b>User action:</b> Contact your ICSF or RACF administrator if you need to use this key in calls to Symmetric Key Encipher (CSNBSYE) or Symmetric Key Decipher (CSNBSYD). Otherwise, use Encipher (CSNBENC) or Decipher (CSNBDEC) instead.</p>
BFC (3068)	<p>A cryptographic operation using a specific PKCS #11 key object is being requested. The key object has exceeded its useful life for the operation requested. The request is not processed.</p> <p><b>User action:</b> Use a different key.</p>
BFD (3069)	<p>A cryptographic operation that requires FIPS 140-2 compliance is being requested. Either ICSF has not been configured to run in FIPS mode or the system environment does not support it. The request is not processed.</p> <p><b>User action:</b> Contact your ICSF administrator to request that ICSF be configured for either FIPS standard mode or FIPS compatibility mode.</p>
BFE (3070)	<p>A cryptographic operation that requires FIPS 140-2 compliance is being requested. The desired algorithm, mode, or key size is not approved for FIPS 140-2. The request is not processed.</p> <p><b>User action:</b> Repeat the request using an algorithm, mode, and/or key size approved for FIPS 140-2. Refer to <i>z/OS Cryptographic Services ICSF Writing PKCS #11 Applications</i> for this list of approved algorithms, modes, and key sizes.</p>
BFF (3071)	<p>An application using a z/OS PKCS #11 token that is marked 'Write Protected' is attempting to do one of the following:</p> <ul style="list-style-type: none"> <li>• Store a persistent object in the token.</li> <li>• Delete the token.</li> <li>• Reinitialize the token.</li> </ul> <p>ICSF always marks the session object only omnipresent token as 'Write Protected.' ICSF will also mark an ordinary token 'Write Protected' if it contains objects not supported by this release of ICSF.</p> <p><b>User action:</b> Use a z/OS PKCS #11 token that is not marked 'Read Only' or, if this is an ordinary token (not the omnipresent token), attempt the delete or reinitialization from a different member of the sysplex.</p>
C04 (3076)	<p>A symmetric key token was supplied in a key identifier parameter which is wrapped using the enhanced X9.24 key wrapping method. The token can not be rewrapped to the original method because the wrapping flag in the control vector prohibits this wrapping.</p>
C07 (3079)	<p>A request was made to use a key token wrapped with the X9.24 enhanced wrapping method introduced in HCR7780. Key tokens wrapped with the enhanced method can not be used on this release. Also, key tokens wrapped with the enhanced method can not be updated or deleted from the CKDS on this release.</p> <p><b>User Action:</b> Run your application on a release that support the enhanced wrapping method.</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
C08 (3080)	<p>Use of an ECC token has been attempted. The usage of this type of token is not supported on the release of ICSF currently running.</p> <p><b>User Action:</b> Check the ICSF release for support of this token type.</p>
C0B (3083)	<p>The specified key token buffer length is of insufficient size for the buffer to contain the output key token.</p> <p><b>User action:</b> Specify a key token buffer that is sufficiently large enough to receive the output key token.</p>
C0C (3084)	<p>The key token associated with the specified key label is not a DES or AES key token, but this callable service is only compatible with DES and AES key tokens.</p> <p><b>User action:</b> Either modify the program logic to utilize only key labels for DES and/or AES key tokens, or use an ICSF callable service that supports all of the symmetric key token types.</p>
C0D (3085)	<p>Rule array keyword specifies a function not supported by this hardware. For example, ECC specified in rule array for PKA Key Token Change callable service but request is being executed on a system that does not support ECC keys.</p> <p><b>User Action:</b> Specify a different, supported, rule array keyword, or execute the service on a system that supports the function.</p>
C0E (3086)	<p>Specified token is not supported by this hardware. For example, an ECC token is being used but request is being executed on a system that does not support ECC keys.</p> <p><b>User Action:</b> Specify a different, supported, token, or execute the request on a system that supports the function.</p>
C0F (3087)	<p>A coordinated KDS refresh was attempted to an empty KDS. The new KDS of a coordinated KDS refresh must be initialized and must contain the same MKVP values as the active KDS.</p> <p><b>User action:</b> Perform a coordinated KDS refresh using a new KDS that is initialized and that contains the same MKVP values as the active KDS.</p>
C10 (3088)	<p>A coordinated KDS change master key was attempted and either the new KDS or backup KDS contained a different LRECL attribute from the active KDS. The new KDS and optionally the backup KDS must contain the same LRECL attribute as the active KDS during a coordinate KDS change master key.</p> <p><b>User action:</b> Perform a coordinated KDS change master key using a new KDS and optionally a backup KDS with the same LRECL attribute as the active KDS.</p>
C11 (3089)	<p>The new KDS specified for a coordinated KDS change master key was not empty when the operation began. The new KDS must be empty before performing a coordinated KDS change master key.</p> <p><b>User action:</b> Perform the coordinated KDS change master key with a new KDS that is empty.</p>
C12 (3090)	<p>The backup KDS specified for a coordinated KDS change master key was not empty when the operation began. When using the optional backup function, the backup KDS must be empty before performing a coordinated KDS change master key.</p> <p><b>User action:</b> Perform the coordinated KDS change master key with a backup KDS that is empty.</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
C13 (3091)	<p>The new KDS specified for a coordinated KDS refresh contains different MKVPs than the active KDS. In order to perform a coordinated KDS refresh, the new KDS specified must contain the same MKVPs as the active KDS.</p> <p><b>User action:</b> Perform the coordinated KDS refresh with a new KDS that contains the same MKVPs as the active KDS.</p>
C1F (3103)	<p>The new KDS specified for either a coordinated KDS refresh or coordinated KDS change master key is not a valid data set name.</p> <p><b>User action:</b> Specify a valid data set name for the new KDS when performing either a coordinated KDS refresh or coordinated KDS change master key.</p>
C20 (3104)	<p>The backup KDS specified for a coordinated KDS change master key is not a valid data set name.</p> <p><b>User action:</b> Specify a valid data set name for the backup KDS when performing a coordinated KDS change master key.</p>
C21 (3105)	<p>A coordinated KDS refresh or coordinated KDS change master key was attempted while at least one ICSF instance in the sysplex was below the HCR7790 FMID level. The coordinated KDS refresh and coordinated KDS change master key functions are only available when all ICSF instances in the sysplex, regardless of active KDS, are running at the HCR7790 FMID level or higher.</p> <p><b>User action:</b> Remove or upgrade ICSF instances in the sysplex that are running below the HCR7790 FMID level and retry the function.</p>
C22 (3106)	<p>Either a coordinated KDS refresh or coordinated KDS change master key was attempted while another coordinated KDS refresh or coordinated KDS change master key was still in progress. The coordinated KDS function was initiated by this ICSF instance. Only one coordinated KDS function may execute at a time in the sysplex.</p> <p><b>User action:</b> Wait for the previous coordinated KDS function to complete and retry the function.</p>
C23 (3107)	<p>A coordinated KDS change master key was attempted using a new KDS with the same name as the active KDS. The new KDS name must be different from the active KDS when performing a coordinated KDS change master key.</p> <p><b>User action:</b> Specify a new KDS with a different name from the active KDS and retry the function. Coordinated KDS change master key requires the new KDS to be allocated and match the same VSAM attributes as the active KDS.</p>
C24 (3108)	<p>A coordinated KDS change master key was attempted using a backup KDS with the same name as the active KDS. When using the backup function, the backup KDS name must be different from the active KDS when performing a coordinated KDS change master key.</p> <p><b>User action:</b> Specify a backup KDS with a different name from the active KDS and retry the function. Coordinated KDS change master key requires the backup KDS to be allocated and match the same VSAM attributes as the active KDS.</p>
C25 (3109)	<p>A coordinated KDS change master key was attempted using a new KDS with the same name as the backup KDS. If a backup KDS is specified, its name must be different from the new KDS.</p> <p><b>User action:</b> Specify a backup KDS with a different name from the new KDS and retry the function. The backup KDS is optional. Coordinated KDS change master key requires the new KDS, and optionally the backup KDS, to be allocated and match the same VSAM attributes as the active KDS.</p>



Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
C26 (3110)	<p>A coordinated KDS refresh or coordinated KDS change master key was attempted using an archive KDS name that is not valid.</p> <p><b>User action:</b> Specify a valid data set name for the archive KDS and retry the function. The archive data set name is optional. The optional archive KDS name must not exist on the system prior to performing a coordinated KDS refresh or a coordinated KDS change master key.</p>
C27 (3111)	<p>A coordinated KDS change master key was attempted using an archive KDS with the same name as the backup KDS. When using the archive and backup functions, the archive KDS name must be different from the backup KDS.</p> <p><b>User action:</b> Specify an archive KDS with a different name from the backup KDS and retry the function. The archive KDS name and the backup KDS are optional. The archive KDS name must not exist on the system prior to performing a coordinated KDS refresh or a coordinated KDS change master key. The backup KDS must be allocated and match the same VSAM attributes as the active KDS.</p>
C28 (3112)	<p>A coordinated KDS refresh or a coordinated KDS change master key was attempted using an archive KDS with the same name as the active KDS. When using the archive function, the archive KDS name must be different from the active KDS.</p> <p><b>User action:</b> Specify an archive KDS with a different name from the active KDS and retry the function. The archive KDS name must not exist on the system prior to performing a coordinated KDS refresh or a coordinated KDS change master key.</p>
C29 (3113)	<p>A coordinated KDS refresh or a coordinated KDS change master key was attempted using an archive KDS with the same name as the new KDS. When using the archive function, the archive KDS name must be different from the new KDS.</p> <p><b>User action:</b> Specify an archive KDS with a different name than the new KDS and retry the function. The archive KDS name must not exist on the system prior to performing a coordinated KDS refresh or a coordinated KDS change master key.</p>
C2A (3114)	<p>Either a coordinated KDS refresh or coordinated KDS change master key was attempted while another coordinated KDS refresh or coordinated KDS change master key was still in progress. The coordinated KDS function was initiated by another ICSF instance in the sysplex. Only one coordinated KDS function may execute at a time in the sysplex.</p> <p><b>User action:</b> Wait for the previous coordinated KDS function to complete and retry the function.</p>
C30 (3120)	<p>A coordinated KDS change master key was attempted on an active KDS that was not initialized. The active KDS must be initialized before performing a coordinated KDS change master key.</p> <p><b>User action:</b> Initialize the active KDS and retry the function</p>
C31 (3121)	<p>The archive option was specified for a coordinated KDS refresh of the active KDS. The archive option is only valid for coordinated KDS refreshes to a new KDS or coordinated KDS change master key.</p> <p><b>User action:</b> Do not specify an archive data set when performing a coordinated KDS refresh of the active KDS.</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
C3C (3132)	<p>The archive data set name specified for coordinated KDS refresh or coordinated KDS change master key is too long. The archive data set name must allow enough space for renaming the KDS VSAM data and index portions within 44 characters.</p> <p><b>User action:</b> Specify a shorter name for the archive data set name to allow enough space for renaming the KDS VSAM data and index portions within 44 characters. The archive data set name is optional. When specified, the archive data set name must not exist on the system prior to performing the coordinated KDS function.</p>
C3D (3133)	<p>During a coordinated KDS refresh or coordinated KDS change master key with the archive option specified, the active KDS could not be renamed to the archive data set name. This failure occurred because the active KDS VSAM data and index suffix names were not valid for performing the rename.</p> <p><b>User action:</b> Consider alternate names for the active KDS VSAM data and index suffixes. The archive data set name is optional. When specified the archive data set name must not exist on the system prior to performing the coordinated KDS function.</p>
C3E (3134)	<p>A coordinated KDS change master key attempted to use a new KDS that is currently another sysplex members active KDS. Performing a coordinated KDS change master key to another sysplex members active KDS is not allowed as it would alter all sysplex members configured in that sysplex KDS cluster (same active KDS).</p> <p><b>User action:</b> Specify a new KDS that is not currently the active KDS of another sysplex member and retry the function.</p>
C81 (3201)	<p>Operation requested requires a clear key, but a secure key was supplied.</p> <p><b>User action:</b> Use a different key, one that is clear.</p>
F9F (3999)	<p>On a call to CKDS Key Record Delete or CKDS Key Record Write2, the label refers to a Variable-length Symmetric key token with an unrecognized algorithm or key type in the associated data section. Only key tokens with a recognized algorithm or key type can be managed on this release of ICSF.</p> <p><b>User action:</b> Call CKDS Key Record Delete or CKDS Key Record Write2 on a release of ICSF which recognizes the algorithm and key type of this token.</p>
FA0 (4000)	<p>The encipher and decipher callable services sometime require text (plaintext or ciphertext) to have a length that is an exact multiple of 8 bytes. Padding schemes always create ciphertext with a length that is an exact multiple of 8. If you want to decipher ciphertext that was produced by a padding scheme, and the text length is not an exact multiple of 8, then an error has occurred. The CBC mode of enciphering requires a text length that is an exact multiple of 8.</p> <p>The ciphertext translate callable service cannot process ciphertext whose length is not an exact multiple of 8.</p> <p><b>User action:</b> Review the requirements of the service you are using. Either adjust the text you are processing or use another process rule.</p> <p><b>REASONCODES:</b> TSS 033 (051)</p>
1388 (5000)	<p>Target cryptographic module is not available in the configuration.</p> <p><b>User action:</b> Correct the target cryptographic module parameter and resubmit.</p>
138C (5004)	<p>Format of the cryptographic request message is not valid.</p> <p><b>User action:</b> Correct the request and resubmit it.</p>



Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
1390 (5008)	<p>Length of the cryptographic request message is not valid.</p> <p><b>User action:</b> Message length of request must be nonzero, a multiple of eight, and less than the system maximum. Correct the request and resubmit it.</p>
1782 (6018)	<p>One or more of the parameters passed to this callable service are in error.</p> <p><b>User action:</b> Refer to the parameter descriptions in this publication under the appropriate callable service to ensure the parameter values specified by your application are valid.</p>
2710 (10000)	<p>A key identifier was passed to a service or token. It is checked in detail to ensure that it is a valid token, and that the fields within it are valid values. There is a token validation value (TVV) in the token, which is a non-cryptographic value. This value was again computed from the rest of the token, and compared to the stored TVV. If these two values are not the same, this reason code is returned.</p> <p><b>User action:</b> The contents of the token have been altered because it was created by ICSF or TSS. Review your program to see how this could have been caused.</p> <p><b>REASONCODES:</b> TSS 0C (12) and 1D (29)</p>
2714 (10004)	<p>A key identifier was passed to a service. The master key verification pattern in the token shows that the key was created with a master key that is neither the current master key nor the old master key. Therefore, it cannot be reenciphered to the current master key.</p> <p><b>User action:</b> Re-import the key from its importable form (if you have it in this form), or repeat the process you used to create the operational key form. If you cannot do one of these, you cannot repeat any previous cryptographic process that you performed with this token.</p> <p><b>REASONCODES:</b> TSS 030 (048)</p>
271C (10012)	<p>A key label was supplied for a key identifier parameter. This label is the label of a key in the in-storage CKDS or the PKDS. Either the key could not be found, or a key record with that label and the specific type required by the ICSF callable service could not be found. For a retained key label, this error code is also returned if the key is not found in the PCICC, PCIXCC, or CCA Crypto Express coprocessor specified in the PKDS record.</p> <p><b>User action:</b> Check with your administrator if you believe that this key should be in the in-storage CKDS or the PKDS. The administrator may be able to bring it into storage. If this key cannot be in storage, use a different label.</p> <p><b>REASONCODES:</b> TSS 01E (030)</p>
2720 (10016)	<p>You specified a value for a <i>key_type</i> parameter that is not an ICSF-defined name.</p> <p><b>User action:</b> Review the ICSF key types and use the appropriate one.</p> <p><b>REASONCODES:</b> TSS 03D (061)</p>
2724 (10020)	<p>You specified the word TOKEN for a <i>key_type</i> parameter, but the corresponding key identifier, which implies the key type to use, has a value that is not valid in the control vector field. Therefore, a valid key type cannot be determined.</p> <p><b>User action:</b> Review the value that you stored in the corresponding key identifier. Check that the value for <i>key_type</i> is obtained from the appropriate <i>key_identifier</i> parameter.</p> <p><b>REASONCODES:</b> TSS 027 (039)</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
272C (10028)	<p>Either the <i>left</i> half of the control vector in a key identifier (internal or external) equates to a key type that is not valid for the service you are using, or the value is not that of any ICSF control vector. For example, an exporter key-encrypting key is not valid in the key import callable service.</p> <p><b>User action:</b> Determine which key identifier is in error and use the key identifier that is required by the service.</p> <p><b>REASONCODES:</b> TSS 027 (039)</p>
2730 (10032)	<p>Either the <i>right</i> half of the control vector in a key identifier (internal or external) equates to a key type that is not valid for the service you are using, or the value is not that of any ICSF control vector. For example, an exporter key-encrypting key is not valid in the key import callable service.</p> <p><b>User action:</b> Determine which key identifier is in error and use the key identifier that is required by the service.</p> <p><b>REASONCODES:</b> TSS 027 (039)</p>
2734 (10036)	<p>Either the complete control vector (CV) in a key identifier (internal or external) equates to a key type that is not valid for the service you are using, or the value is not that of any ICSF control vector.</p> <p>The difference between this and reason codes 10028 and 10032 is that each half of the control vector is valid, but <i>as a combination</i>, the whole is not valid. For example, the left half of the control vector may be the importer key-encrypting key and the right half may be the input PIN-encrypting (IPINENC) key.</p> <p><b>User action:</b> Determine which key identifier is in error and use the key identifier that is required by the service.</p> <p><b>REASONCODES:</b> TSS 027 (039)</p>
2738 (10040)	<p>Key identifiers contain a version number. The version number in a supplied key identifier (internal or external) is inconsistent with one or more fields in the key identifier, making the key identifier unusable.</p> <p><b>User action:</b> Use a token containing the required version number.</p> <p><b>REASONCODES:</b> TSS 031 (049)</p>
273C (10044)	<p>A cross-check of the control vector the key type implies has shown that it does not correspond with the control vector present in the supplied internal key identifier.</p> <p><b>User action:</b> Change either the key type or key identifier.</p> <p><b>REASONCODES:</b> TSS 0B7 (183)</p>
2740 (10048)	<p>The <i>key_type</i> parameter does not contain one of the valid types for the service or the keyword TOKEN.</p> <p><b>User action:</b> Check the supplied parameter with the ICSF key types. If you supplied the keyword TOKEN, check that you have padded it on the right with blanks.</p> <p><b>REASONCODES:</b> TSS 03D (061)</p>
2744 (10052)	<p>A null key identifier was supplied and the <i>key_type</i> parameter contained the word TOKEN. This combination of parameters is not valid.</p> <p><b>User action:</b> Use either a null key identifier or the word TOKEN, not both.</p> <p><b>REASONCODES:</b> TSS 027 (039)</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
2748 (10056)	<p>You called the key import callable service. The importer key-encrypting key is a NOCV importer and you specified TOKEN for the <i>key_type</i> parameter. This combination is not valid.</p> <p><b>User action:</b> Specify a value in the <i>key_type</i> parameter for the operational key form.</p>
274C (10060)	<p>You called the key export callable service. A label was supplied in the <i>key_identifier</i> parameter for the key to be exported and the <i>key_type</i> was TOKEN. This combination is not valid because the service needs a key type in order to retrieve a key from the CKDS.</p> <p><b>User action:</b> Specify the type of key to be exported in the <i>key_type</i> parameter.</p> <p><b>REASONCODES:</b> TSS 03D (061)</p>
2754 (10068)	<p>A flag in a key identifier indicates the master key verification pattern (MKVP) is not present in an internal key token. This setting is not valid.</p> <p><b>User action:</b> Use a token containing the required flag values.</p> <p><b>REASONCODES:</b> TSS 02F (047)</p>
2758 (10072)	<p>A flag in a key identifier indicates the encrypted key is not present in an external token. This setting is not valid.</p> <p><b>User action:</b> Use a token containing the required flag values.</p> <p><b>REASONCODES:</b> TSS 02F (047)</p>
275C (10076)	<p>A flag in a key identifier indicates the control vector is not present. This setting is not valid.</p> <p><b>User action:</b> Use a token containing the required flag values.</p> <p><b>REASONCODES:</b> TSS 02F (047)</p>
2760 (10080)	<p>An ICSF private flag in a key identifier has been set to a value that is not valid.</p> <p><b>User action:</b> Use a token containing the required flag values. Do not modify ICSF or the reserved flags for your own use.</p>
2768 (10088)	<p>If you supplied a label in the <i>key_identifier</i> parameter, a record with the supplied label was found in the CKDS, but the key type (CV) is not valid for the service. If you supplied an internal key token for the <i>key_identifier</i> parameter, it contained a key type that is not valid.</p> <p><b>User action:</b> Check with your ICSF administrator if you believe that this key should be in the in-storage CKDS. The administrator may be able to bring it into storage. If this key cannot be in storage, use a different label.</p> <p><b>REASONCODES:</b> TSS 027 (039)</p>
276C (10092)	<p>You supplied a source key that does not have odd parity and specified ENFORCE as the parity rule on the <i>rule_array</i> parameter for either the ANSI X9.17 key export, ANSI X9.17 key import, or ANSI X9.17 key translate callable service.</p> <p><b>User action:</b> Either supply an ODD parity key or change the <i>rule_array</i> parameter to specify a parity rule of IGNORE.</p>
2770 (10096)	<p>The transport key you specified is a single-length key, which cannot be used to encrypt a double-length AKEK or (*KK).</p> <p><b>User action:</b> Use a double-length AKEK for the transport key.</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
2774 (10100)	<p>You specified a transport key that cannot be notarized and specified the keyword NOTARIZE in the <i>rule_array</i> parameter. The transport key may have already been partially notarized.</p> <p><b>User action:</b> Use a transport key that allows notarization or change the <i>rule_array</i> parameter keyword to CPLT-NOT.</p>
2778 (10104)	<p>The AKEK you specified is either partially notarized or is a partial AKEK, which is not valid for this service.</p> <p><b>User action:</b> Use a correct AKEK that is not partially notarized. A partially notarized key can be used as a transport key if you specify CPLT-NOT in the <i>rule_array</i> parameter.</p>
277C (10108)	<p>You did not supply a partial AKEK for the <i>key_identifier</i> parameter of the key part import service.</p> <p><b>User action:</b> Correct the <i>key_id</i> parameter.</p>
2780 (10112)	<p>The transport key you specified has not been partially notarized and you have specified CPLT-NOT for the <i>rule_array</i> parameter.</p> <p><b>User action:</b> Use a transport key that has been partially notarized or change the <i>rule_array</i> parameter.</p>
2784 (10116)	<p>You attempted to export an AKEK with a CCA key export service, which is not supported.</p> <p><b>User action:</b> Use the ANSI X9.17 Key Export callable service.</p>
2788 (10120)	<p>The internal key token you supplied, or the key token that was retrieved by the label you supplied, contains a flag setting or data encryption algorithm bit that is not valid for this service.</p> <p><b>User action:</b> Ensure that you supply a key token, or label, for a non-ANSI key type.</p>
278C (10124)	<p>The key identifier you supplied cannot be exported because there is a prohibit-export restriction on the key.</p> <p><b>User action:</b> Use the correct key for the service.</p> <p><b>REASONCODES:</b> TSS 027 (039)</p>
2790 (10128)	<p>The keyword you supplied in the <i>rule_array</i> parameter is not consistent or not valid with another parameter you specified. For example, the keyword SINGLE is not valid with the key type of EXPORTER in the key token build callable service.</p> <p><b>User action:</b> Correct either the <i>rule_array</i> parameter or the other parameter.</p> <p><b>REASONCODES:</b> TSS 09C (156)</p>
2791 (10129)	<p>S390 KEKs with NOCV (flagged as such by the MASK_NOCV bit in the flags field of the token), are not permitted in the RKX service.</p>
2AF8 (11000)	<p>The value specified for length parameter for a key token, key, or text field is not valid.</p> <p><b>User action:</b> Correct the appropriate length field parameter.</p> <p><b>REASONCODES:</b> TSS 048 (072)</p>
2AFC (11004)	<p>The hash value (of the secret quantities) in the private key section of the internal token failed validation. The values in the token are corrupted. You cannot use this key.</p> <p><b>User action:</b> Recreate the token using the appropriate combination of the PKA key token build, PKA key generate, and PKA key import callable services.</p> <p><b>REASONCODES:</b> TSS 02F (047)</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
2B00 (11008)	<p>The public or private key values are not valid. (For example, the modulus or an exponent is zero.) You cannot use the key.</p> <p><b>User action:</b> You may need to recreate the token using the PKA key token build or PKA key import callable service or regenerate the key values on another platform.</p> <p><b>REASONCODES:</b> TSS 302 (770)</p>
2B04 (11012)	<p>The internal or external private key token contains flags that are not valid.</p> <p><b>User action:</b> You may need to recreate the token using the PKA key token build or PKA key import callable service.</p> <p><b>REASONCODES:</b> TSS 02F (047)</p>
2B08 (11016)	<p>The calculated hash of the public information in the PKA token does not match the hash in the private section of the token. The values in the token are corrupted.</p> <p><b>User action:</b> Verify the public key section and the key name section of the token. If the token is still rejected, then you need to recreate the token using the appropriate combination of the PKA key token build, PKA key generate, and PKA key import callable services.</p> <p><b>REASONCODES:</b> TSS 02F (047)</p>
2B0C (11020)	<p>The hash pattern of the PKA master key (SMK or KMMK) in the supplied internal PKA private key token does not match the current system's PKA master key. This indicates the system PKA master key has changed since the token was created. You cannot use the token.</p> <p><b>User action:</b> Recreate the token using the appropriate combination of the PKA key token build, PKA key generate, and PKA key import callable services.</p> <p><b>REASONCODES:</b> TSS 030 (048)</p>
2B10 (11024)	<p>The PKA tokens have incomplete values, for example, a PKA public key token without modulus.</p> <p><b>User action:</b> Recreate the key.</p> <p><b>REASONCODES:</b> TSS 02F (047)</p>
2B14 (11028)	<p>The modulus of the PKA key is too short for processing the hash or PKCS block.</p> <p><b>User action:</b> Either use a PKA key with a larger modulus size, use a hash algorithm that generates a smaller hash (digital signature services), or specify a shorter DATA key size (symmetric key export, symmetric key generate).</p> <p><b>REASONCODES:</b> TSS 048 (072)</p>
2B18 (11032)	<p>The supplied private key can be used only for digital signature. Key management services are disallowed.</p> <p><b>User action:</b> Supply a key with key management enabled.</p> <p><b>REASONCODES:</b> TSS 040 (064)</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
2B20 (11040)	<p>The recovered encryption block was not a valid PKCS-1.2 or zero-pad format. (The format is verified according to the recovery method specified in the rule-array.) If the recovery method specified was PKCS-1.2, refer to PKCS-1.2 for the possible error in parsing the encryption block.</p> <p><b>User action:</b> Ensure that the parameters passed to CSNDSYI or CSNFSYI are correct. Possible causes for this error are incorrect values for the RSA private key or incorrect values in the <i>RSA_enciphered_key</i> parameter, which must be formatted according to PKCS-1.2 or zero-pad rules when created.</p> <p><b>REASONCODES:</b> TSS 42 (66)</p>
2B24 (11044)	<p>The first section of a supplied PKA token was not a private or public key section.</p> <p><b>User action:</b> Recreate the key.</p> <p><b>REASONCODES:</b> TSS 0B5(181)</p>
2B28 (11048)	<p>The eyecatcher on the PKA internal private token is not valid.</p> <p><b>User action:</b> Reimport the private token using the PKA key import callable service.</p>
2B2C (11052)	<p>An incorrect PKA token was supplied. One of the following situations is possible:</p> <ul style="list-style-type: none"> <li>• The service requires a private key token of the correct type.</li> <li>• The supplied token may be of a type that is not supported on this system.</li> </ul> <p><b>User action:</b> Check that the supplied token is:</p> <ul style="list-style-type: none"> <li>• a PKA private key token of the correct type.</li> <li>• a type supported by this system.</li> </ul>
2B30 (11056)	<p>The input PKA token contains length fields that are not valid.</p> <p><b>User action:</b> Recreate the key token.</p>
2B38 (11064)	<p>The RSA-OAEP block did not verify when it decomposed. The block type is incorrect (must be X'03').</p> <p><b>User action:</b> Recreate the RSA-OAEP block.</p> <p><b>REASONCODES:</b> TSS 2CF (719)</p>
2B3C (11068)	<p>The RSA-OAEP block did not verify when it decomposed. The verification code is not correct (must be all zeros).</p> <p><b>User action:</b> Recreate the RSA-OAEP block.</p> <p><b>REASONCODES:</b> TSS 2D1 (721)</p>
2B40 (11072)	<p>The RSA-OAEP block did not verify when it decomposed. The random number I is not correct (must be non-zero with the high-order bit equal to zero).</p> <p><b>User action:</b> Recreate the RSA-OAEP block.</p> <p><b>REASONCODES:</b> TSS 2D0 (720)</p>
2B48 (11080)	<p>The RSA public or private key specified a modulus length that is incorrect for this service.</p> <p><b>User action:</b> Re-invoke the service with an RSA key with the proper modulus length.</p> <p><b>REASONCODES:</b> See reason codes 41 (65) and 2F8 (760)</p>

Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
2B4C (11084)	This service requires an RSA public key and the key identifier specified is not a public key. <b>User action:</b> Re-invoke the service with an RSA public key.
2B50 (11088)	This service requires an RSA private key that is for signature use only. <b>User action:</b> Re-invoke the service with a supported private key.
2B54 (11092)	There was an invalid subsection in the PKA token. <b>User action:</b> Correct the PKA token.
2B58 (11096)	This service requires an RSA private key that is for signature use. The specified key may be used for key management purposes only. <b>User action:</b> Re-invoke the service with a supported private key. <b>REASONCODES:</b> TSS 040 (064)
3E80 (16000)	RACF failed your request to use this service. <b>User action:</b> Contact your ICSF or RACF administrator if you need this service.
3E84 (16004)	RACF failed your request to use the key label. This may be caused by either CSFKEYS or XCSFKEY class, depending on the setting of the Granular Keylabel Access Controls and the type of token provided. <b>User action:</b> Contact your ICSF or RACF administrator if you need this key.
3E88 (16008)	Clear key generation denied by policy. Secure PKCS #11 services are not available and caller's RACF access to CRYPTOZ class resource CLEARKEY.token-label does not permit the generation of non-secure (clear) PKCS #11 keys. <b>User action:</b> Contact your ICSF administrator <b>ICSF administrator action:</b> Either configure ICSF for secure PKCS #11 services or have your RACF administrator grant the user authority to use clear keys
3E8C (16012)	You requested the conversion service, but you are not running in an authorized state. <b>User action:</b> You must be running in supervisor state to use the conversion service. Contact your ICSF administrator.
3E90 (16016)	The input/output field contained a valid internal token with the NOCV bit on or encryption algorithm mark, but the key type was incorrect or did not match the type of the generated or imported key. Processing failed. <b>User action:</b> Correct the calling application. <b>REASONCODES:</b> TSS 027 (039)
3E94 (16020)	You requested dynamic CKDS update services for a system key, which is not allowed. <b>User action:</b> Correct the calling application. <b>REASONCODES:</b> TSS 0B5 (181)
3E98 (16024)	You called the CKDS key record write callable service, but the key token you supplied is not valid. <b>User action:</b> Check with your ICSF administrator if you believe that this key should be in the in-storage CKDS. The administrator may be able to bring it into storage. If this key cannot be in storage, use a different label.



Table 346. Reason Codes for Return Code 8 (8) (continued)

Reason Code Hex (Decimal)	Description
3EA0 (16032)	Invalid syntax for CKDS, PKDS or TKDS label name.  <b>User action:</b> Correct <i>key_label</i> syntax.  <b>REASONCODES:</b> TSS 020 (032)
3EA4 (16036)	The CKDS key record create callable service requires that the key created not already exist in the CKDS or PKDS. A key of the same label was found.  <b>User action:</b> Make sure the application specifies the correct label. If the label is correct, contact your ICSF security administrator or system programmer.  <b>REASONCODES:</b> TSS 02C (044)
3EA8 (16040)	Data in the PKDS record did not match the expected data. This occurs if the record does not contain a null PKA token and CHECK was specified.  <b>User action:</b> If the record is to be overwritten regardless of its content, specify OVERLAY.
3EAC (16044)	One or more key labels specified as input to the PKA key generate or PKA key import service incorrectly refer to a retained private key. If generating a retained private key, this error may result from one of these conditions: <ul style="list-style-type: none"> <li>• The private key name of the retained private key being generated is the same as an existing PKDS record, but the PKDS record label was not specified as the input skeleton (source) key identifier.</li> <li>• The label specified in the <i>generated_key_token</i> parameter as the target for the retained private key was not the same as the private key name</li> </ul> <p>If generating or importing a non-retained key, this error occurs when the label specified as the target key specifies a retained private key. The retained private key cannot be over-written.</p> <p><b>User action:</b> Make sure the application specifies the correct label. If the label is correct, contact your ICSF security administrator or system programmer.</p>
3EB0 (16048)	Retained keys on the PKDS cannot be deleted or updated using the PKDS key record delete or PKDS key record write callable services, respectively.  <b>User action:</b> Use the retained key delete callable service to delete retained keys.
Reason code 0, return code 308 (776)	RACF failed your request to use this service.  <b>User action:</b> Contact your ICSF or RACF administrator if you need this service.
Reason code 1, return code 308 (776)	RACF failed your request to use the key label.  <b>User action:</b> Contact your ICSF or RACF administrator if you need this key.
06E (110)-PAN, 028 (040)-ser. code, 02A (042)-exp. date, 066 (102)-dec table, 067 (103)-val. table, 06C (198)-pad data	The PAN, expiration date, service code, decimalization table data, validation data, or pad data is not numeric (X'F0' through X'F9'). The parameter must be character representations of numerics or hexadecimal data.  <b>User action:</b> Review the numeric parameters or fields required in the service that you called and change to the format and values required.

## Reason Codes for Return Code C (12)

Table 347 on page 822 lists reason codes returned from callable services that give return code 12. These reason codes indicate that the call to the callable service was



not successful. Either cryptographic processing did not take place, or the last cryptographic unit was switched offline. Therefore, no output parameters were filled.

**Note:** The higher-order halfword of the reason code field for return code C (12) may contain additional coding. See reason codes 1790, 273C, and 2740 in this table. For example, in the reason code 42738, the 4 is an SVC 99 error code and the 2738 is listed in this table:

Table 347. Reason Codes for Return Code C (12)

Reason Code Hex (Decimal)	Description
0 (0)	<p>ICSF is not available. One of the following situations is possible:</p> <ul style="list-style-type: none"> <li>• ICSF is not started</li> <li>• ICSF is started, but does not have access to any cryptographic units.</li> <li>• ICSF is started, but the DES-MK, AES-MK, or ECC-MK is not defined.</li> <li>• ICSF is started, but the requested function is not available. For instance, an ECC operation was requested but the required hardware is not installed.</li> </ul> <p><b>User action:</b> Check the availability of ICSF with your ICSF administrator.</p> <p>OR</p> <p>CKDS Key Record Create2 or CKDS Key Record Write2 was called to add a variable-length key record to a fixed-length CKDS. A variable-length symmetric key token can only be added to a CKDS that supports variable-length records.</p> <p><b>User action:</b> Contact the security administrator or system programmer to activate (refresh) a CKDS that supports variable-length records.</p>
4 (4)	<p>The CKDS or PKDS management service you called is not available because it has been disallowed by the ICSF User Control Functions panel.</p> <p><b>User action:</b> Contact the security administrator or system programmer to determine why the CKDS or PKDS management services have been disallowed.</p>
8 (8)	<p>The service or algorithm is not available on current hardware. Your request cannot be processed.</p> <p><b>User action:</b> Correct the calling program or run on applicable hardware.</p>
C (12)	<p>The service that you called is unavailable because the installation exit for that service had previously failed.</p> <p><b>User action:</b> Contact your ICSF administrator or system programmer.</p>
10 (16)	<p>A requested installation service routine could not be found. Your request was not processed.</p> <p><b>User action:</b> Contact your ICSF administrator or system programmer.</p>
1C (28)	<p>Cryptographic asynchronous processor failed.</p> <p><b>User action:</b> Contact your IBM support center.</p>
20 (32)	<p>Cryptographic asynchronous instruction was not executed.</p> <p><b>User action:</b> Ensure cryptographic services are enabled.</p>
28 (40)	<p>The callable service that you called is unsupported for AMODE(64) applications. Your request cannot be processed.</p>

Table 347. Reason Codes for Return Code C (12) (continued)

Reason Code Hex (Decimal)	Description
2C (44)	The callable service that you called was linked with the AMODE(64) stub. The application is not running AMODE(64). Your request cannot be processed.  <b>User action:</b> Link your application with the service stub with the appropriate addressing mode.
0C5 (197)	I/O error reading or writing to the DASD copy of the CKDS or PKDS in use by ICSF.  <b>User action:</b> Contact your ICSF security administrator or system programmer. The RPL feedback code will be placed in the high-order halfword of the reason code field.
144 (324)	There was insufficient coprocessor memory available to process your request. This could include the Flash EPROM used to store keys, profiles and other application data.  <b>User action:</b> Contact your system programmer or the IBM Support Center.
2FC (764)	The master key is not in a valid state.  <b>User action:</b> Contact your ICSF administrator.  <b>REASONCODES:</b> ICSF 2B08 (11016)
301 (769)	A cryptographic internal device driver component detected data contained in a cryptographic request that is not valid.
7D6 (2006)	TKE: PCB service error.
7D7 (2007)	TKE: Change type in PCB is not recognized.
7DF (2015)	Domain in CPRB not enabled by EMB mask.
7E1 (2017)	MKVP mismatch on Set MK.
7E5 (2021)	Cryptographic coprocessor adapter disabled.
7E9 (2025)	Enforcement mask error.
7F3 (2035)	Intrusion latch has been tripped. Services disabled.
7F5 (2037)	The domain specified is not valid.
7FB (2043)	OA certificate not found.
819 (2073)	The PCIXCC or CCA Crypto Express coprocessor has been disabled on the Support Element. It must be enabled on the Support Element prior to TKE accessing it.  <b>User action:</b> Permit the selected PCIXCC or CCA Crypto Express coprocessor for TKE Commands on the Support Element and then re-open the Host on TKE.
835 (2101)	AES flags in the function control vector are not valid.
BBD (3005)	The KDS I/O subtask timed out waiting for an exclusive ENQ on the SYSZxKDS.xKDSdsn resource, where x indicates the KDS type (C for CKDS, P for PKDS, and T for TKDS). A timeout will occur if one or more members of the ICSF sysplex group has not relinquished its ENQ on the resource. The KDS update operation has failed.  <b>User action:</b> Issue D GRS,RES=(nnnnn), where nnnnn is the KDS resource name from message CSFM302A, to determine which system or systems hold the resource. Determine if action should be taken to cause the holding system to release its ENQ on the KDS resource.
BBE (3006)	Failure after exhausting retry attempts. IXCMSSGO issued from CSFMIOST.  <b>User action:</b> Contact your system programmer or the IBM Support Center.

Table 347. Reason Codes for Return Code C (12) (continued)

Reason Code Hex (Decimal)	Description
BBF (3007)	<p>The CKDS service failed due to unexpected termination of the ICSF Cross-System Services environment. The termination of the ICSF Cross-System Services environment was caused by a failure when ICSF issued the IXCMSGI macro. Message CSFM603 has been issued.</p> <p><b>User action:</b> Report the occurrence of this error to your ICSF system programmer.</p>
BC6 (3014)	<p>There is an I/O error reading or writing to the DASD copy of the TKDS in use by ICSF.</p> <p><b>User action:</b> Report the occurrence of this error to your ICSF system programmer.</p>
BC7 (3015)	<p>A bad header record is detected for the TKDS in CSFM TDSL.</p> <p><b>User action:</b> Report the occurrence of this error to your ICSF system programmer.</p>
BCF (3023)	<p>The PKCS #11 TKDS is not available for processing.</p> <p><b>User action:</b> Report the occurrence of this error to your ICSF system programmer.</p>
BE6 (3046)	<p>An RSA retained key can no longer be generated with its key-usage flag set to allow key unwrapping (KM-ONLY or KEY-MGMT). Key usage must be SIG-ONLY.</p> <p><b>User action:</b> None required.</p>
BE8 (3048)	<p>The services using encrypted AES keys, encrypted DES, or encrypted ECC keys are not available because the master key is required but not loaded or there is no access to any cryptographic units. Your request cannot be processed.</p> <p><b>User action:</b> Check the availability of ICSF with your ICSF administrator</p>
C00 (3072)	<p>The serialization subtask terminated for an unexpected reason prior to completing the request. No dynamic CKDS or PKDS update services are possible at this point.</p> <p><b>User action:</b> Contact your system programmer who can investigate the problem and restart the I/O subtask by stopping and restarting ICSF.</p>
C01 (3073)	<p>An error occurred attempting to obtain the system ENQ for a key data set update.</p> <p><b>User action:</b> If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C03 (3075)	<p>A symmetric key token was supplied in a key identifier parameter which is wrapped using the enhanced X9.24 key wrapping method. The cryptographic coprocessors available to process the request don't support the enhanced key wrapping.</p> <p><b>User action:</b> Contact system personnel to get coprocessors installed on your system which will support the enhanced X9.24 key wrapping.</p>
C06 (3078)	<p>The CKDS was created with an unsupported LRECL.</p>
C09 (3081)	<p>An attempt was made to load a PKDS that only uses the ECC master key on a pre-HCR7780 release of ICSF. Pre-HCR7780 systems do not support the ECC master key and use of an ECC MK-only PKDS is not allowed.</p> <p><b>User Action:</b> Change the PKDS selected. Specify a PKDS that is empty, uses an RSA master key, or uses both RSA and ECC master keys.</p>
C0A (3082)	<p>A callable service generated or updated a symmetric key token and the X9.24 enhanced wrapping method was used to wrap the key. This key token is not usable on your system and ICSF will not allow the key to be generated. The key was wrapped with the enhanced wrapping method because a CEX3C or CEX4C coprocessor has the default wrapping configuration set to enhanced. This was most likely done by TKE changing the configuration.</p> <p><b>User Action:</b> Have the ICSF administrator set the default wrapping configuration to original for the LPAR that this system is running in.</p>

Table 347. Reason Codes for Return Code C (12) (continued)

Reason Code Hex (Decimal)	Description
C17 (3095)	<p>The sysplex KDS cluster members' new AES master key registers were loaded with different values during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new AES master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key.</p> <p><b>User action:</b> Ensure all sysplex KDS cluster members' new AES master key registers are loaded with the same value or all are empty and retry the function.</p>
C18 (3096)	<p>One or more sysplex KDS cluster members' new DES master key registers were loaded and others were empty during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new DES master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key.</p> <p><b>User action:</b> Ensure all sysplex KDS cluster members' new DES master key registers are loaded with the same value or all are empty and retry the function.</p>
C19 (3097)	<p>The sysplex KDS cluster members' new DES master key registers were loaded with different values during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new DES master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key.</p> <p><b>User action:</b> Ensure all sysplex KDS cluster members' new DES master key registers are loaded with the same value or all are empty and retry the function.</p>
C1A (3098)	<p>A coordinated KDS change master key was attempted with empty new master key registers. At least one of the new master key registers must be loaded with a value to perform a coordinated KDS change master key.</p> <p><b>User action:</b> Load at least one of the new master key registers on all sysplex KDS cluster members with the same value and retry the function.</p>
C1B (3099)	<p>An ICSF subtask terminated during coordinated KDS refresh or coordinated KDS change master key processing.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C1C (3100)	<p>An error occurred attempting to obtain an ENQ for performing either a coordinated KDS refresh or coordinated KDS change master key.</p> <p><b>User action:</b> The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C1D (3101)	<p>A target system (member of the sysplex KDS cluster) was unable to open the new KDS for either a coordinated KDS refresh or coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C1E (3102)	<p>One or more sysplex KDS cluster members' new AES master key registers were loaded and others were empty during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new AES master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key.</p> <p><b>User action:</b> Ensure all sysplex KDS cluster members new AES master key registers are loaded with the same value or all are empty and retry the function.</p>

Table 347. Reason Codes for Return Code C (12) (continued)

Reason Code Hex (Decimal)	Description
C2B (3115)	<p>Either a coordinated KDS refresh or coordinated KDS change master key was cancelled.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C2C (3116)	<p>A catalog problem occurred during either a coordinated KDS refresh or coordinated KDS change master key. The problem occurred when looking up either the active KDS or new KDS in the catalog.</p> <p><b>User action:</b> Ensure both the active KDS and new KDS are cataloged and retry the function.</p>
C2D (3117)	<p>A coordinated KDS refresh or coordinated KDS change master key was attempted on a system with a level of hardware that is not supported by the function. This reason code is also used if the licensed internal code (LIC) level on the originating system is lower than the licensed internal code (LIC) level on 1 or more of the other sysplex KDS cluster members.</p> <p><b>User action:</b> Refer to "Coordinated KDS Administration (CSFCRC and CSFCRC6)" on page 628 for a list of supported hardware levels. Perform the coordinated KDS function from the system running the highest level of licensed internal code (LIC).</p>
C2E (3118)	<p>A coordinated KDS change master key was attempted with the DES new master key register loaded but with no current DES master key set. In order to perform a coordinated KDS change master key to a new DES master key, a valid DES master key must have previously been set.</p> <p><b>User action:</b> Set a valid DES master key and then use the coordinated KDS change master key to change the DES master key.</p>
C2F (3119)	<p>A coordinated KDS change master key was attempted with the AES new master key register loaded but with no current AES master key set. In order to perform a coordinated KDS change master key to a new AES master key, a valid AES master key must have previously been set.</p> <p><b>User action:</b> Set a valid AES master key and then use the coordinated KDS change master key to change the AES master key.</p>
C32 (3122)	<p>A sysplex communication failure occurred during either coordinated KDS refresh or coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated CKDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C33 (3123)	<p>A failure occurred processing KDS updates during a coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C34 (3124)	<p>An internal failure occurred in a coordinated KDS subtask while performing either a coordinated KDS refresh or a coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>

Table 347. Reason Codes for Return Code C (12) (continued)

Reason Code Hex (Decimal)	Description
C35 (3125)	<p>An internal failure occurred in a coordinated KDS subtask while performing either a coordinated KDS refresh or a coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C36 (3126)	<p>An internal failure occurred in the sysplex subtask while performing either a coordinated KDS refresh or coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C37 (3127)	<p>An internal failure occurred in the serialization subtask while performing either a coordinated KDS refresh or coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C38 (3128)	<p>An internal failure occurred in the I/O subtask while performing a coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function may be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C3A (3130)	<p>A target system (member of the sysplex KDS cluster) is not being responsive to a system that is originating either a coordinated KDS refresh or coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C3B (3131)	<p>The active KDS could not be reenciphered to the new KDS during a coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C3E (3134)	<p>A failure occurred either renaming the active KDS to the archive KDS or renaming the new KDS to the active KDS during a coordinated KDS refresh or coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>



Table 347. Reason Codes for Return Code C (12) (continued)

Reason Code Hex (Decimal)	Description
C40 (3136)	<p>A coordinated KDS refresh or coordinated KDS change master key was originated from a system at a lower ICSF FMID release level than one or more of the target systems (sysplex KDS cluster members). The coordinated KDS functions must be originated from a system running the highest ICSF FMID level.</p> <p><b>User action:</b> Retry the function from a sysplex KDS cluster member running the highest ICSF FMID level.</p>
C41 (3137)	<p>An internal failure occurred during the set master key step of a coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C42 (3138)	<p>A failure occurred trying to back out from a failed rename of the active KDS to the archive KDS or a failed rename of the new KDS to the active KDS during a coordinated KDS refresh or coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C43 (3139)	<p>A failure occurred switching the new KDS to the active KDS during either a coordinated KDS refresh or a coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
C44 (3140)	<p>A coordinated KDS refresh or a coordinated KDS change master key failed because one of the target systems (sysplex KDS cluster members) had not finished ICSF initialization.</p> <p><b>User action:</b> Allow all sysplex KDS cluster members to finish ICSF initialization and retry the function.</p>
C45 (3141)	<p>A coordinated KDS change master key was attempted with the RSA new master key register loaded but with no current RSA master key set. In order to perform a coordinated KDS change master key to a new RSA master key, a valid RSA master key must have previously been set.</p> <p><b>User action:</b> Set a valid RSA master key and then use the coordinated KDS change master key to change the RSA master key.</p>
C46 (3142)	<p>A coordinated KDS change master key was attempted with the ECC new master key register loaded but with no current ECC master key set. In order to perform a coordinated KDS change master key to a new ECC master key, a valid ECC master key must have previously been set.</p> <p><b>User action:</b> Set a valid ECC master key and then use the coordinated KDS change master key to change the ECC master key.</p>
C47 (3143)	<p>A coordinated KDS change master key was attempted with the PKCS #11 new master key register loaded but with no current PKCS #11 master key set. In order to perform a coordinated KDS change master key to a new PKCS #11 master key, a valid PKCS #11 master key must have previously been set.</p> <p><b>User action:</b> Set a valid PKCS #11 master key and then use the coordinated KDS change master key to change the PKCS #11 master key.</p>

Table 347. Reason Codes for Return Code C (12) (continued)

Reason Code Hex (Decimal)	Description
C48 (3144)	<p>The sysplex KDS cluster members' new RSA master key registers were loaded with different values during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new RSA master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key.</p> <p><b>User action:</b> Ensure all sysplex KDS cluster members' new RSA master key registers are loaded with the same value or are all empty, and retry the function.</p>
C49 (3145)	<p>The sysplex KDS cluster members' new ECC master key registers were loaded with different values during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new ECC master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key.</p> <p><b>User action:</b> Ensure all sysplex KDS cluster members' new ECC master key registers are loaded with the same value or are all empty, and retry the function.</p>
C4A (3146)	<p>One or more sysplex KDS cluster members' new RSA master key registers were loaded and others were empty during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new RSA master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key.</p> <p><b>User action:</b> Ensure all sysplex KDS cluster members' new RSA master key registers are loaded with the same value or all are empty and retry the function.</p>
C4B (3147)	<p>One or more sysplex KDS cluster members' new ECC master key registers were loaded and others were empty during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new ECC master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key.</p> <p><b>User action:</b> Ensure all sysplex KDS cluster members' new ECC master key registers are loaded with the same value or are all empty and retry the function.</p>
C4C (3148)	<p>The sysplex KDS cluster members' new PKCS #11 master key registers were loaded with different values during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new PKCS #11 master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key.</p> <p><b>User action:</b> Ensure all sysplex KDS cluster members' new PKCS #11 master key registers are loaded with the same value or are all empty and retry the function.</p>
C4D (3149)	<p>One or more sysplex KDS cluster members' new P11 master key registers were loaded and others were empty during a coordinated KDS change master key. All sysplex KDS cluster members' (same active KDS) new P11 master key registers must be loaded with the same value or all must be empty when performing a coordinated KDS change master key.</p> <p><b>User action:</b> Ensure all sysplex KDS cluster members' new P11 master key registers are loaded with the same value or are all empty and retry the function.</p>
C80 (3200)	<p>Key object's compliance mode is different than current setting of the Enterprise PKCS #11 coprocessors</p> <p><b>User action:</b> Contact your ICSF administrator or system programmer.</p> <p><b>ICSF administrator action:</b> The compliance mode setting on the Enterprise PKCS #11 coprocessors must be set to a value at least as restrictive as the key object that failed. Using the PKCS #11 Token Browser ISPF panels, examine the IBM CARD COMPLIANCE value for the key that failed. Set each Enterprise PKCS #11 coprocessor to this value using TKE.</p>



Table 347. Reason Codes for Return Code C (12) (continued)

Reason Code Hex (Decimal)	Description
1779 (6009)	<p>One or more target systems (sysplex KDS cluster members) did not successfully load the new KDS during a coordinated KDS refresh or coordinated KDS change master key. This a common result of an unresponsive target system.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated CKDS administration failure. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
177A (6010)	<p>A failure occurred during a coordinated KDS change master key operation while attempting to reencipher a secure P11 object.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
1780 (6016)	<p>A DASD IO error was encountered during access of the CKDS, PKDS, or TKDS.</p> <p><b>User action:</b> Contact your ICSF security administrator or system programmer. The SVC 99 error code will be placed in the high-order halfword of the reason code field.</p>
178C (6028)	<p>ESTAE could not be established in common I/O routines.</p> <p><b>User action:</b> Contact your system programmer or the IBM Support Center.</p>
1790 (6032)	<p>The dynamic allocation of the DASD copy of the CKDS, PKDS, or TKDS in use by ICSF failed.</p> <p><b>User action:</b> Contact your ICSF security administrator or system programmer. The SVC 99 error code will be placed in the high-order halfword of the reason code field.</p>
1794 (6036)	<p>A dynamic deallocation error occurred when closing and deallocating a CKDS, PKDS, or TKDS.</p> <p><b>User action:</b> Contact your security administrator or system programmer. The SVC 99 error code will be placed in the high-order halfword of the reason code field.</p>
1795 (6037)	<p>A failure occurred routing KDS updates to the originating system of a coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
1796 (6038)	<p>The I/O subtask became out of sync with the sysplex KDS cluster during a coordinated KDS change master key. The I/O subtask will be restarted to get back in sync with the sysplex KDS cluster.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>
1797 (6039)	<p>ICSF was unable to attach a coordinated KDS subtask for either a coordinated KDS refresh or coordinated KDS change master key.</p> <p><b>User action:</b> Refer to the <i>z/OS Cryptographic Services ICSF Administrator's Guide</i> for information on recovering from a coordinated KDS administration failure. The function can be retried. If the error is common and persistent, contact your system programmer or the IBM Support Center.</p>

Table 347. Reason Codes for Return Code C (12) (continued)

Reason Code Hex (Decimal)	Description
2724 (10020)	<p>A key retrieved from the in-storage CKDS failed the MAC verification (MACVER) check and is unusable.</p> <p><b>User action:</b> Contact your ICSF administrator.</p>
2728 (10024)	<p>A key retrieved from the in-storage CKDS or a key to be written to the PKDS was rejected for use by the installation exit.</p> <p><b>User action:</b> Contact your ICSF administrator or system programmer.</p>
272C (10028)	<p>You cannot use the secure key import or multiple secure key import callable services because the cryptographic unit is not enabled for processing. The cryptographic unit is not in special secure mode or is disabled in the environment control mask (ECM).</p> <p><b>User action:</b> Contact your ICSF administrator (your administrator can enable the processing mode or the ECM).</p>
2734 (10036)	<p>More than one key with the same label was found in the CKDS or PKDS. This function requires a unique key per label. The probable cause may be the use of an incorrect label pointing to a key type that allows multiple keys per label.</p> <p><b>User action:</b> Make sure the application specifies the correct label. If the label is correct, contact your ICSF security administrator or system programmer to verify the contents of the CKDS or PKDS.</p>
273C (10044)	<p>OPEN of the PKDS in use by ICSF failed.</p> <p><b>User action:</b> Contact your ICSF security administrator or system programmer.</p>
2740 (10048)	<p>I/O error reading or writing to the DASD copy of the CKDS or PKDS in use by ICSF.</p> <p><b>User action:</b> Contact your ICSF security administrator or system programmer. The RPL feedback code will be placed in the high-order halfword of the reason code field.</p> <p><b>REASONCODES:</b> TSS 0C5 (197)</p>
2744 (10052)	<p>Automatic REFRESH to free storage in the linear section of the CKT failed.</p> <p><b>User action:</b> Contact your ICSF security administrator or system programmer and request that a REFRESH be done.</p>
274C (10060)	<p>The I/O subtask terminated for an unexpected reason prior to completing the request. No dynamic CKDS or PKDS update services are possible at this point.</p> <p><b>User action:</b> Contact your system programmer who can investigate the problem and restart the I/O subtask by stopping and restarting ICSF.</p>
2B04 (11012)	<p>This function is disabled in the environment control mask (ECM).</p> <p><b>User action:</b> Contact your ICSF administrator.</p>
2B08 (11016)	<p>The master key is not in a valid state.</p> <p><b>User action:</b> Contact your ICSF administrator.</p> <p><b>REASONCODES:</b> TSS 2FC (764)</p>
2B0C (11020)	<p>The modulus of the public or private key is larger than allowed and configured in the CCC or FCV. You cannot use this key on this system.</p> <p><b>User action:</b> Regenerate the key with a smaller modulus size.</p>

Table 347. Reason Codes for Return Code C (12) (continued)

Reason Code Hex (Decimal)	Description
2B10 (11024)	The system administrator has used the ICSF User Control Functions panel to disable the RSA functions.  <b>User action:</b> Wait until administrator functions are complete and the RSA functions are again enabled.
2B18 (11032)	A CAMQ is valid for PKSC but not for PKA.  <b>User action:</b> Contact your ICSF administrator.
2B1C (11036)	A PKDS is not available for processing.  <b>User action:</b> Contact your ICSF administrator.
2B20 (11040)	The PKDS Control Record hash pattern is not valid.  <b>User action:</b> Contact your ICSF administrator.
2B24 (11044)	The PKDS could not be accessed.  <b>User action:</b> Contact your ICSF administrator.
2B28 (11048)	The PCICC, PCIXCC, or CCA Crypto Express coprocessor failed.  <b>User action:</b> Contact your IBM support center.
2B2C (11052)	The specific PCICC, PCIXCC, or CCA Crypto Express coprocessor requested for service is temporarily unavailable. PKDS could not be accessed. The specific coprocessor may be attempting some recovery action. If recovery action is successful, the coprocessor will be made available. If the recovery action fails, the coprocessor will be made permanently unavailable.  <b>User action:</b> Retry the function.
2B30 (11056)	The PCICC, PCIXCC, CEX2C, or CEX3C failed. The response from the processor was incomplete.  <b>User action:</b> Contact your IBM support center.
2B34 (11060)	The service could not be performed because the required PCICC, PCIXCC, or CCA Crypto Express coprocessor was not active, or did not have a master key set.  <b>User action:</b> If the service required a specific PCICC, PCIXCC, or CCA Crypto Express coprocessor, verify that the value specified is correct. Reissue the request when the required coprocessor is available, and has the master key set.
2B38 (11064)	Service could not be performed because of a hardware error on the PCICC, PCIXCC, or CCA Crypto Express coprocessor.
2B40 (11072)	Coprocessor configuration change. A CCA or EP11 coprocessor has been configured as an accelerator. TKE does not recognize coprocessors configured as accelerators.
2B41 (11073)	Coprocessor configuration change. Either a CCA coprocessor has been reconfigured to be a EP11 coprocessor, or a PKCS #11 coprocessor has been reconfigured to be a CCA coprocessor.
2EDC (11996)	The Cryptographic Coprocessor Feature is not available for CKDS initialization because the cryptographic unit is not in special secure mode.  <b>User action:</b> Contact your ICSF administrator.
2EE0 (12000)	You cannot use the Clear PIN generate callable service because the cryptographic unit is not enabled for processing. The cryptographic unit is not in special secure mode.  <b>User action:</b> Contact your ICSF administrator who can enable the processing mode.

Table 347. Reason Codes for Return Code C (12) (continued)

Reason Code Hex (Decimal)	Description
8CA2 (36002)	CSFPFI was called to set the RSA master key in a PCIXCC, CEX2C or CEX3C. This function is disabled because dynamic RSA master key change is enabled and the RSA master key can only be changed from the ICSF TSO Change asymmetric master key utility.
8CB4 (36020)	A refresh of the CKDS failed because the DASD copy of the CKDS is enciphered under the wrong master key. This may have resulted from an automatic refresh during processing of the CKDS key record create callable service.  <b>User action:</b> Contact your ICSF administrator.
8CE5 (36069)	A failure occurred during a coordinated KDS change master key operation because the DASD copy of the TKDS is enciphered under the wrong master key.  <b>User action:</b> Contact your ICSF administrator.
8D14 (36116)	The PKDS specified for refresh, reencipher or activate has an incorrect dataset attribute.  <b>User action:</b> Create a larger PKDS. See <i>z/OS Cryptographic Services ICSF System Programmer's Guide</i> .
8D3C (36156)	A PKCS #11 service is being requested. The service is disabled due to an ICSF FIPS self test failure. The request is not processed.  <b>User action:</b> Report the problem to your IBM support center
8D40 (36160)	The attempt to reencipher the CKDS failed because there is an enhanced wrapped token in the CKDS.  <b>User Action:</b> Reencipher the CKDS on a system that supports the enhanced wrapping method.
8D56 (36182)	A coprocessor failure was detected during initialization.  <b>User action:</b> The error is accompanied by the CSFM540I message. Follow instructions associated with that message.
8D5A (36186)	A request was made to reencipher a CKDS. The CKDS specified cannot be reenciphered on this release of ICSF because the CKDS contains Variable-length Symmetric key tokens with an unrecognized algorithm or key type in the associated data section. Only key tokens with a recognized algorithm or key type can be managed on this release of ICSF.  <b>User action:</b> Perform the reencipher operation on a release of ICSF which recognizes the algorithm and key type of all tokens in the specified CKDS.
8D5D (36189)	The TKDS has an incorrect dataset attribute.  <b>User action:</b> Create a TKDS with valid dataset attributes. See <i>z/OS Cryptographic Services ICSF System Programmer's Guide</i>

## Reason Codes for Return Code 10 (16)

Table 348 lists reason codes returned from callable services that give return code 16.

Table 348. Reason Codes for Return Code 10 (16)

Reason Code Hex (Decimal)	Description
4 (4)	ICSF: Your call to an ICSF callable service resulted in an abnormal ending.  <b>User action:</b> Contact your system programmer or the IBM Support Center.

Table 348. Reason Codes for Return Code 10 (16) (continued)

Reason Code Hex (Decimal)	Description
150 (336)	<p>An error occurred in the cryptographic hardware component.</p> <p><b>User action:</b> Contact your system programmer or the IBM Support Center.</p> <p><b>REASONCODES:</b> ICSF 4 (4)</p>
22C (556)	<p>The request parameter block failed consistency checking.</p> <p><b>User action:</b> Contact your system programmer or the IBM Support Center.</p> <p><b>REASONCODES:</b> ICSF 4 (4)</p>
2C4 (708)	<p>Inconsistent data was returned from the cryptographic engine.</p> <p><b>User action:</b> Contact your system programmer or the IBM Support Center.</p> <p><b>REASONCODES:</b> ICSF 4 (4)</p>
2C5 (709)	<p>Cryptographic engine internal error; could not access the master key data.</p> <p><b>User action:</b> Contact your system programmer or the IBM Support Center.</p> <p><b>REASONCODES:</b> ICSF 4 (4)</p>
2C8 (712)	<p>An unexpected error occurred in the Master Key manager.</p> <p><b>User action:</b> Contact your system programmer or the IBM Support Center.</p> <p><b>REASONCODES:</b> ICSF 4 (4)</p>

---

## Appendix B. Key Token Formats

For debugging purposes, this appendix provides the formats for AES, DES internal, external, and null key tokens and for PKA key tokens.

- “AES Internal Key Token” on page 836
- “DES Internal Key Token” on page 837
- “DES External Key Token” on page 839
- “External RKX DES Key Token” on page 839
- “DES Null Key Token” on page 841
- “Variable-length Symmetric Key Token” on page 841
- “Variable-length Symmetric Null Key Token” on page 850
- “PKA Null Key Token” on page 851
- “RSA Public Key Token” on page 851
- “RSA Private External Key Token” on page 852
  - “RSA Private Key Token, 1024-bit Modulus-Exponent External Form” on page 853
  - “RSA private key token, 4096-bit Modulus-Exponent external form” on page 854
  - “RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section external form” on page 855
  - “RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section external form” on page 856
  - “RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Form” on page 858
- “RSA Private Internal Key Token” on page 860
  - “RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for Cryptographic Coprocessor Feature” on page 861
  - “RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for PCICC, PCIXCC, or CCA Crypto Express coprocessor” on page 862
  - “RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section internal form” on page 863
  - “RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section internal form” on page 865
  - “RSA Private Key Token, 4096-bit Chinese Remainder Theorem Internal Form” on page 867
- “DSS Public Key Token” on page 869
- “DSS Private External Key Token” on page 869
- “DSS Private Internal Key Token” on page 871
- “ECC Key Token Format” on page 873
- “Trusted Block Key Token” on page 877

---

## AES Key Token Formats

### AES Internal Key Token

Table 349 shows the format for an AES internal key token.

Table 349. Internal Key Token Format

Bytes	Description										
0	X'01' (flag indicating this is an internal key token)										
1–3	Implementation-dependent bytes (X'000000' for ICSF)										
4	Key token version number (X'04')										
5	Reserved - must be set to X'00'										
6	Flag byte <table><tr><th>Bit</th><th>Meaning When Set On</th></tr><tr><td>0</td><td>Encrypted key and master key verification pattern (MKVP) are present. Off for a clear key token, on for an encrypted key token.</td></tr><tr><td>1</td><td>Control vector (CV) value in this token has been applied to the key.</td></tr><tr><td>2</td><td>No key is present or the AES MKVP is not present if the key is encrypted.</td></tr><tr><td>3- 7</td><td>Reserved. Must be set to 0.</td></tr></table>	Bit	Meaning When Set On	0	Encrypted key and master key verification pattern (MKVP) are present. Off for a clear key token, on for an encrypted key token.	1	Control vector (CV) value in this token has been applied to the key.	2	No key is present or the AES MKVP is not present if the key is encrypted.	3- 7	Reserved. Must be set to 0.
Bit	Meaning When Set On										
0	Encrypted key and master key verification pattern (MKVP) are present. Off for a clear key token, on for an encrypted key token.										
1	Control vector (CV) value in this token has been applied to the key.										
2	No key is present or the AES MKVP is not present if the key is encrypted.										
3- 7	Reserved. Must be set to 0.										
7	1-byte LRC checksum of clear key value.										
8–15	Master key verification pattern (MKVP) (For a clear AES key token this value will be hex zeros.)										
16–47	128-bit, 192-bit, or 256-bit key value, left-justified and padded on the right with hex zeros.										
48–55	8-byte control vector. (For a clear AES key token this value will be hex zeros.)										
56–57	2-byte integer specifying the length in bits of the clear key value.										
58–59	2-byte integer specifying the length in bytes of the encrypted key value. (For a clear AES key token this value will be hex zeros.)										
60–63	Token validation value (TVV). See “Token Validation Value” for more information.										

### Token Validation Value

ICSF uses the *token validation value (TVV)* to verify that a token is valid. The TVV prevents a key token that is not valid or that is overlaid from being accepted by ICSF. It provides a checksum to detect a corruption in the key token.

When an ICSF callable service generates a key token, it generates a TVV and stores the TVV in bytes 60-63 of the key token. When an application program passes a key token to a callable service, ICSF checks the TVV. To generate the TVV, ICSF performs a twos complement ADD operation (ignoring carries and overflow) on the key token, operating on four bytes at a time, starting with bytes 0-3 and ending with bytes 56-59.

## DES Key Token Formats

### DES Internal Key Token

Table 350 shows the format for a DES internal key token.

Table 350. Internal Key Token Format

Bytes	Description																		
0	X'01' (flag indicating this is an internal key token)																		
1–3	Implementation-dependent bytes (X'000000' for ICSF)																		
4	Key token version number (X'00' or X'01')																		
5	Reserved (X'00')																		
6	Flag byte <table><tr><th>Bit</th><th>Meaning When Set On</th></tr><tr><td>0</td><td>Encrypted key and master key verification pattern (MKVP) are present.</td></tr><tr><td>1</td><td>Control vector (CV) value in this token has been applied to the key.</td></tr><tr><td>2</td><td>Key is used for no control vector (NOCV) processing. Valid for transport keys only.</td></tr><tr><td>3</td><td>Key is an ANSI key-encrypting key (AKEK).</td></tr><tr><td>4</td><td>AKEK is a double-length key (16 bytes). <b>Note:</b> When bit 3 is on and bit 4 is off, AKEK is a single-length key (8 bytes).</td></tr><tr><td>5</td><td>AKEK is partially notarized.</td></tr><tr><td>6</td><td>Key is an ANSI partial key.</td></tr><tr><td>7</td><td>Export prohibited.</td></tr></table>	Bit	Meaning When Set On	0	Encrypted key and master key verification pattern (MKVP) are present.	1	Control vector (CV) value in this token has been applied to the key.	2	Key is used for no control vector (NOCV) processing. Valid for transport keys only.	3	Key is an ANSI key-encrypting key (AKEK).	4	AKEK is a double-length key (16 bytes). <b>Note:</b> When bit 3 is on and bit 4 is off, AKEK is a single-length key (8 bytes).	5	AKEK is partially notarized.	6	Key is an ANSI partial key.	7	Export prohibited.
Bit	Meaning When Set On																		
0	Encrypted key and master key verification pattern (MKVP) are present.																		
1	Control vector (CV) value in this token has been applied to the key.																		
2	Key is used for no control vector (NOCV) processing. Valid for transport keys only.																		
3	Key is an ANSI key-encrypting key (AKEK).																		
4	AKEK is a double-length key (16 bytes). <b>Note:</b> When bit 3 is on and bit 4 is off, AKEK is a single-length key (8 bytes).																		
5	AKEK is partially notarized.																		
6	Key is an ANSI partial key.																		
7	Export prohibited.																		
7	<table><tr><th>Bit</th><th>Meaning When Set On</th></tr><tr><td>0-2</td><td>Key value encryption method.<ul style="list-style-type: none"><li>000 - the key is encrypted using the original CCA method (ECB).</li><li>001 - the key is encrypted using the X9.24 enhanced method (CBC).</li></ul>These bits are ignored if the token contains no key or a clear key.</td></tr><tr><td>3-7</td><td>Reserved.</td></tr></table>	Bit	Meaning When Set On	0-2	Key value encryption method. <ul style="list-style-type: none"><li>000 - the key is encrypted using the original CCA method (ECB).</li><li>001 - the key is encrypted using the X9.24 enhanced method (CBC).</li></ul> These bits are ignored if the token contains no key or a clear key.	3-7	Reserved.												
Bit	Meaning When Set On																		
0-2	Key value encryption method. <ul style="list-style-type: none"><li>000 - the key is encrypted using the original CCA method (ECB).</li><li>001 - the key is encrypted using the X9.24 enhanced method (CBC).</li></ul> These bits are ignored if the token contains no key or a clear key.																		
3-7	Reserved.																		
8–15	Master key verification pattern (MKVP)																		
16–23	A single-length key, the left half of a double-length key, or Part A of a triple-length key. The value is encrypted under the master key when flag bit 0 is on, otherwise it is in the clear.																		
24–31	X'0000000000000000' if a single-length key, or the right half of a double-length operational key, or Part B of a triple-length operational key. The right half of the double-length key or Part B of the triple-length key is encrypted under the master key when flag bit 0 is on, otherwise it is in the clear.																		
32–39	The control vector (CV) for a single-length key or the left half of the control vector for a double-length key.																		
40–47	X'0000000000000000' if a single-length key or the right half of the control vector for a double-length operational key.																		
48–55	X'0000000000000000' if a single-length key or double-length key, or Part C of a triple-length operational key. Part C of a triple-length key is encrypted under the master key when flag bit 0 is on, otherwise it is in the clear.																		
56–58	Reserved (X'000000')																		



Table 350. Internal Key Token Format (continued)

Bytes	Description
59 bits 0 and 1	<b>B'10'</b> Indicates CDMF DATA or KEK. <b>B'00'</b> Indicates DES for DATA keys or the system default algorithm for a KEK. <b>B'01'</b> Indicates DES for a KEK.
59 bits 2 and 3	<b>B'00'</b> Indicates single-length key (version 0 only). <b>B'01'</b> Indicates double-length key (version 1 only). <b>B'10'</b> Indicates triple-length key (version 1 only).
59 bits 4 –7	B'0000'
60–63	Token validation value (TVV).

**Note:** A key token stored in the CKDS will not have an MKVP or TVV. Before such a key token is used, the MKVP is copied from the CKDS header record and the TVV is calculated and placed in the token. See “Token Validation Value” on page 836 for more information.

## DES External Key Token

Table 351 shows the format for a DES external key token.

Table 351. Format of External Key Tokens

Bytes	Description						
0	X'02' (flag indicating an external key token)						
1	Reserved (X'00')						
2–3	Implementation-dependent bytes (X'0000' for ICSF)						
4	Key token version number (X'00' or X'01')						
5	Reserved (X'00')						
6	Flag byte  <table> <tr> <th>Bit</th><th>Meaning When Set On</th></tr> <tr> <td>0</td><td>Encrypted key is present.</td></tr> <tr> <td>1</td><td>Control vector (CV) value has been applied to the key.</td></tr> </table> <p>Other bits are reserved and are binary zeros.</p>	Bit	Meaning When Set On	0	Encrypted key is present.	1	Control vector (CV) value has been applied to the key.
Bit	Meaning When Set On						
0	Encrypted key is present.						
1	Control vector (CV) value has been applied to the key.						
7	<table> <tr> <th>Bit</th><th>Meaning When Set On</th></tr> <tr> <td>0–2</td><td>Key value encryption method.  <ul style="list-style-type: none"> <li>000 - the key is encrypted using the original CCA method (ECB).</li> <li>001 - the key is encrypted using the X9.24 enhanced method (CBC).</li> </ul> <p>These bits are ignored if the token contains no key or a clear key.</p> </td></tr> <tr> <td>3–7</td><td>Reserved.</td></tr> </table>	Bit	Meaning When Set On	0–2	Key value encryption method. <ul style="list-style-type: none"> <li>000 - the key is encrypted using the original CCA method (ECB).</li> <li>001 - the key is encrypted using the X9.24 enhanced method (CBC).</li> </ul> <p>These bits are ignored if the token contains no key or a clear key.</p>	3–7	Reserved.
Bit	Meaning When Set On						
0–2	Key value encryption method. <ul style="list-style-type: none"> <li>000 - the key is encrypted using the original CCA method (ECB).</li> <li>001 - the key is encrypted using the X9.24 enhanced method (CBC).</li> </ul> <p>These bits are ignored if the token contains no key or a clear key.</p>						
3–7	Reserved.						
8–15	Reserved (X'0000000000000000')						
16–23	Single-length key or left half of a double-length key, or Part A of a triple-length key. The value is encrypted under a transport key-encrypting key when flag bit 0 is on, otherwise it is in the clear.						
24–31	X'0000000000000000' if a single-length key or right half of a double-length key, or Part B of a triple-length key. The right half of a double-length key or Part B of a triple-length key is encrypted under a transport key-encrypting key when flag bit 0 is on, otherwise it is in the clear.						
32–39	Control vector (CV) for single-length key or left half of CV for double-length key						
40–47	X'0000000000000000' if single-length key or right half of CV for double-length key						
48–55	X'0000000000000000' if a single-length key, double-length key, or Part C of a triple-length key. This key part is encrypted under a transport key-encrypting key when flag bit 0 is on, otherwise it is in the clear.						
56–58	Reserved (X'000000')						
59 bits 0 and 1	B'00'						
59 bits 2 and 3	<table> <tr> <td><b>B'00'</b></td><td>Indicates single-length key (version 0 only).</td></tr> <tr> <td><b>B'01'</b></td><td>Indicates double-length key (version 1 only).</td></tr> <tr> <td><b>B'10'</b></td><td>Indicates triple-length key (version 1 only).</td></tr> </table>	<b>B'00'</b>	Indicates single-length key (version 0 only).	<b>B'01'</b>	Indicates double-length key (version 1 only).	<b>B'10'</b>	Indicates triple-length key (version 1 only).
<b>B'00'</b>	Indicates single-length key (version 0 only).						
<b>B'01'</b>	Indicates double-length key (version 1 only).						
<b>B'10'</b>	Indicates triple-length key (version 1 only).						
59 bits 4–7	B'0000'						
60–63	Token validation value (see “Token Validation Value” on page 836 for a description).						

## External RKX DES Key Token

Table 352 on page 840 defines an external DES key-token called an *RKX key-token*. An RKX key-token is a special token used exclusively by the Remote Key Export (CSNDRKX and CSNFRKX) and DES key-storage callable services (for example, CKDS Key Record Write). No other callable services use or reference an RKX

key-token or key-token record. For additional information about the usage of RKX key tokens, see “Remote Key Loading” on page 35.

**Note:** Callable services other than the Remote Key Export and the DES key-storage callable services do not support RKX key tokens or RKX key token records.

As can be seen in the table, RKX key tokens are 64 bytes in length, have a token identifier flag (X'02'), a token version number (X'10'), and room for encrypted keys like normal CCA DES key tokens. Unlike normal CCA DES key-tokens, RKX key tokens do not have a control vector, flag bits, and a token-validation value. In addition, they have a confounder value, a MAC value, and room for a third encrypted key.

*Table 352. External RKX DES key-token format, version X'10'*

Offset	Length	Meaning
00	1	X'02' (a token identifier flag that indicates an external key-token)
01	3	Reserved, binary zero
04	1	The token version number (X'10')
05	2	Reserved, binary zero
07	1	Key length in bytes, including confounder
08	8	Confounder
16	8	Key left
24	8	Key middle (binary zero if not used)
32	8	Key right (binary zero if not used)
40	8	<p>Rule ID</p> <p>The trusted block rule identifier used to create this key token. A subsequent call to Remote Key Export (CSNDRKX or CSNFRKX) can use this token with a trusted block rule that references the rule ID that must have been used to create this token. The trusted block rule can be compared with this rule ID for verification purposes.</p> <p>The Rule ID is an 8-byte string of ASCII characters, left justified and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.</p>
48	8	Reserved, binary zero
56	8	<p>MAC value</p> <p>ISO 16609 TDES CBC-mode MAC, computed over the 56 bytes starting at offset 0 and including the encrypted key value and the rule ID using the same MAC key that is used to protect the trusted block itself.</p> <p>This MAC value guarantees that the key and the rule ID cannot be modified without detection, providing integrity and binding the rule ID to the key itself. This MAC value must verify with the same trusted block used to create the key, thus binding the key structure to that specific trusted block.</p>

**Notes:**

1. A fixed, randomly derived variant is exclusive-ORed with the MAC key before it is used to encipher the generated or exported key and confounder.

2. The MAC key is located within a trusted block (internal format) and can be recovered by decipherment under a variant of the PKA master key.
3. The trusted block is originally created in external form by the Trusted Block Create callable service and then converted to internal form by the PKA Key Import callable service prior to the Remote Key Export call.

## DES Null Key Token

Table 353 shows the format for a DES null key token.

Table 353. Format of Null Key Tokens

Bytes	Description
0	X'00' (flag indicating this is a null key token).
1–15	Reserved (set to binary zeros).
16–23	Single-length encrypted key, or left half of double-length encrypted key, or Part A of triple-length encrypted key.
24–31	X'0000000000000000' if a single-length encrypted key, the right half of double-length encrypted key, or Part B of triple-length encrypted key.
32–39	X'0000000000000000' if a single-length encrypted key or double-length encrypted key.
40–47	Reserved (set to binary zeros).
48–55	Part C of a triple-length encrypted key.
56–63	Reserved (set to binary zeros).

## Variable-length Symmetric Key Token Formats

### Variable-length Symmetric Key Token

The following table presents the presents the format for a variable-length symmetric key token. The length of the token depends on the key type and algorithm.

Table 354. Variable-length Symmetric Key Token

Offset (Dec)	Length of Field (Bytes)	Description
<b>Header</b>		
0	1	Token flag X'00' for null token X'01' for internal tokens X'02' for external tokens
1	1	Reserved (X'00')
2	2	Length of the token in bytes
4	1	Token version number X'05'
5	3	Reserved (X'000000')
<b>Wrapping information</b>		

Table 354. Variable-length Symmetric Key Token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
8	1	Key material state. X'00' no key present (internal or external) X'01' key is clear (internal) X'02' key is encrypted under a key-encrypting key (external) X'03' key is encrypted under the master key (internal)
9	1	Key verification pattern (KVP) type. X'00' No KVP X'01' AES master key verification pattern X'02' key-encrypting key verification pattern
10	16	Verification pattern of the key used to wrap the payload. Value is left justified.
26	1	Wrapping method - This value indicates the wrapping method used to protect the data in the encrypted section. X'00' key is in the clear X'02' AESKW X'03' PKOAEP2
27	1	Hash algorithm used in wrapping algorithm. • For wrapping method X'00' X'00' None. For clear key tokens. • For wrapping method X'02' X'02' SHA-256 • For wrapping method X'03' X'01' SHA-1 X'02' SHA-256 X'04' SHA-384 X'08' SHA-512
28	2	Reserved (X'0000')
<b>AESKW Components: Associated data and clear key or encrypted AESKW payload</b>		
<b>Associated data section</b>		
30	1	Associated data version (X'01')
31	1	Reserved (X'00')
32	2	Length of the associated data in bytes: <i>adl</i>
34	1	Length of the key name in bytes: <i>kl</i>
35	1	Length of the IBM extended associated data in bytes: <i>iead</i>
36	1	Length of the installation-definable associated data in bytes: <i>uad</i>
37	1	Reserved (X'00')
38	2	Length of the payload in bits: <i>pl</i>
40	1	Reserved (X'00')

Table 354. Variable-length Symmetric Key Token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
41	1	Type of algorithm for which the key can be used X'02' AES X'03' HMAC
42	2	Key type:  For algorithm AES: X'0001' CIPHER X'0003' EXPORTER X'0004' IMPORTER  For algorithm HMAC: X'0002' MAC
44	1	Key-usage field count ( <i>kuf</i> ) - (1 byte)
45	<i>kuf</i> * 2	Key-usage fields ( <i>kuf</i> * 2 bytes) <ul style="list-style-type: none"> <li>For HMAC algorithm keys, refer to Table 355 on page 846.</li> <li>For AES algorithm Key-Encrypting Keys (Exporter or Importer), refer to Table 356 on page 847.</li> <li>For AES algorithm Cipher Keys, refer to Table 357 on page 850.</li> </ul>
45 + <i>kuf</i> * 2	1	Key-management field count ( <i>kmf</i> ): 2 (no pedigree information) or 3 (has pedigree information)
46 + <i>kuf</i> * 2	2	Key-management field 1  High-order byte: 1xxx xxxx Allow export using symmetric key x1xx xxxx Allow export using unauthenticated asymmetric key xx1x xxxx Allow export using authenticated asymmetric key xxx1 xxxx Allow export in RAW format.  All other bits are reserved and must be zero.  Low-order byte: --symmetric-- 1xxx xxxx Prohibit export using DES key. x1xx xxxx Prohibit export using AES key. --asymmetric-- xxxx 1xxx Prohibit export using RSA key.  All other bits are reserved and must be zero.

Table 354. Variable-length Symmetric Key Token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
48 + <i>kuf</i> * 2	2	<p>Key-management field 2</p> <p>High-order byte:</p> <p><b>11xx xxxx</b> Key, if present, is incomplete. Key requires at least 2 more parts.</p> <p><b>10xx xxxx</b> Key, if present, is incomplete. Key requires at least 1 more part.</p> <p><b>01xx xxxx</b> Key, if present, is incomplete. Key can be completed or have more parts added.</p> <p><b>00xx xxxx</b> Key, if present, is complete. No more parts can be added.</p> <p>All other bits are reserved and must be zero.</p> <p>Low-order byte (Security History):</p> <p><b>xxx1 xxxx</b> Key was encrypted with an untrusted KEK</p> <p><b>xxxx 1xxx</b> Key was in a format without type/usage attributes</p> <p><b>xxxx x1xx</b> Key was encrypted with key weaker than itself</p> <p><b>xxxx xx1x</b> Key was in a non-CCA format</p> <p><b>xxxx xxx1</b> Key was encrypted in ECB mode.</p> <p>All other bits are reserved and must be zero.</p>
50 + <i>kuf</i> * 2	2	<p>Key-management field 3 - Pedigree (this field may or may not be present)</p> <p>Indicates how key was originally created and how it got into the current system.</p> <p>High-order byte: Pedigree Original.</p> <p><b>X'00'</b> Unknown (Key Token Build2, Key Translate2)</p> <p><b>X'01'</b> Other - method other than those defined here, probably used in UDX</p> <p><b>X'02'</b> Randomly Generated (Key Generate2)</p> <p><b>X'03'</b> Established by key agreement (ECC Diffie-Hellman)</p> <p><b>X'04'</b> Created from cleartext key components (Key Part Import2)</p> <p><b>X'05'</b> Entered as a cleartext key value (Key Part Import2, Secure Key Import2)</p> <p><b>X'06'</b> Derived from another key</p> <p><b>X'07'</b> Cleartext keys or key parts that were entered at TKE and secured from there to the target card (operational key load)</p> <p>All unused values are reserved and undefined.</p>

Table 354. Variable-length Symmetric Key Token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
		<p>Low-order byte: Pedigree Current.</p> <p>X'00' Unknown (Key Token Build2)</p> <p>X'01' Other - method other than those defined here, probably used in UDX</p> <p>X'02' Randomly Generated (Key Generate2)</p> <p>X'03' Established by key agreement (ECC Diffie-Hellman)</p> <p>X'04' Created from cleartext key components (Key Part Import2)</p> <p>X'05' Entered as a cleartext key value (Key Part Import2, Secure Key Import2)</p> <p>X'06' Derived from another key</p> <p>X'07' Imported from a CCA 05 variable length token with pedigree field (Symmetric Key Import2)</p> <p>X'08' Imported from a CCA 05 variable length token with no pedigree field (Symmetric Key Import2)</p> <p>X'09' Imported from a CCA token that had a CV</p> <p>X'0A' Imported from a CCA token that had no CV or a zero CV</p> <p>X'0B' Imported from a TR-31 key block that contained a CCA CV (ATTR-CV option) (TR-31 Import)</p> <p>X'0C' Imported from a TR-31 key block that did not contain a CCA CV (TR-31 Import)</p> <p>X'0D' Imported using PKCS 1.2 RSA encryption (Symmetric Key Import2)</p> <p>X'0E' Imported using PKCS OAEP encryption (Symmetric Key Import2)</p> <p>X'0F' Imported using PKA92 RSA encryption (Symmetric Key Import2)</p> <p>X'10' Imported using RSA ZERO-PAD encryption (Symmetric Key Import2)</p> <p>X'11' Converted from a CCA token that had a CV (Key Translate2)</p> <p>X'12' Converted from a CCA token that had no CV or a zero CV (Key Translate2)</p> <p>X'13' Cleartext keys or key parts that were entered at TKE and secured from there to the target card (operational key load)</p> <p>X'14' Exported from a CCA 05 variable length token with pedigree field (Symmetric Key Export)</p> <p>X'15' Exported from a CCA 05 variable length token with no pedigree field (Symmetric Key Export)</p> <p>X'16' Exported using PKCS OAEP encryption (Symmetric Key Export)</p> <p>All unused values are reserved and undefined.</p>
46 + $kuf * 2 + kmf * 2$	<i>kl</i>	Key name
46 + $kuf * 2 + kmf * 2 + kl$	<i>iead</i>	IBM extended associated data
46 + $kuf * 2 + kmf * 2 + kl + iead$	<i>uad</i>	Installation-defined associated data



Table 354. Variable-length Symmetric Key Token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
<b>Clear key or encrypted payload</b>		
30 + <i>adl</i>	$(pl+7)/8$	<p><b>Encrypted AESKW payload (internal keys):</b> The encrypted AESKW payload is created from the unencrypted AESKW payload which is made up of the ICV/pad length/hash options and hash length/hash options/hash of the associated data/key material/padding. See unencrypted AESKW payload below.</p> <p><b>Encrypted PKOAEP2 payload (external keys):</b> The encrypted PKOAEP2 payload is created using the PKCS #1 v1.2 encoding method for a given hash algorithm. The message (M) inside the encoding contains: [2 bytes: bit length of key]    [clear HMAC key]. M is encoded using OAEP and then encrypted with an RSA public key according to the standard.</p> <p><b>Clear key payload:</b> When the key is clear, only the key material will be in the payload padded to the nearest byte with binary zeros.</p>
30 + <i>adl</i> + $(pl+7)/8$		End of AESKW components
<b>Unencrypted AESKW payload (This data will never appear in the clear outside of the cryptographic coprocessor)</b>		
0	6	<p>Integrity check value.</p> <p>Six byte constant: X'A6A6A6A6A6A6'.</p>
6	1	Length of the padding in bits: <i>pb</i>
7	1	Length of hash options and hash of the associated data in bytes ( <i>hoh</i> )
8	4	Hash options
12	<i>hoh</i> - 4	Hash of the associated data
8 + <i>hoh</i>	$(pl/8) - 8 - hoh$	Key data and padding (key data is left justified).
<i>pl/8</i>		<i>pl</i> is the bit length of the payload

Table 355. HMAC Algorithm Key-usage fields

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count ( <i>kuf</i> ): 2

Table 355. HMAC Algorithm Key-usage fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p><b>1xxx xxxx</b> Key can be used for generate.</p> <p><b>x1xx xxxx</b> Key can be used for verify.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p><b>xxxx 1xxx</b> The key can only be used in UDXs (used in KGN, KIM, KEX).</p> <p><b>xxxx 0xxx</b> The key can be used in both UDXs and CCA.</p> <p><b>xxxx xuuu</b> Reserved for UDXs, where uuu are UDX-defined bits.</p> <p>All unused bits are reserved and must be zero.</p>
47	2	<p>Key-usage field 2</p> <p>High-order byte:</p> <p><b>1xxx xxxx</b> SHA-1 hash method is allowed for the key.</p> <p><b>x1xx xxxx</b> SHA-224 hash method is allowed for the key.</p> <p><b>xx1x xxxx</b> SHA-256 hash method is allowed for the key.</p> <p><b>xxx1 xxxx</b> SHA-384 hash method is allowed for the key.</p> <p><b>xxxx 1xxx</b> SHA-512 hash method is allowed for the key.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>All bits are reserved and must be zero.</p>

Table 356. AES Algorithm KEK Key-usage fields

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count ( <i>kuf</i> ): 4

Table 356. AES Algorithm KEK Key-usage fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	<p>Key-usage field 1</p> <p>High-order byte for EXPORTER:</p> <p>1xxx xxxx Key can be used for EXPORT.</p> <p>x1xx xxxx Key can be used for TRANSLAT.</p> <p>xx1x xxxx Key can be used for GENERATE-OPEX.</p> <p>xxx1 xxxx Key can be used for GENERATE-IMEX.</p> <p>xxxx 1xxx Key can be used for GENERATE-EXEX.</p> <p>xxxx x1xx Key can be used for GENERATE-PUB.</p> <p>All unused bits are reserved and must be zero.</p> <p>High-order byte for IMPORTER:</p> <p>1xxx xxxx Key can be used for IMPORT.</p> <p>x1xx xxxx Key can be used for TRANSLAT.</p> <p>xx1x xxxx Key can be used for GENERATE-OPIM.</p> <p>xxx1 xxxx Key can be used for GENERATE-IMEX.</p> <p>xxxx 1xxx Key can be used for GENERATE-IMIM.</p> <p>xxxx x1xx Key can be used for GENERATE-PUB.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>xxxx 1xxx The key can only be used in UDXs (used in KGN, KIM, KEX).</p> <p>xxxx 0xxx The key can be used in both UDXs and CCA.</p> <p>xxxx xuuu Reserved for UDXs, where uuu are UDX-defined bits.</p> <p>All unused bits are reserved and must be zero.</p>

Table 356. AES Algorithm KEK Key-usage fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
47	2	<p>Key-usage field 2</p> <p>High-order byte:</p> <p><b>1xxx xxxx</b> Key can wrap a TR-31 key.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p><b>xxxx xxx1</b> This KEK can export a key in RAW format.</p> <p>All unused bits are reserved and must be zero</p>
49	2	<p>Key-usage field 3</p> <p>High-order byte:</p> <p><b>1xxx xxxx</b> Key can wrap DES keys</p> <p><b>x1xx xxxx</b> Key can wrap AES keys</p> <p><b>xx1x xxxx</b> Key can wrap HMAC keys</p> <p><b>xxx1 xxxx</b> Key can wrap RSA keys</p> <p><b>xxxx 1xxx</b> Key can wrap ECC keys</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>All bits are reserved and must be zero.</p>
51	2	<p>Key-usage field 4</p> <p>High-order byte:</p> <p><b>1xxx xxxx</b> Key can wrap DATA class keys</p> <p><b>x1xx xxxx</b> Key can wrap KEK class keys</p> <p><b>xx1x xxxx</b> Key can wrap PIN class keys</p> <p><b>xxx1 xxxx</b> Key can wrap DERIVATION class keys</p> <p><b>xxxx 1xxx</b> Key can wrap CARD class keys</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>All bits are reserved and must be zero.</p>

Table 357. AES Algorithm Cipher Key Associated Data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count ( <i>kuf</i> ): 2
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p><b>1xxx xxxx</b> Key can be used for encryption.</p> <p><b>x1xx xxxx</b> Key can be used for decryption.</p> <p><b>xx1x xxxx</b> Key can be used for cipher text translate only.</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p><b>xxxx 1xxx</b> The key can only be used in UDXs (used in KGN, KIM, KEX).</p> <p><b>xxxx 0xxx</b> The key can be used in both UDXs and CCA.</p> <p><b>xxxx xuuu</b> Reserved for UDXs, where uuu are UDX-defined bits.</p> <p>All unused bits are reserved and must be zero.</p>
47	2	<p>Key-usage field 2</p> <p>High-order byte:</p> <p><b>X'00'</b> Key can be used for Cipher Block Chaining (CBC).</p> <p><b>X'01'</b> Key can be used for Electronic Code Book (ECB).</p> <p><b>X'02'</b> Key can be used for Cipher Feedback (CFB).</p> <p><b>X'03'</b> Key can be used for Output Feedback (OFB).</p> <p><b>X'04'</b> Key can be used for Galois/Counter Mode (GCM)</p> <p><b>X'05'</b> Key can be used for XEX-based Tweaked CodeBook Mode with CipherText Stealing (XTS)</p> <p>All unused values are reserved and must not be used.</p> <p>Low-order byte:</p> <p>All bits are reserved and must be zero.</p>

## Variable-length Symmetric Null Key Token

The following table shows the format for a variable-length symmetric null key token.

Table 358. Variable-length Symmetric Null Token

Bytes	Description
0	X'00' Token identifier (indicates that this is a null key token).
1	Version, X'00'.

Table 358. Variable-length Symmetric Null Token (continued)

Bytes	Description
2-3	X'0008' Length of the key token structure.
4-7	Ignored (zero).

## PKA Key Token Formats

### PKA Null Key Token

Table 359 shows the format for a PKA null key token.

Table 359. Format of PKA Null Key Tokens

Bytes	Description
0	X'00' Token identifier (indicates that this is a null key token).
1	Version, X'00'
2-3	X'0008' Length of the key token structure.
4-7	Ignored (should be zero).

## RSA Key Token Formats

### RSA Public Key Token

An RSA public key token contains the following sections:

- A required token header, starting with the token identifier X'1E'
- A required RSA public key section, starting with the section identifier X'04'

Table 360 presents the format of an RSA public key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format).

Table 360. RSA Public Key Token

Offset (Dec)	Number of Bytes	Description
<b>Token Header (required)</b>		
000	001	Token identifier. X'1E' indicates an external token.
001	001	Version, X'00'.
002	002	Length of the key token structure.
004	004	Ignored. Should be zero.
<b>RSA Public Key Section (required)</b>		
000	001	X'04', section identifier, RSA public key.
001	001	X'00', version.
002	002	Section length, 12+xxx+yyy.
004	002	Reserved field.
006	002	RSA public key exponent field length in bytes, "xxx".
008	002	Public key modulus length in bits.
010	002	RSA public key modulus field length in bytes, "yyy".
012	xxx	Public key exponent (this is generally a 1-, 3-, or 64- to 512-byte quantity), e. e must be odd and $1 < e < n$ . (Frequently, the value of e is $2^{16}+1$ )

Table 360. RSA Public Key Token (continued)

Offset (Dec)	Number of Bytes	Description
12+xxx	yyy	Modulus, n.

## RSA Private External Key Token

An RSA private external key token contains the following sections:

- A required PKA token header starting with the token identifier X'1E'
- A required RSA private key section starting with one of the following section identifiers:
  - X'02' which indicates a modulus-exponent form RSA private key section (not optimized) with modulus length of up to 1024 bits for use with the Cryptographic Coprocessor Feature or the PCI Cryptographic Coprocessor.
  - X'08' which indicates an optimized Chinese Remainder Theorem form private key section with modulus bit length of up to 4096 bits for use with the PCICC, PCIXCC, or CCA Crypto Express coprocessor..
  - X'09' which indicates a modulus-exponent form RSA private key section (not optimized) with modulus length of up to 4096 bits for use with the CCA Crypto Express coprocessor.
- A required RSA public key section, starting with the section identifier X'04'
- An optional private key name section, starting with the section identifier X'10'

Table 361 presents the basic record format of an RSA private external key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format). All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

Table 361. RSA Private External Key Token Basic Record Format

Offset (Dec)	Number of Bytes	Description
<b>Token Header (required)</b>		
000	001	Token identifier. X'1E' indicates an external token. The private key is either in cleartext or enciphered with a transport key-encrypting key.
001	001	Version, X'00'.
002	002	Length of the key token structure.
004	004	Ignored. Should be zero.
<b>RSA Private Key Section (required)</b>		
<ul style="list-style-type: none"> <li>• For 1024-bit Modulus-Exponent form refer to “RSA Private Key Token, 1024-bit Modulus-Exponent External Form” on page 853</li> <li>• For 4096-bit Modulus-Exponent form refer to “RSA private key token, 4096-bit Modulus-Exponent external form” on page 854</li> <li>• For 4096-bit Chinese Remainder Theorem form refer to “RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Form” on page 858</li> </ul>		
<b>RSA Public Key Section (required)</b>		
000	001	X'04', section identifier, RSA public key.
001	001	X'00', version.
002	002	Section length, 12+xxx.
004	002	Reserved field.
006	002	RSA public key exponent field length in bytes, “xxx”.
008	002	Public key modulus length in bits.

Table 361. RSA Private External Key Token Basic Record Format (continued)

Offset (Dec)	Number of Bytes	Description
010	002	RSA public key modulus field length in bytes, which is zero for a private token. <b>Note:</b> In an RSA private key token, this field should be zero. The RSA private key section contains the modulus.
012	xxx	Public key exponent, e (this is generally a 1-, 3-, or 64- to 512-byte quantity). e must be odd and $1 < e < n$ . (Frequently, the value of e is $2^{16}+1$ (=65,537).
<b>Private Key Name (optional)</b>		
000	001	X'10', section identifier, private key name.
001	001	X'00', version.
002	002	Section length, X'0044' (68 decimal).
004	064	Private key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.

**RSA Private Key Token, 1024-bit Modulus-Exponent External Form:** This RSA private key token and the external X'02' token is supported on the Cryptographic Coprocessor Feature and PCI Cryptographic Coprocessor.

Table 362. RSA Private Key Token, 1024-bit Modulus-Exponent External Format

Offset (Dec)	Number of Bytes	Description								
000	001	X'02', section identifier, RSA private key, modulus-exponent format (RSA-PRIV)								
001	001	X'00', version.								
002	002	Length of the RSA private key section X'016C' (364 decimal).								
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.								
024	004	Reserved; set to binary zero.								
028	001	Key format and security: X'00'     Unencrypted RSA private key subsection identifier. X'82'     Encrypted RSA private key subsection identifier.								
029	001	Reserved, binary zero.								
030	020	SHA-1 hash of the optional key-name section. If there is no key-name section, then 20 bytes of X'00'.								
050	004	Key use flag bits.  <table><tr><td>Bit</td><td>Meaning When Set On</td></tr><tr><td>0</td><td>Key management usage permitted.</td></tr><tr><td>1</td><td>Signature usage not permitted.</td></tr><tr><td>6</td><td>The key is translatable.</td></tr></table> All other bits reserved, set to binary zero.	Bit	Meaning When Set On	0	Key management usage permitted.	1	Signature usage not permitted.	6	The key is translatable.
Bit	Meaning When Set On									
0	Key management usage permitted.									
1	Signature usage not permitted.									
6	The key is translatable.									
054	006	Reserved; set to binary zero.								
060	024	Reserved; set to binary zero.								
084	Start of the optionally-encrypted secure subsection.									
084	024	Random number, confounder.								



Table 362. RSA Private Key Token, 1024-bit Modulus-Exponent External Format (continued)

Offset (Dec)	Number of Bytes	Description
108	128	Private-key exponent, d. $d = e^{-1} \bmod((p-1)(q-1))$ , and $1 < d < n$ where e is the public exponent.
		End of the optionally-encrypted subsection; the confounder field and the private-key exponent field are enciphered for key confidentiality when the key format and security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the ede2 algorithm.
236	128	Modulus, n. $n = pq$ where p and q are prime and $1 < n < 2^{1024}$ .

**RSA private key token, 4096-bit Modulus-Exponent external form:** This RSA private key token and the external X'09' token is supported on a CCA Crypto Express coprocessor.

Table 363. RSA Private Key Token, 4096-bit Modulus-Exponent External Format

Offset (Dec)	Number of Bytes	Description								
000	001	X'09', section identifier, RSA private key, modulus-exponent format (RSAMEVAR).								
001	001	X'00', version.								
002	002	Length of the RSA private key section 132+ddd+nnn+xxx.								
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.								
024	002	Length of the encrypted private key section 8+ddd+xxx.								
026	002	Reserved; set to binary zero.								
028	001	Key format and security: X'00'    Unencrypted RSA private key subsection identifier. X'82'    Encrypted RSA private key subsection identifier.								
029	001	Reserved, set to binary zero.								
030	020	SHA-1 hash of the optional key-name section. If there is no key-name section, then 20 bytes of X'00'.								
050	001	Key use flag bits. <table><tr><th>Bit</th><th>Meaning When Set On</th></tr><tr><td>0</td><td>Key management usage permitted.</td></tr><tr><td>1</td><td>Signature usage not permitted.</td></tr><tr><td>6</td><td>The key is translatable</td></tr></table> All other bits reserved, set to binary zero.	Bit	Meaning When Set On	0	Key management usage permitted.	1	Signature usage not permitted.	6	The key is translatable
Bit	Meaning When Set On									
0	Key management usage permitted.									
1	Signature usage not permitted.									
6	The key is translatable									
051	001	Reserved; set to binary zero.								
052	048	Reserved; set to binary zero.								
100	016	Reserved; set to binary zero.								
116	002	Length of private exponent, d, in bytes: ddd.								
118	002	Length of modulus, n, in bytes: nnn.								
120	002	Length of padding field, in bytes: xxx.								
122	002	Reserved; set to binary zero.								
124	Start of the optionally-encrypted secure subsection.									
124	008	Random number, confounder.								

Table 363. RSA Private Key Token, 4096-bit Modulus-Exponent External Format (continued)

Offset (Dec)	Number of Bytes	Description
132	ddd	Private-key exponent, d. $d = e^{-1} \bmod((p-1)(q-1))$ , and $1 < d < n$ where e is the public exponent.
132+ddd	xxx	X'00' padding of length xxx bytes such that the length from the start of the random number above to the end of the padding field is a multiple of eight bytes.
	End of the optionally-encrypted subsection; the confounder field and the private-key exponent field are enciphered for key confidentiality when the key format and security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the ede2 algorithm.	
132+ddd+xxx	nnn	Modulus, n. $n = pq$ where p and q are prime and $1 < n < 2^{4096}$ .

**RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section external form:** This RSA private key token is supported on the Crypto Express3 Coprocessor and Crypto Express4 Coprocessor.

Table 364. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external form

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'30' RSA private key, ME format with AES encrypted OPK.
001	001	Section version number (X'00').
002	002	Section length: 122 + nnn + ppp
004	002	Length of "Associated Data" section
006	002	Length of payload data: ppp
008	002	Reserved, binary zero.
		<b>Start of Associated Data</b>
010	001	Associated Data Version: X'02' Version 2
011	001	Key format and security flag: X'00' Unencrypted ME RSA private-key subsection identifier X'82' Encrypted ME RSA private-key subsection identifier
012	001	Key source flag: Reserved, binary zero.
013	001	Reserved, binary zeroes.
014	001	Hash type: X'00' Clear key X'02' SHA-256
015	032	SHA-256 hash of all optional sections that follow the public key section, if any; else 32 bytes of X'00'.
047	003	Reserved, binary zero.

Table 364. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external form (continued)

Offset (bytes)	Length (bytes)	Description
050	001	Key-usage flag:  <b>B'11xx xxxx'</b> Only key unwrapping (KM-ONLY)  <b>B'10xx xxxx'</b> Both signature generation and key unwrapping (KEY-MGMT)  <b>B'01xx xxxx'</b> Undefined  <b>B'00xx xxxx'</b> Only signature generation (SIG-ONLY)  Translation control:  <b>B'xxxx xx1x'</b> Private key translation is allowed (XLATE-OK)  <b>B'xxxx xx0x'</b> Private key translation is not allowed (NO-XLATE)
051	001	Reserved, binary zero.
052	002	Length of modulus: nnn bytes
054	002	Length of private exponent: ddd bytes
		<b>End of Associated Data</b>
056	048	16 byte confounder + 32-byte Object Protection Key.  OPK used as an AES key.  encrypted with an AES KEK.
104	016	Key verification pattern <ul style="list-style-type: none"> <li>• For an encrypted private key, KEK verification pattern (KVP)</li> <li>• For a clear private key, binary zeros</li> <li>• For a skeleton, binary zeros</li> </ul>
120	002	Reserved, binary zeros.
122	nnn	Modulus
122+nnn	ppp	Payload starts here and includes:  When this section is unencrypted: <ul style="list-style-type: none"> <li>• Clear private exponent d.</li> <li>• Length ppp bytes : ddd + 0</li> </ul> When this section is encrypted: <ul style="list-style-type: none"> <li>• Private exponent d within the AESKW-wrapped payload.</li> <li>• Length ppp bytes : ddd + AESKW format overhead</li> </ul>

**RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section external form:** This RSA private key token is supported on the Crypto Express3 Coprocessor and Crypto Express4 Coprocessor.

Table 365. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier:  <b>X'31'</b> RSA private key, CRT format with AES encrypted OPK
001	001	Section version number (X'00').

Table 365. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form (continued)

Offset (bytes)	Length (bytes)	Description
002	002	Section length: 134 + nnn + xxx
004	002	Length of "Associated Data" section
006	002	Length of payload data: xxx
008	002	Reserved, binary zero.
		<b>Start of Associated Data</b>
010	001	Associated Data Version: X'03'      Version 3
011	001	Key format and security flag: X'40'      Unencrypted RSA private-key subsection identifier X'42'      Encrypted RSA private-key subsection identifier
012	001	Key source flag: Reserved, binary zero.
013	001	Reserved, binary zeroes.
014	001	Hash type: X'00'      Clear key X'01'      SHA-256
015	032	SHA-256 hash of all optional sections that follow the public key section, if any; else 32 bytes of X'00'.
047	003	Reserved, binary zero.
050	001	Key-usage flag: B'11xx xxxx'      Only key unwrapping (KM-ONLY) B'10xx xxxx'      Both signature generation and key unwrapping (KEY-MGMT) B'01xx xxxx'      Undefined B'00xx xxxx'      Only signature generation (SIG-ONLY) Translation control: B'xxxx xx1x'      Private key translation is allowed (XLATE-OK) B'xxxx xx0x'      Private key translation is not allowed (NO-XLATE)
051	001	Reserved, binary zero.
052	002	Length of the prime number, p, in bytes: ppp.
054	002	Length of the prime number, q, in bytes: qq
056	002	Length of dp : rrr.
058	002	Length of dq : sss.
060	002	Length of U: uu.
062	002	Length of modulus, nnn.
064	002	Reserved, binary zero.
066	002	Reserved, binary zero.
		<b>End of Associated Data</b>

Table 365. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form (continued)

Offset (bytes)	Length (bytes)	Description
068	048	16 byte confounder + 32-byte Object Protection Key.  OPK used as an AES key.  External tokens:  encrypted with an AES KEK.  Internal tokens:  encrypted with the APKA master key.
116	016	Key verification pattern <ul style="list-style-type: none"> <li>For an encrypted private key, KEK verification pattern (KVP)</li> <li>For a clear private key, binary zeros</li> <li>For a skeleton, binary zeros</li> </ul>
132	002	Reserved, binary zeros
134	nnn	Modulus, n, n=pq, where p and q are prime.
134+nnn	xxx	Payload starts here and includes:  When this section is unencrypted: <ul style="list-style-type: none"> <li>Clear prime number p</li> <li>Clear prime number q</li> <li>Clear dp</li> <li>Clear dq</li> <li>Clear U</li> <li>Length xxx bytes: ppp + qqg + rrr + sss +uuu + 0</li> </ul> When this section is encrypted: <ul style="list-style-type: none"> <li>prime number p</li> <li>prime number q</li> <li>dp</li> <li>dq</li> <li>U</li> <li>within the AESKW-wrapped payload.</li> </ul> Length xxx bytes : ppp + qqg + rrr + sss +uuu + AESKW format overhead

#### RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Form:

This RSA private key token (up to 2048-bit modulus) is supported on the PCICC, PCIXCC, and CCA Crypto Express coprocessor. The 4096-bit modulus private key token is supported on the z9 EC, z9 BC, z10 EC and z10 BC with the Nov. 2007 or later version of the licensed internal code installed on the CCA Crypto Express coprocessor.

Table 366. RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Format

Offset (Dec)	Number of Bytes	Description
000	001	X'08', section identifier, RSA private key, CRT format (RSA-CRT)
001	001	X'00', version.
002	002	Length of the RSA private-key section, 132 + ppp + qqg + rrr + sss + uuu + xxx + nnn.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the end of the modulus.

Table 366. RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Format (continued)

Offset (Dec)	Number of Bytes	Description								
024	004	Reserved; set to binary zero.								
028	001	Key format and security: <b>X'40'</b> Unencrypted RSA private-key subsection identifier, Chinese Remainder form. <b>X'42'</b> Encrypted RSA private-key subsection identifier, Chinese Remainder form.								
029	001	Reserved; set to binary zero.								
030	020	SHA-1 hash of the optional key-name section and any following optional sections. If there are no optional sections, then 20 bytes of X'00'.								
050	004	Key use flag bits. <table><tr><th>Bit</th><th>Meaning When Set On</th></tr><tr><td>0</td><td>Key management usage permitted.</td></tr><tr><td>1</td><td>Signature usage not permitted.</td></tr><tr><td>6</td><td>The key is translatable.</td></tr></table> All other bits reserved, set to binary zero.	Bit	Meaning When Set On	0	Key management usage permitted.	1	Signature usage not permitted.	6	The key is translatable.
Bit	Meaning When Set On									
0	Key management usage permitted.									
1	Signature usage not permitted.									
6	The key is translatable.									
054	002	Length of prime number, p, in bytes: ppp.								
056	002	Length of prime number, q, in bytes: qq.								
058	002	Length of d <sub>p</sub> , in bytes: rrr.								
060	002	Length of d <sub>q</sub> , in bytes: sss.								
062	002	Length of U, in bytes: uuu.								
064	002	Length of modulus, n, in bytes: nnn.								
066	004	Reserved; set to binary zero.								
070	002	Length of padding field, in bytes: xxx.								
072	004	Reserved, set to binary zero.								
076	016	Reserved, set to binary zero.								
092	032	Reserved; set to binary zero.								
124	Start of the optionally-encrypted secure subsection.									
124	008	Random number, confounder.								
132	ppp	Prime number, p.								
132 + ppp	qqq	Prime number, q								
132 + ppp + qq	rrr	d <sub>p</sub> = d mod(p - 1)								
132 + ppp + qq + rrr	sss	d <sub>q</sub> = d mod(q - 1)								
132 + ppp + qq + rrr + sss	uuu	U = q <sup>-1</sup> mod(p).								
132 + ppp + qq + rrr + sss + uuu	xxx	X'00' padding of length xxx bytes such that the length from the start of the random number above to the end of the padding field is a multiple of eight bytes.								
	End of the optionally-encrypted secure subsection; all of the fields starting with the confounder field and ending with the variable length pad field are enciphered for key confidentiality when the key format-and-security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the TDES (CBC outer chaining) algorithm.									

Table 366. RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Format (continued)

Offset (Dec)	Number of Bytes	Description
132 + ppp + qqg + rrr + sss + uuu + xxx	nnn	Modulus, n. $n = pq$ where p and q are prime and $1 < n < 2^{4096}$ .

### RSA Private Internal Key Token

An RSA private internal key token contains the following sections:

- A required PKA token header, starting with the token identifier X'1F'
- basic record format of an RSA private internal key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format). All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

Table 367. RSA Private Internal Key Token Basic Record Format

Offset (Dec)	Number of Bytes	Description
<b>Token Header (required)</b>		
000	001	Token identifier. X'1F' indicates an internal token. The private key is enciphered with a PKA master key.
001	001	Version, X'00'.
002	002	Length of the key token structure excluding the internal information section.
004	004	Ignored; should be zero.
<b>RSA Private Key Section and Secured Subsection (required)</b>		
<ul style="list-style-type: none"> <li>• For 1024-bit X'02' Modulus-Exponent form refer to "RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for Cryptographic Coprocessor Feature" on page 861</li> <li>• For 1024-bit X'06' Modulus-Exponent form refer to "RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for PCICC, PCIXCC, or CCA Crypto Express coprocessor" on page 862</li> <li>• For 4096-bit X'08' Chinese Remainder Theorem form refer to "RSA Private Key Token, 4096-bit Chinese Remainder Theorem Internal Form" on page 867</li> </ul>		
<b>RSA Public Key Section (required)</b>		
000	001	X'04', section identifier, RSA public key.
001	001	X'00', version.
002	002	Section length, 12+xxx.
004	002	Reserved field.
006	002	RSA public key exponent field length in bytes, "xxx".
008	002	Public key modulus length in bits.
010	002	RSA public key modulus field length in bytes, which is zero for a private token.
012	xxx	Public key exponent (this is generally a 1, 3, or 64 to 512 byte quantity), e. e must be odd and $1 < e < n$ . (Frequently, the value of e is $2^{16}+1$ (=65,537).)
<b>Private Key Name (optional)</b>		
000	001	X'10', section identifier, private key name.
001	001	X'00', version.
002	002	Section length, X'0044' (68 decimal).

Table 367. RSA Private Internal Key Token Basic Record Format (continued)

Offset (Dec)	Number of Bytes	Description																		
004	064	Private key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.																		
Internal Information Section (required)																				
000	004	Eye catcher 'PKTN'.																		
004	004	PKA token type. <table><tr><th>Bit</th><th>Meaning When Set On</th></tr><tr><td>0</td><td>RSA key.</td></tr><tr><td>1</td><td>DSS key.</td></tr><tr><td>2</td><td>Private key.</td></tr><tr><td>3</td><td>Public key.</td></tr><tr><td>4</td><td>Private key name section exists.</td></tr><tr><td>5</td><td>Private key unenciphered.</td></tr><tr><td>6</td><td>Blinding information present.</td></tr><tr><td>7</td><td>Retained private key.</td></tr></table>	Bit	Meaning When Set On	0	RSA key.	1	DSS key.	2	Private key.	3	Public key.	4	Private key name section exists.	5	Private key unenciphered.	6	Blinding information present.	7	Retained private key.
Bit	Meaning When Set On																			
0	RSA key.																			
1	DSS key.																			
2	Private key.																			
3	Public key.																			
4	Private key name section exists.																			
5	Private key unenciphered.																			
6	Blinding information present.																			
7	Retained private key.																			
008	004	Address of token header.																		
012	002	Total length of total structure including this information section.																		
014	002	Count of number of sections.																		
016	016	PKA master key hash pattern.																		
032	001	Domain of retained key.																		
033	008	Serial number of processor holding retained key.																		
041	007	Reserved.																		

**RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for Cryptographic Coprocessor Feature:**

Table 368. RSA Private Internal Key Token, 1024-bit ME Form for Cryptographic Coprocessor Feature

Offset (Dec)	Number of Bytes	Description
000	001	X'02', section identifier, RSA private key.
001	001	X'00', version.
002	002	Length of the RSA private key section X'016C' (364 decimal).
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.
024	004	Reserved; set to binary zero.
028	001	Key format and security: X'02' RSA private key.
029	001	Format of external key from which this token was derived: X'21' External private key was specified in the clear. X'22' External private key was encrypted.
030	020	SHA-1 hash of the key token structure contents that follow the public key section. If no sections follow, this field is set to binary zeros.



Table 368. RSA Private Internal Key Token, 1024-bit ME Form for Cryptographic Coprocessor Feature (continued)

Offset (Dec)	Number of Bytes	Description
050	001	Key use flag bits.  <div> <div>Bit</div> <div>Meaning When Set On</div> </div> <div> <div>0</div> <div>Key management usage permitted.</div> </div> <div> <div>1</div> <div>Signature usage not permitted.</div> </div> All other bits reserved, set to binary zero.
051	009	Reserved; set to binary zero.
060	048	Object Protection Key (OPK) encrypted under a PKA master key—can be under the Signature Master Key (SMK) or Key Management Master Key (KMMK) depending on key use.
108	128	Secret key exponent d, encrypted under the OPK. $d=e^{-1} \bmod((p-1)(q-1))$
236	128	Modulus, n. $n=pq$ where p and q are prime and $1 < n < 2^{1024}$ .

**RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for PCICC, PCIXCC, or CCA Crypto Express coprocessor:**

Table 369. RSA Private Internal Key Token, 1024-bit ME Form for PCICC, PCIXCC, or CCA Crypto Express coprocessor

Offset (Dec)	Number of Bytes	Description
000	001	X'06', section identifier, RSA private key modulus-exponent format (RSA-PRIV).
001	001	X'00', version.
002	002	Length of the RSA private key section X'0198' (408 decimal) + rrr + iii + xxx.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to and including the modulus at offset 236.
024	004	Reserved; set to binary zero.
028	001	Key format and security: X'02' RSA private key.
029	001	Format of external key from which this token was derived: X'21' External private key was specified in the clear. X'22' External private key was encrypted. X'23' Private key was generated using regeneration data. X'24' Private key was randomly generated.
030	020	SHA-1 hash of the optional key-name section and any following optional sections. If there are no optional sections, this field is set to binary zeros.
050	004	Key use flag bits.  <div> <div>Bit</div> <div>Meaning When Set On</div> </div> <div> <div>0</div> <div>Key management usage permitted.</div> </div> <div> <div>1</div> <div>Signature usage not permitted.</div> </div> All other bits reserved, set to binary zeros.
054	006	Reserved; set to binary zero.
060	048	Object Protection Key (OPK) encrypted under the Asymmetric Keys Master Key using the ede3 algorithm.

Table 369. RSA Private Internal Key Token, 1024-bit ME Form for PCICC, PCIXCC, or CCA Crypto Express coprocessor (continued)

Offset (Dec)	Number of Bytes	Description
108	128	Private key exponent d, encrypted under the OPK using the ede5 algorithm. $d=e^{-1} \bmod ((p-1)(q-1))$ , and $1 < d < n$ where e is the public exponent.
236	128	Modulus, n. $n=pq$ where p and q are prime and $2^{512} < n < 2^{1024}$ .
364	016	Asymmetric-Keys Master Key hash pattern.
380	020	SHA-1 hash value of the blinding information subsection cleartext, offset 400 to the end of the section.
400	002	Length of the random number r, in bytes: rrr.
402	002	Length of the random number $r^{-1}$ , in bytes: iii.
404	002	Length of the padding field, in bytes: xxx.
406	002	Reserved; set to binary zeros.
408	Start of the encrypted blinding subsection	
408	rrr	Random number r (used in blinding).
408 + rrr	iii	Random number $r^{-1}$ (used in blinding).
408 + rrr + iii	xxx	X'00' padding of length xxx bytes such that the length from the start of the encrypted blinding subsection to the end of the padding field is a multiple of eight bytes.
End of the encrypted blinding subsection; all of the fields starting with the random number r and ending with the variable length pad field are encrypted under the OPK using TDES (CBC outer chaining) algorithm.		

**RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section internal form:** This RSA private key token is supported on the Crypto Express3 Coprocessor and Crypto Express4 Coprocessor.

Table 370. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'30' RSA private key, ME format with AES encrypted OPK.
001	001	Section version number (X'00').
002	002	Section length: 122 + nnn + ppp
004	002	Length of "Associated Data" section
006	002	Length of payload data: ppp
008	002	Reserved, binary zero.
		<b>Start of Associated Data</b>
010	001	Associated Data Version: X'02' Version 2
011	001	Key format and security flag: X'02' Encrypted ME RSA private-key subsection identifier

Table 370. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form (continued)

Offset (bytes)	Length (bytes)	Description
012	001	Key source flag:  Internal tokens:  X'21' Imported from cleartext X'22' Imported from ciphertext X'23' Generated using regeneration data X'24' Randomly generated
013	001	Reserved, binary zeroes.
014	001	Hash type:  X'00' Clear key X'02' SHA-256
015	032	SHA-256 hash of all optional sections that follow the public key section, if any; else 32 bytes of X'00'.
047	003	Reserved, binary zero.
050	001	Key-usage flag:  B'11xx xxxx' Only key unwrapping (KM-ONLY)  B'10xx xxxx' Both signature generation and key unwrapping (KEY-MGMT)  B'01xx xxxx' Undefined  B'00xx xxxx' Only signature generation (SIG-ONLY)  Translation control:  B'xxxx xx1x' Private key translation is allowed (XLATE-OK)  B'xxxx xx0x' Private key translation is not allowed (NO-XLATE)
051	001	Reserved, binary zero.
052	002	Length of modulus: nnn bytes
054	002	Length of private exponent: ddd bytes
		<b>End of Associated Data</b>
056	048	16 byte confounder + 32-byte Object Protection Key.  OPK used as an AES key.  encrypted with the ECC master key.
104	016	Key verification pattern • For an encrypted private key, ECC master-key verification pattern (MKVP) • For a skeleton, binary zeros
120	002	Reserved, binary zeros.
122	nnn	Modulus

Table 370. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') internal form (continued)

Offset (bytes)	Length (bytes)	Description
122+nnn	ppp	Payload starts here and includes:  When this section is unencrypted: <ul style="list-style-type: none"> <li>• Clear private exponent d.</li> <li>• Length ppp bytes : ddd + 0</li> </ul> When this section is encrypted: <ul style="list-style-type: none"> <li>• Private exponent d within the AESKW-wrapped payload.</li> <li>• Length ppp bytes : ddd + AESKW format overhead</li> </ul>

**RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section internal form:** This RSA private key token is supported on the Crypto Express3 Coprocessor and Crypto Express4 Coprocessor.

RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form

Table 371. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier:  X'31' RSA private key, CRT format with AES encrypted OPK
001	001	Section version number (X'00').
002	002	Section length: 134 + nnn + xxx
004	002	Length of "Associated Data" section
006	002	Length of payload data: xxx
008	002	Reserved, binary zero.
		<b>Start of Associated Data</b>
010	001	Associated Data Version:  X'03' Version 3
011	001	Key format and security flag:  X'08' Unencrypted RSA private-key subsection identifier
012	001	Key source flag:  X'21' Imported from cleartext X'22' Imported from ciphertext X'23' Generated using regeneration data X'24' Randomly generated
013	001	Reserved, binary zeroes.
014	001	Hash type:  X'00' Clear key X'01' SHA-256
015	032	SHA-256 hash of all optional sections that follow the public key section, if any; else 32 bytes of X'00'.
047	003	Reserved, binary zero.

Table 371. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form (continued)

Offset (bytes)	Length (bytes)	Description
050	001	Key-usage flag:  <b>B'11xx xxxx'</b> Only key unwrapping (KM-ONLY)  <b>B'10xx xxxx'</b> Both signature generation and key unwrapping (KEY-MGMT)  <b>B'01xx xxxx'</b> Undefined  <b>B'00xx xxxx'</b> Only signature generation (SIG-ONLY)  Translation control:  <b>B'xxxx xx1x'</b> Private key translation is allowed (XLATE-OK)  <b>B'xxxx xx0x'</b> Private key translation is not allowed (NO-XLATE)
051	001	Reserved, binary zero.
052	002	Length of the prime number, p, in bytes: ppp.
054	002	Length of the prime number, q, in bytes: qq
056	002	Length of dp : rrr.
058	002	Length of dq : sss.
060	002	Length of U: uuu.
062	002	Length of modulus, nnn.
064	002	Reserved, binary zero.
066	002	Reserved, binary zero.
		<b>End of Associated Data</b>
068	048	16 byte confounder + 32-byte Object Protection Key.  OPK used as an AES key.  encrypted with the APKA master key.
116	016	Key verification pattern <ul style="list-style-type: none"> <li>• For an encrypted private key, APKA master-key verification pattern (MKVP)</li> <li>• For a skeleton, binary zeros</li> </ul>
132	002	Reserved, binary zeros
134	nnn	Modulus, n, n=pq, where p and q are prime.

Table 371. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form (continued)

Offset (bytes)	Length (bytes)	Description
134+nnn	xxx	<p>Payload starts here and includes:</p> <p>When this section is unencrypted:</p> <ul style="list-style-type: none"> <li>• Clear prime number p</li> <li>• Clear prime number q</li> <li>• Clear dp</li> <li>• Clear dq</li> <li>• Clear U</li> <li>• Length xxx bytes: ppp + qqg + rrr + sss +uuu + 0</li> </ul> <p>When this section is encrypted:</p> <ul style="list-style-type: none"> <li>• prime number p</li> <li>• prime number q</li> <li>• dp</li> <li>• dq</li> <li>• U</li> <li>• within the AESKW-wrapped payload.</li> </ul> <p>Length xxx bytes : ppp + qqg + rrr + sss +uuu + AESKW format overhead</p>

**RSA Private Key Token, 4096-bit Chinese Remainder Theorem Internal Form:**

This RSA private key token (up to 2048-bit modulus) is supported on the PCICC, PCIXCC, or CCA Crypto Express coprocessor. The 4096-bit modulus private key token is supported on the z9 EC, z9 BC, z10 EC, z10 BC, or z196 with the Nov. 2007 or later version of the licensed internal code installed on the CCA Crypto Express coprocessor.

Table 372. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format

Offset (Dec)	Number of Bytes	Description
000	001	X'08', section identifier, RSA private key, CRT format (RSA-CRT)
001	001	X'00', version.
002	002	Length of the RSA private-key section, 132 + ppp + qqg + rrr + sss + uuu + ttt + iii + xxx + nnn.
004	020	SHA-1 hash value of the private-key subsection cleartext, offset 28 to the end of the modulus.
024	004	Reserved; set to binary zero.
028	001	Key format and security: X'08' Encrypted RSA private-key subsection identifier, Chinese Remainder form.
029	001	Key derivation method: X'21' External private key was specified in the clear. X'22' External private key was encrypted. X'23' Private key was generated using regeneration data. X'24' Private key was randomly generated.
030	020	SHA-1 hash of the optional key-name section and any following sections. If there are no optional sections, then 20 bytes of X'00'.

Table 372. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format (continued)

Offset (Dec)	Number of Bytes	Description
050	004	Key use flag bits:  <div> <div>Bit</div> <div>Meaning When Set On</div> </div> <div> <div>0</div> <div>Key management usage permitted.</div> </div> <div> <div>1</div> <div>Signature usage not permitted.</div> </div> All other bits reserved, set to binary zero.
054	002	Length of prime number, p, in bytes: ppp.
056	002	Length of prime number, q, in bytes: qq.
058	002	Length of $d_p$ , in bytes: rrr.
060	002	Length of $d_q$ , in bytes: sss.
062	002	Length of U, in bytes: uuu.
064	002	Length of modulus, n, in bytes: nnn.
066	002	Length of the random number r, in bytes: ttt.
068	002	Length of the random number $r^{-1}$ , in bytes: iii.
070	002	Length of padding field, in bytes: xxx.
072	004	Reserved, set to binary zero.
076	016	Asymmetric-Keys Master Key hash pattern.
092	032	Object Protection Key (OPK) encrypted under the Asymmetric-Keys Master Key using the TDES (CBC outer chaining) algorithm.
124	Start of the encrypted secure subsection, encrypted under the OPK using TDES (CBC outer chaining).	
124	008	Random number, confounder.
132	ppp	Prime number, p.
132 + ppp	qqq	Prime number, q
132 + ppp + qq	rrr	$d_p = d \bmod(p - 1)$
132 + ppp + qq + rrr	sss	$d_q = d \bmod(q - 1)$
132 + ppp + qq + rrr + sss	uuu	$U = q^{-1} \bmod(p)$ .
132 + ppp + qq + rrr + sss + uuu	ttt	Random number r (used in blinding).
132 + ppp + qq + rrr + sss + uuu + ttt	iii	Random number $r^{-1}$ (used in blinding).
132 + ppp + qq + rrr + sss + uuu + ttt + iii	xxx	X'00' padding of length xxx bytes such that the length from the start of the confounder at offset 124 to the end of the padding field is a multiple of eight bytes.
	End of the encrypted secure subsection; all of the fields starting with the confounder field and ending with the variable length pad field are encrypted under the OPK using TDES (CBC outer chaining) for key confidentiality.	
132 + ppp + qq + rrr + sss + uuu + ttt + iii + xxx	nnn	Modulus, n. $n = pq$ where p and q are prime and $1 < n < 2^{4096}$ .

## DSS Key Token Formats

### DSS Public Key Token

A DSS public key token contains the following sections:

- A required token header, starting with the token identifier X'1E'
- A required DSS public key section, starting with the section identifier X'03'

Table 373 presents the format of a DSS public key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format).

Table 373. DSS Public Key Token

Offset (Dec)	Number of Bytes	Description
<b>Token Header (required)</b>		
000	001	Token identifier. X'1E' indicates an external token.
001	001	Version, X'00'.
002	002	Length of the key token structure.
004	004	Ignored. Should be zero.
<b>DSS Public Key Section (required)</b>		
000	001	X'03', section identifier, DSS public key.
001	001	X'00', version.
002	002	Section length, 14+ppp+qqq+ggg+yyy.
004	002	Size of p in bits. The size of p must be one of: 512, 576, 640, 704, 768, 832, 896, 960, or 1024.
006	002	Size of the p field in bytes, "ppp".
008	002	Size of the q field in bytes, "qqq".
010	002	Size of the g field in bytes, "ggg".
012	002	Size of the y field in bytes, "yyy".
014	ppp	Prime modulus (large public modulus), p.
014 +ppp	qqq	Prime divisor (small public modulus), q. $2^{159} < q < 2^{160}$ .
014 +ppp +qqq	ggg	Public key generator, g.
014 +ppp +qqq +ggg	yyy	Public key, y. $y = g^x \text{ mod}(p)$ ; $1 < y < p$ .

### DSS Private External Key Token

A DSS private external key token contains the following sections:

- A required PKA token header, starting with the token identifier X'1E'
- A required DSS private key section, starting with the section identifier X'01'
- A required DSS public key section, starting with the section identifier X'03'
- An optional private key name section, starting with the section identifier X'10'

Table 374 on page 870 presents the format of a DSS private external key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format). All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.



Table 374. DSS Private External Key Token

Offset (Dec)	Number of Bytes	Description
<i>Token Header (required)</i>		
000	001	Token identifier. X'1E' indicates an external token. The private key is enciphered with a PKA master key.
001	001	Version, X'00'.
002	002	Length of the key token structure.
004	004	Ignored. Should be zero.
<i>DSS Private Key Section and Secured Subsection (required)</i>		
000	001	X'01', section identifier, DSS private key.
001	001	X'00', version.
002	002	Length of the DSS private key section, 436, X'01B4'.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.
024	004	Reserved; set to binary zero.
028	001	Key security: X'00' Unencrypted DSS private key subsection identifier. X'81' Encrypted DSS private key subsection identifier.
029	001	Padding, X'00'.
030	020	SHA-1 hash of the key token structure contents that follow the public key section. If no sections follow, this field is set to binary zeros.
050	010	Reserved; set to binary zero.
060	048	Ignored; set to binary zero.
108	128	Public key generator, g. $1 < g < p$ .
236	128	Prime modulus (large public modulus), p. $2^{L-1} < p < 2^L$ and L (the modulus length) must be a multiple of 64.
364	020	Prime divisor (small public modulus), q. $2^{159} < q < 2^{160}$ .
384	004	Reserved; set to binary zero.
388	024	Random number, confounder. <b>Note:</b> This field and the next two fields are enciphered for key confidentiality when the key security flag (offset 28) indicates the private key is enciphered.
412	020	Secret DSS key, x; x is random. (See the preceding note.)
432	004	Random number, generated when the secret key is generated. (See the preceding note.)
<i>DSS Public Key Section (required)</i>		
000	001	X'03', section identifier, DSS public key.
001	001	X'00', version.
002	002	Section length, 14+yyy.
004	002	Size of p in bits. The size of p must be one of: 512, 576, 640, 704, 768, 832, 896, 960, or 1024.
006	002	Size of the p field in bytes, which is zero for a private token.
008	002	Size of the q field in bytes, which is zero for a private token.
010	002	Size of the g field in bytes, which is zero for a private token.

Table 374. DSS Private External Key Token (continued)

Offset (Dec)	Number of Bytes	Description
012	002	Size of the y field in bytes, "yyy".
014	yyy	Public key, y. $y = g^x \text{ mod}(p)$ <b>Note:</b> p, q, and y are defined in the DSS public key token.
<b>Private Key Name (optional)</b>		
000	001	X'10', section identifier, private key. name
001	001	X'00', version.
002	002	Section length, X'0044' (68 decimal).
004	064	Private key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.

## DSS Private Internal Key Token

A DSS private internal key token contains the following sections:

- A required PKA token header, starting with the token identifier X'1F'
- A required DSS private key section, starting with the section identifier X'01'
- A required DSS public key section, starting with the section identifier X'03'
- An optional private key name section, starting with the section identifier X'10'
- A required internal information section, starting with the eyecatcher 'PKTN'

Table 375 presents the format of a DSS private internal token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format). All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

Table 375. DSS Private Internal Key Token

Offset (Dec)	Number of Bytes	Description
<b>Token Header (required)</b>		
000	001	Token identifier. X'1F' indicates an internal token. The private key is enciphered with a PKA master key.
001	001	Version, X'00'.
002	002	Length of the key token structure excluding the internal information section.
004	004	Ignored; should be zero.
<b>DSS Private Key Section and Secured Subsection (required)</b>		
000	001	X'01', section identifier, DSS private key.
001	001	X'00', version.
002	002	Length of the DSS private key section, 436, X'01B4'.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.
024	004	Reserved; set to binary zero.
028	001	Key security: X'01' DSS private key.
029	001	Format of external key token: X'10' Private key generated on an ICSF host. X'11' External private key was specified in the clear. X'12' External private key was encrypted.

Table 375. DSS Private Internal Key Token (continued)

Offset (Dec)	Number of Bytes	Description
030	020	SHA-1 hash of the key token structure contents that follow the public key section. If no sections follow, this field is set to binary zeros.
050	010	Reserved; set to binary zero.
060	048	The OPK encrypted under a PKA master key (Signature Master Key (SMK)).
108	128	Public key generator, $g$ . $1 < g < p$ .
236	128	Prime modulus (large public modulus), $p$ . $2^{L-1} < p < 2^L$ for $512 \leq L \leq 1024$ , and $L$ (the modulus length) must be a multiple of 64.
364	020	Prime divisor (small public modulus), $q$ . $2^{159} < q < 2^{160}$ .
384	004	Reserved; set to binary zero.
388	024	Random number, confounder. <b>Note:</b> This field and the two that follow are enciphered under the OPK.
412	020	Secret DSS key, $x$ . $x$ is random. (See the preceding note.)
432	004	Random number, generated when the secret key is generated. (See the preceding note.)
<b>DSS Public Key Section (required)</b>		
000	001	X'03', section identifier, DSS public key.
001	001	X'00', version.
002	002	Section length, 14+yyy.
004	002	Size of $p$ in bits. The size of $p$ must be one of: 512, 576, 640, 704, 768, 832, 896, 960, or 1024.
006	002	Size of the $p$ field in bytes, which is zero for a private token.
008	002	Size of the $q$ field in bytes, which is zero for a private token.
010	002	Size of the $g$ field in bytes, which is zero for a private token.
012	002	Size of the $y$ field in bytes, "yyy".
014	yyy	Public key, $y$ . $y = g^x \text{ mod}(p)$ ; <b>Note:</b> $p$ , $g$ , and $y$ are defined in the DSS public key token.
<b>Private Key Name (optional)</b>		
000	001	X'10', section identifier, private key name.
001	001	X'00', version.
002	002	Section length, X'0044' (68 decimal).
004	064	Private key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.
<b>Internal Information Section (required)</b>		
000	004	Eye catcher 'PKTN'.

Table 375. DSS Private Internal Key Token (continued)

Offset (Dec)	Number of Bytes	Description
004	004	PKA token type.
		<b>Bit</b> <b>Meaning When Set On</b>
		<b>0</b> RSA key.
		<b>1</b> DSS key.
		<b>2</b> Private key.
		<b>3</b> Public key.
		<b>4</b> Private key name section exists.
008	004	Address of token header.
012	002	Length of internal work area.
014	002	Count of number of sections.
016	016	PKA master key hash pattern.
032	016	Reserved.

## ECC Key Token Format

The following table presents the format of the ECC Key Token.

Table 376. ECC Key Token Format

Offset (Dec)	Number of Bytes	Description
Token Header		
000	001	Token identifier. <b>X'00'</b> Null token <b>X'1E'</b> External token <b>X'1F'</b> Internal token; the private key is protected by the master key
001	001	Version, X'00'.
002	002	Length of the key token structure excluding the internal information section.
004	004	Ignored; should be zero.
ECC Token Private section		
000	001	X'20', section identifier, ECC private key
001	001	X'00', version.
002	002	Section length.
004	001	Wrapping Method: This value indicates the wrapping method used to protect the data in the encrypted section. It is not the method used to protect the Object Protection Key (OPK). <b>X'00'</b> Clear – section is unencrypted. <b>X'01'</b> AESKW <b>X'02'</b> CBC Wrap - Other

Table 376. ECC Key Token Format (continued)

Offset (Dec)	Number of Bytes	Description
005	001	Hash used for Wrapping X'01' SHA224 X'02' SHA256 X'04' Reserved. X'08' Reserved
006	002	Reserved Binary Zero
008	001	Key Usage: X'C0' Key Agreement X'80' Both signature generation and key agreement X'00' Signature generation only X'02' Translate allowed The two high-order bits indicate permitted key usage in the decryption of symmetric keys and in the generation of digital signatures. The bit in the second nibble indicates if the key is translatable. A key is translatable if it can be re-encrypted from one key encrypting key to another.
009	001	Curve type: X'00' Prime curve X'01' Brainpool curve
010	001	Key Format and Security Flag. External Token: X'40' Unencrypted ECC private key identifier X'42' Encrypted ECC private key identifier Internal Token: X'08' Encrypted ECC private key identifier
011	001	Reserved Binary Zero
012	002	Length of p in bits X'00C0' Prime P-192 X'00E0' Prime P-224 X'0100' Prime P-256 X'0180' Prime P-384 X'0209' Prime P-521 X'00A0' Brainpool p-160 X'00C0' Brainpool P-192 X'00E0' Brainpool P-224 X'0100' Brainpool P-256 X'0140' Brainpool P-320 X'0180' Brainpool P-384 X'0200' Brainpool P-512)

Table 376. ECC Key Token Format (continued)

Offset (Dec)	Number of Bytes	Description
014	002	IBM Associated Data length. The length of this field must be greater than or equal to 16
016	008	External Token: <ul style="list-style-type: none"> <li>• Unencrypted – Reserved Binary 0x'00'</li> <li>• Encrypted – KVP of the AESKEK</li> </ul> Internal Token: MKVP of the ECC-MK
024	048	External Token: reserved binary zeros.  Internal Token: Object Protection Key (OPK), ICV (Integrity Check value), 8 byte confounder and a 256-bit AES key used with the AESKW algorithm to encrypt the ECC private key.  The OPK is encrypted by the AES master key using AESKW as well. Example format for OPK data passed to AESKW: <ul style="list-style-type: none"> <li>• 8 bytes = A6A6A6A6A6A60000</li> <li>• 40 bytes = Confounder(8)/Key(32)</li> </ul>
072	002	Associated data length, aa
074	002	Length of formatted section in bytes, bb
076	aa	Associated data (See Table 377 on page 876 for the Associated Data format).
076 + aa	Start of formatted section	If this section is in the clear it contains private key d.  If it is encrypted it contains the AESKW wrapped payload.
76 + aa	bb	Formatted section which includes Private key d  See Table 378 on page 877 for the format of the AESKW Wrapped Payload
76 + aa + bb	End of formatted section	
<b>ECC Token Public Section</b>		
000	001	X'21', section identifier
001	001	X'00', version.
002	002	Section length
004	004	Reserved field, binary zero
008	001	Curve type  X'00' Prime curve X'01' Brainpool curve
009	001	Reserved field, binary zero

Table 376. ECC Key Token Format (continued)

Offset (Dec)	Number of Bytes	Description
010	002	Length of p in bits: X'00C0' Prime P-192 X'00E0' Prime P-224 X'0100' Prime P-256 X'0180' Prime P-384 X'0209' Prime P-521 X'00A0' Brainpool P-160 X'00C0' Brainpool P-192 X'00E0' Brainpool P-224 X'0100' Brainpool P-256 X'0140' Brainpool P-320 X'0180' Brainpool P-384 X'0200' Brainpool P-512
012	002	This field is the length of the public key q value in bytes, the maximum value could be up to 133 bytes, cc. The value includes the key material length and one byte to indicate if the key material is compressed or uncompressed.
014	cc	Public Key , q field

### Associated Data Format for ECC Token

The table below defines the associated data as it is stored in the ECC token in the clear. Associated data is data whose integrity but not confidentiality is protected by a key wrap mechanism.

Table 377. Associated Data Format for ECC Private Key Token

Offset (Dec)	Number of Bytes	Description
000	001	Associated Data Version. 0 for ECC
001	001	Length of Key Label, kl
002	002	IBM Associated Data length, 16 + kl + xxx
004	002	IBM Extended Associated Data length, xxx
006	001	User Definable Associated Data length, yyy. User definable lengths are from 0 bytes to 100 bytes.
007	001	Curve Type
008	002	Length of p in bits
010	001	Usage flag
011	001	Format and Security flag
012	004	reserved
016	kl	Key Label (optional)
016 + kl	xxx	IBM Extended Associated Data
016 + kl + xxx	yyy	User-definable Associated Data

## AESKW Wrapped Payload Format for ECC Private Key Token

This table defines the contents of the AESKW payload: data will be copied into this format, then encrypted with the OPK according to the AESKW specification, and the result will be stored in the encrypted data section.

Table 378. AESKW Wrapped Payload Format for ECC Private Key Token

Offset (Dec)	Number of Bytes	Description
000	006	ICV ('A6'....)
006	001	Length of padding in bits
007	001	Length of the hash of the associated data in bytes, ii
008	004	Hash options
012	ii	Hash of Associated Data
12+ii	mm	Key data
12+ii+mm	0-7	Padding to a multiple of 8 bytes

## Trusted Block Key Token

A trusted block key-token (trusted block) is an extension of CCA PKA key tokens using new section identifiers. A trusted block was introduced to CCA beginning with Release 3.25. They are an integral part of a remote key-loading process.

Trusted blocks contain various items, some of which are optional, and some of which can be present in different forms. Tokens are composed of concatenated sections that, unlike CCA PKA key tokens, occur in no prescribed order.

As with other CCA key-tokens, both internal and external forms are defined:

- An external trusted block contains a randomly generated confounder and a triple-length MAC key enciphered under a DES IMP-PKA transport key. The MAC key is used to calculate an ISO 16609 CBC mode TDES MAC of the trusted block contents. An external trusted block is created by the Trusted\_Block\_Create verb. This verb can:
  1. Create an inactive external trusted block
  2. Change an external trusted block from inactive to active
- An internal trusted block contains a confounder and triple-length MAC key enciphered under a variant of the PKA master key. The MAC key is used to calculate a TDES MAC of the trusted block contents. A PKA master key verification pattern is also included to enable determination that the proper master key is available to process the key. The Remote\_Key\_Export verb only operates on trusted blocks that are internal. An internal trusted block must be imported from an external trusted block that is active using the PKA\_Key\_Import verb.

**Note:** Trusted blocks do not contain a private key section.

### Trusted block sections

A trusted block is a concatenation of a header followed by an unordered set of sections. The data structures of these sections are summarized in the following table:

Section	Reference	Usage
Header	Table 379 on page 879	Trusted block token header
X'11'	Table 380 on page 880	Trusted block public key



Section	Reference	Usage
X'12'	Table 381 on page 881	Trusted block rule
X'13'	Table 388 on page 888	Trusted block name (key label)
X'14'	Table 389 on page 888	Trusted block information
X'15'	Table 393 on page 890	Trusted block application-defined data

Every trusted block starts with a token header. The first byte of the token header determines the key form:

- An external header (first byte X'1E'), created by the `Trusted_Block_Create` verb
- An internal header (first byte X'1F'), imported from an active external trusted block by the `PKA_Key_Import` verb

Following the token header of a trusted block is an unordered set of sections. A trusted block is formed by concatenating these sections to a trusted block header:

- An optional public-key section (trusted block section identifier X'11')  
The trusted block trusted RSA public-key section includes the key itself in addition to a key-usage flag. No multiple sections are allowed.
- An optional rule section (trusted block section identifier X'12')  
A trusted block may have zero or more rule sections.
  1. A trusted block with no rule sections can be used by the `PKA_Key_Token_Change` and `PKA_Key_Import` callable services. A trusted block with no rule sections can also be used by the `Digital_Signature_Verify` verb, provided there is an RSA public-key section that has its key-usage flag bits set to allow digital signature operations.
  2. At least one rule section is required when the `Remote_Key_Export` verb is used to:
    - Generate an RKX key-token
    - Export an RKX key-token
    - Export a CCA DES key-token
    - Encrypt the clear generated or exported key using the provided vendor certificate
  3. If a trusted block has multiple rule sections, each rule section must have a unique 8-character Rule ID.
- An optional name (key label) section (trusted block section identifier X'13')  
The trusted block name section provides a 64-byte variable to identify the trusted block, just as key labels are used to identify other CCA keys. This name, or label, enables a host access-control system such as RACF to use the name to verify that the application has authority to use the trusted block. No multiple sections are allowed.
- A required information section (trusted block section identifier X'14')  
The trusted block information section contains control and security information related to the trusted block. The information section is required while the others are optional. This section contains the cryptographic information that guarantees its integrity and binds it to the local system. No multiple sections are allowed.
- An optional application-defined data section (trusted block section identifier X'15')  
The trusted block application-defined data section can be used to include application-defined data in the trusted block. The purpose of the data in this section is defined by the application. CCA does not examine or use this data in any way. No multiple sections are allowed.

## Trusted block integrity

An enciphered confounder and triple-length MAC key contained within the required information section of the trusted block is used to protect the integrity of the trusted block. The randomly generated MAC key is used to calculate an ISO 16609 CBC mode TDES MAC of the trusted block contents. Together, the MAC key and MAC value provide a way to verify that the trusted block originated from an authorized source, and binds it to the local system.

An external trusted block has its MAC key enciphered under an IMP-PKA key-encrypting key. An internal trusted block has its MAC key enciphered under a variant of the PKA master key, and the master key verification pattern is stored in the information section.

## Number representation in trusted blocks

- All length fields are in binary
- All binary fields (exponents, lengths, and so forth) are stored with the high-order byte first (left, low-address, z/OS format); thus the least significant bits are to the right and preceded with zero-bits to the width of a field
- In variable-length binary fields that have an associated field-length value, leading bytes that would otherwise contain X'00' can be dropped and the field shortened to contain only the significant bits

## Format of trusted block sections

At the beginning of every trusted block is a trusted block header. The header contains the following information:

- A token identifier, which specifies if the token contains an external or internal key-token
- A token version number to allow for future changes
- A length in bytes of the trusted block, including the length of the header

The trusted block header is defined in the following table:

*Table 379. Trusted block header*

Offset (bytes)	Length (bytes)	Description
000	001	Token identifier (a flag that indicates token type) X'1E' External trusted block token X'1F' Internal trusted block token
001	001	Token version number (X'00').
002	002	Length of the key-token structure in bytes.
004	004	Reserved, binary zero.

**Note:** See "Number representation in trusted blocks."

Following the header, in no particular order, are trusted block sections. There are five different sections defined, each identified by a one-byte section identifier (X'11' - X'15'). Two of the five sections have subsections defined. A subsection is a tag-length-value (TLV) object, identified by a two-byte subsection tag.

Only sections X'12' and X'14' have subsections defined; the other sections do not. A section and its subsections, if any, are one contiguous unit of data. The subsections are concatenated to the related section, but are otherwise in no particular order. Section X'12' has five subsections defined (X'0001' - X'0005'), and section X'14' has

two (X'0001' and X'0002'). Of all the subsections, only subsection X'0001' of section X'14' is required. Section X'14' is also required.

The trusted block sections and subsections are described in detail in the following sections.

**Trusted block section X'11':** Trusted block section X'11' contains the trusted RSA public key in addition to a key-usage flag indicating whether the public key is usable in key-management operations, digital signature operations, or both.

Section X'11' is optional. No multiple sections are allowed. It has no subsections defined.

This section is defined in the following table:

*Table 380. Trusted block trusted RSA public-key section (X'11')*

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier:  X'11' Trusted block trusted RSA public key
001	001	Section version number (X'00').
002	002	Section length (16+xxx+yyy).
004	002	Reserved, must be binary zero.
006	002	RSA public-key exponent field length in bytes, xxx.
008	002	RSA public-key modulus length in bits.
010	002	RSA public-key modulus field length in bytes, yyy.
012	xxx	Public-key exponent, $e$ (this field length is typically 1, 3, or 64 - 512 bytes). $e$ must be odd and $1 \leq e < n$ . ( $e$ is frequently valued to 3 or $2^{16}+1$ (=65537), otherwise $e$ is of the same order of magnitude as the modulus). <b>Note:</b> Although the current product implementation does not generate such a public key, you can import an RSA public key having an exponent valued to two (2). Such a public key (a Rabin key) can correctly validate an ISO 9796-1 digital signature.
012+xxx	yyy	RSA public-key modulus, $n$ . $n=pq$ , where $p$ and $q$ are prime and $2^{512} \leq n < 2^{4096}$ . The field length is 64 - 512 bytes.
012+xxx+yyy	004	Flags: X'00000000' Trusted block public key can be used in digital signature operations only X'80000000' Trusted block public key can be used in both digital signature and key management operations X'C0000000' Trusted block public key can be used in key management operations only

**Note:** See "Number representation in trusted blocks" on page 879.

**Trusted block section X'12':** Trusted block section X'12' contains information that defines a rule. A trusted block may have zero or more rule sections.

1. A trusted block with no rule sections can be used by the PKA\_Key\_Token\_Change and PKA\_Key\_Import callable services. A trusted block with no rule sections can be used by the Digital\_Signature\_Verify verb, provided there is an RSA public-key section that has its key-usage flag set to allow digital signature operations.
2. At least one rule section is required when the Remote\_Key\_Export verb is used to:

- Generate an RKX key-token
  - Export an RKX key-token
  - Export a CCA DES key-token
  - Generate or export a key encrypted by a public key. The public key is contained in a vendor certificate (section X'11'), and is the root certification key for the ATM vendor. It is used to verify the digital signature on public-key certificates for specific individual ATMs.
3. If a trusted block has multiple rule sections, each rule section must have a unique 8-character Rule ID.

Section X'12' is the only section allowed to have multiple sections. Section X'12' is optional. Multiple sections are allowed.

**Note:** The overall length of the trusted block may not exceed its maximum size of 3500 bytes.

Five subsections (TLV objects) are defined.

This section is defined in the following table:

*Table 381. Trusted block rule section (X'12')*

Offset (bytes)	Length (bytes)	Description								
000	001	Section identifier: <b>X'12'</b> Trusted block rule								
001	001	Section version number (X'00').								
002	002	Section length in bytes (20+yyy).								
004	008	Rule ID (in ASCII).  An 8-byte character string that uniquely identifies the rule within the trusted block.  Valid ASCII characters are: A...Z, a...z, 0...9, - (hyphen), and _ (underscore), left justified and padded on the right with space characters.								
012	004	Flags (undefined flag bits are reserved and must be zero). <b>X'00000000'</b> Generate new key <b>X'00000001'</b> Export existing key								
016	001	Generated key length.  Length in bytes of key to be generated when flags value (offset 012) is set to generate a new key; otherwise ignore this value. Valid values are 8, 16, or 24; return an error if not valid.								
017	001	Key-check algorithm identifier (all others are reserved and must not be used): <table><tr><th>Value</th><th>Meaning</th></tr><tr><td><b>X'00'</b></td><td>Do not compute key-check value. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to zero.</td></tr><tr><td><b>X'01'</b></td><td>Encrypt an 8-byte block of binary zeros with the key. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to 8.</td></tr><tr><td><b>X'02'</b></td><td>Compute the MDC-2 hash of the key. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to 16.</td></tr></table>	Value	Meaning	<b>X'00'</b>	Do not compute key-check value. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to zero.	<b>X'01'</b>	Encrypt an 8-byte block of binary zeros with the key. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to 8.	<b>X'02'</b>	Compute the MDC-2 hash of the key. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to 16.
Value	Meaning									
<b>X'00'</b>	Do not compute key-check value. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to zero.									
<b>X'01'</b>	Encrypt an 8-byte block of binary zeros with the key. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to 8.									
<b>X'02'</b>	Compute the MDC-2 hash of the key. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to 16.									

Table 381. Trusted block rule section (X'12') (continued)

Offset (bytes)	Length (bytes)	Description								
018	001	<p>Symmetric encrypted output key format flag (all other values are reserved and must not be used).</p> <p>Return the indicated symmetric key-token using the <i>sym_encrypted_key_identifier</i> parameter.</p> <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>X'00'</td><td>Return an RKX key-token encrypted under a variant of the MAC key. <b>Note:</b> This is the only key format permitted when the flags value (offset 012) is set to generate a new key.</td></tr><tr><td>X'01'</td><td>Return a CCA DES key-token encrypted under a transport key. <b>Note:</b> This is the only key format permitted when the flags value (offset 012) is set to export an existing key.</td></tr></table>	Value	Meaning	X'00'	Return an RKX key-token encrypted under a variant of the MAC key. <b>Note:</b> This is the only key format permitted when the flags value (offset 012) is set to generate a new key.	X'01'	Return a CCA DES key-token encrypted under a transport key. <b>Note:</b> This is the only key format permitted when the flags value (offset 012) is set to export an existing key.		
Value	Meaning									
X'00'	Return an RKX key-token encrypted under a variant of the MAC key. <b>Note:</b> This is the only key format permitted when the flags value (offset 012) is set to generate a new key.									
X'01'	Return a CCA DES key-token encrypted under a transport key. <b>Note:</b> This is the only key format permitted when the flags value (offset 012) is set to export an existing key.									
019	001	<p>Asymmetric encrypted output key format flag (all other values are reserved and must not be used).</p> <p>Return the indicated asymmetric key-token in the <i>asym_encrypted_key</i> variable.</p> <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>X'00'</td><td>Do not return an asymmetric key. Set the <i>asym_encrypted_key_length</i> variable to zero.</td></tr><tr><td>X'01'</td><td>Output in PKCS1.2 format.</td></tr><tr><td>X'02'</td><td>Output in RSAOAEP format.</td></tr></table>	Value	Meaning	X'00'	Do not return an asymmetric key. Set the <i>asym_encrypted_key_length</i> variable to zero.	X'01'	Output in PKCS1.2 format.	X'02'	Output in RSAOAEP format.
Value	Meaning									
X'00'	Do not return an asymmetric key. Set the <i>asym_encrypted_key_length</i> variable to zero.									
X'01'	Output in PKCS1.2 format.									
X'02'	Output in RSAOAEP format.									
020	yyy	Rule section subsections (tag-length-value objects). A series of 0 - 5 objects in TLV format.								

**Note:** See “Number representation in trusted blocks” on page 879.

Section X'12' has five rule subsections (tag-length-value objects) defined. These subsections are summarized in the following table:

Table 382. Summary of trusted block rule subsection

Rule subsection tag	TLV object	Optional or required	Comments
X'0001'	Transport key variant	Optional	Contains variant to be exclusive-ORed into the cleartext transport key.
X'0002'	Transport key rule reference	Optional; required to use an RKX key-token as a transport key	Contains the rule ID for the rule that must have been used to create the transport key.
X'0003'	Common export key parameters	Optional for key generation; required for key export of an existing key	Contains the export key and source key minimum and maximum lengths, an output key variant length and variant, a CV length, and a CV to be exclusive-ORed with the cleartext transport key to control usage of the key.
X'0004'	Source key reference	Optional; required if the source key is an RKX key-token	Contains the rule ID for the rule used to create the source key. <b>Note:</b> Include all rules that will ever be needed when a trusted block is created. A rule cannot be added to a trusted block after it has been created.

Table 382. Summary of trusted block rule subsection (continued)

Rule subsection tag	TLV object	Optional or required	Comments
X'0005'	Export key CCA token parameters	Optional; used for export of CCA DES key tokens only	<p>Contains mask length, mask, and CV template to limit the usage of the exported key. Also contains the template length and template which defines which source key labels are allowed.</p> <p>The key type of a source key input parameter can be "filtered" by using the export key CV limit mask (offset 005) and limit template (offset 005+yyy) in this subsection.</p>

**Note:** See "Number representation in trusted blocks" on page 879.

**Trusted block section X'12' subsection X'0001':** Subsection X'0001' of the trusted block rule section (X'12') is the transport key variant TLV object. This subsection is optional. It contains a variant to be exclusive-ORed into the cleartext transport key.

This subsection is defined in the following table:

Table 383. Transport key variant subsection (X'0001' of trusted block rule section (X'12'))

Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: X'0001' Transport key variant TLV object
002	002	Subsection length in bytes (8+nnn).
004	001	Subsection version number (X'00').
005	002	Reserved, must be binary zero.
007	001	<p>Length of variant field in bytes (nnn).</p> <p>This length must be greater than or equal to the length of the transport key that is identified by the <i>transport_key_identifier</i> parameter. If the variant is longer than the key, truncate it on the right to the length of the key prior to use.</p>
008	nnn	<p>Transport key variant.</p> <p>Exclusive-OR this variant into the cleartext transport key, provided: (1) the length of the variant field value (offset 007) is not zero, and (2) the symmetric encrypted output key format flag (offset 018 in section X'12') is X'01'.</p> <p><b>Note:</b> A transport key is not used when the symmetric encrypted output key is in RKX key-token format.</p>

**Note:** See "Number representation in trusted blocks" on page 879.

**Trusted block section X'12' subsection X'0002':** Subsection X'0002' of the trusted block rule section (X'12') is the transport key rule reference TLV object. This subsection is optional. It contains the rule ID for the rule that must have been used to create the transport key. This subsection must be present to use an RKX key-token as a transport key.

This subsection is defined in the following table:

Table 384. Transport key rule reference subsection (X'0002') of trusted block rule section (X'12')

Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: <b>X'0002'</b> Transport key rule reference TLV object
002	002	Subsection length in bytes (14).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.
006	008	Rule ID.  Contains the rule identifier for the rule that must have been used to create the RKX key-token used as the transport key.  The Rule ID is an 8-byte string of ASCII characters, left justified and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.

**Trusted block section (X'12') subsection X'0003':** Subsection X'0003' of the trusted block rule section (X'12') is the common export key parameters TLV object. This subsection is optional, but is required for the key export of an existing source key (identified by the *source\_key\_identifier* parameter) in either RKX key-token format or CCA DES key-token format. For new key generation, this subsection applies the output key variant to the cleartext generated key, if such an option is desired. It contains the input source key and output export key minimum and maximum lengths, an output key variant length and variant, a CV length, and a CV to be exclusive-ORed with the cleartext transport key.

This subsection is defined in the following table:

Table 385. Common export key parameters subsection (X'0003') of trusted block rule section (X'12')

Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: <b>X'0003'</b> Common export key parameters TLV object
002	002	Subsection length in bytes (12+xxx+yyy).
004	001	Subsection version number (X'00').
005	002	Reserved, must be binary zero.
007	001	Flags (must be set to binary zero).
008	001	Export key minimum length in bytes. Length must be 8, 16, or 24.  Also applies to the source key.
009	001	Export key maximum length in bytes (yyy). Length must be 8, 16, or 24.  Also applies to the source key.
010	001	Output key variant length in bytes (xxx).  Valid values are 0 or 8 - 255. If greater than 0, the length must be at least as long as the longest key ever to be exported using this rule. If the variant is longer than the key, truncate it on the right to the length of the key prior to use. <b>Note:</b> The output key variant (offset 011) is not used if this length is zero.

Table 385. Common export key parameters subsection (X'0003') of trusted block rule section (X'12') (continued)

Offset (bytes)	Length (bytes)	Description
011	xxx	Output key variant.  The variant can be any value. Exclusive-OR this variant into the cleartext value of the output.
011+xxx	001	CV length in bytes ( <i>yyy</i> ). <ul style="list-style-type: none"> <li>If the length is not 0, 8, or 16, return an error.</li> <li>If the length is 0, and if the source key is a CCA DES key-token, preserve the CV in the symmetric encrypted output if the output is to be in the form of a CCA DES key-token.</li> <li>If a non-zero length is less than the length of the key identified by the <i>source_key_identifier</i> parameter, return an error.</li> <li>If the length is 16, and if the CV (offset 012+xxx) is valued to 16 bytes of X'00' (ignoring the key-part bit), then: <ol style="list-style-type: none"> <li>Ignore all CV bit definitions</li> <li>If CCA DES key-token format, set the flag byte of the symmetric encrypted output key to indicate a CV value is present.</li> <li>If the source key is 8 bytes in length, do not replicate the key to 16 bytes.</li> </ol> </li> </ul>
012+xxx	yyy	CV.  Place this CV into the output exported key-token, provided that the symmetric encrypted output key format selected (offset 018 in rule section) is CCA DES key-token. <ul style="list-style-type: none"> <li>If the symmetric encrypted output key format flag (offset 018 in section X'12') indicates return an RKX key-token (X'00'), then ignore this CV. Otherwise, exclusive-OR this CV into the cleartext transport key.</li> <li>Exclusive-OR the CV of the source key into the cleartext transport key if the CV length (offset 011+xxx) is set to 0. If a transport key to encrypt a source key has equal left and right key halves, return an error. Replicate the key halves of the key identified by the <i>source_key_identifier</i> parameter whenever all of these conditions are met: <ol style="list-style-type: none"> <li>The Replicate Key command (offset X'00DB') is enabled in the active role</li> <li>The CV length (offset 011+xxx) is 16, and both CV halves are non-zero</li> <li>The <i>source_key_identifier</i> parameter (contained in either a CCA DES key-token or RKX key-token) identifies an 8-byte key</li> <li>The key-form bits (40 - 42) of this CV do not indicate a single-length key (are not set to zero)</li> <li>Key-form bit 40 of this CV does not indicate the key is to have guaranteed unique halves (is not set to 1).</li> </ol> </li> </ul> <p><b>Note:</b> A transport key is not used when the symmetric encrypted output key is in RKX key-token format.</p>

**Note:** See “Number representation in trusted blocks” on page 879.

**Trusted block section X'12' subsection X'0004':** Subsection X'0004' of the trusted block rule section (X'12') is the source key rule reference TLV object. This subsection is optional, but is required if using an RKX key-token as a source key (identified by *source\_key\_identifier* parameter). It contains the rule ID for the rule used to create the export key. If this subsection is not present, an RKX key-token format source key will not be accepted for use.



This subsection is defined in the following table:

*Table 386. Source key rule reference subsection (X'0004' of trusted block rule section (X'12')*

Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: <b>X'0004'</b> Source key rule reference TLV object
002	002	Subsection length in bytes (14).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.
006	008	Rule ID.  Rule identifier for the rule that must have been used to create the source key.  The Rule ID is an 8-byte string of ASCII characters, left justified and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F). All other characters are reserved for future use.

**Note:** See "Number representation in trusted blocks" on page 879.

**Trusted block section X'12' subsection X'0005':** Subsection X'0005' of the trusted block rule section (X'12') is the export key CCA token parameters TLV object. This subsection is optional. It contains a mask length, mask, and template for the export key CV limit. It also contains the template length and template for the source key label. When using a CCA DES key-token as a source key input parameter, its key type can be "filtered" by using the export key CV limit mask (offset 005) and limit template (offset 005+yyy) in this subsection.

This subsection is defined in the following table:

*Table 387. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12')*

Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: <b>X'0005'</b> Export key CCA token parameters TLV object
002	002	Subsection length in bytes (10+yyy+yyy+zzz).
004	001	Subsection version number (X'00').
005	002	Reserved, must be binary zero.
007	001	Flags (must be set to binary zero).
008	001	Export key CV limit mask length in bytes (yyy).  Do not use CV limits if this CV limit mask length (yyy ) is zero. Use CV limits if yyy is non-zero, in which case yyy: <ul style="list-style-type: none"> <li>• Must be 8 or 16</li> <li>• Must not be less than the export key minimum length (offset 008 in subsection X'0003')</li> <li>• Must be equal in length to the actual source key length of the key</li> </ul> <b>Example:</b> An export key minimum length of 16 and an export key CV limit mask length of 8 returns an error.

Table 387. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12') (continued)

Offset (bytes)	Length (bytes)	Description
009	yyy	<p>Export key CV limit mask (does not exist if yyy=0).</p> <p>Indicates which CV bits to check against the source key CV limit template (offset 009+yyy).</p> <p><b>Examples:</b> A mask of X'FF' means check all bits in a byte. A mask of X'FE' ignores the parity bit in a byte.</p>
009+yyy	yyy	<p>Export key CV limit template (does not exist if yyy=0).</p> <p>Specifies the required values for those CV bits that are checked based on the export key CV limit mask (offset 009).</p> <p>The export key CV limit mask and template have the same length, yyy. This is because these two variables work together to restrict the acceptable CVs for CCA DES key tokens to be exported. The checks work as follows:</p> <ol style="list-style-type: none"> <li>1. If the length of the key to be exported is less than yyy, return an error</li> <li>2. Logical AND the CV for the key to be exported with the export key CV limit mask</li> <li>3. Compare the result to the export key CV limit template</li> <li>4. Return an error if the comparison is not equal</li> </ol> <p><b>Examples:</b> An export key CV limit mask of X'FF' for CV byte 1 (key type) along with an export key CV limit template of X'3F' (key type CVARENC) for byte 1 filters out all key types except CVARENC keys.</p> <p><b>Note:</b> Using the mask and template to permit multiple key types is possible, but cannot consistently be achieved with one rule section. For example, setting bit 10 to 1 in the mask and the template permits PIN processing keys and cryptographic variable encrypting keys, and only those keys. However, a mask to permit PIN-processing keys and key-encrypting keys, and only those keys, is not possible. In this case, multiple rule sections are required, one to permit PIN-processing keys and the other to permit key-encrypting keys.</p>
009+ yyy+ yyy	001	<p>Source key label template length in bytes (zzz).</p> <p>Valid values are 0 and 64. Return an error if the length is 64 and a source key label is not provided.</p>
010+ yyy+ yyy	zzz	<p>Source key label template (does not exist if zzz=0).</p> <p>If a key label is identified by the <i>source_key_identifier</i> parameter, verify that the key label name matches this template. If the comparison fails, return an error. The source key label template must conform to the following rules:</p> <ul style="list-style-type: none"> <li>• The key label template must be 64 bytes in length</li> <li>• The first character cannot be in the range X'00' - X'1F', nor can it be X'FF'</li> <li>• The first character cannot be numeric (X'30' - X'39')</li> <li>• A key label name is terminated by a space character (X'20') on the right and must be padded on the right with space characters</li> <li>• The only special characters permitted are #, \$, @, and * (X'23', X'24', X'40', and X'2A')</li> <li>• The wildcard X'2A' (*) is only permitted as the first character, the last character, or the only character in the template</li> <li>• Only alphanumeric characters (a...z, A...Z, 0...9), the four special characters (X'23', X'24', X'40', and X'2A'), and the space character (X'20') are allowed</li> </ul>

**Note:** See “Number representation in trusted blocks” on page 879.

**Trusted block section X'13':** Trusted block section X'13' contains the name (key label). The trusted block name section provides a 64-byte variable to identify the trusted block, just as key labels are used to identify other CCA keys. This name, or label, enables a host access-control system such as RACF to use the name to verify that the application has authority to use the trusted block.

Section X'13' is optional. No multiple sections are allowed. It has no subsections defined. This section is defined in the following table:

*Table 388. Trusted block key label (name) section X'13'*

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: <b>X'13'</b> Trusted block name (key label)
001	001	Section version number (X'00').
002	002	Section length in bytes (68).
004	064	Name (key label).

**Note:** See “Number representation in trusted blocks” on page 879.

**Trusted block section X'14':** Trusted block section X'14' contains control and security information related to the trusted block. This information section is separate from the public key and other sections because this section is required while the others are optional. This section contains the cryptographic information that guarantees its integrity and binds it to the local system.

Section X'14' is required. No multiple sections are allowed. Two subsections are defined. This section is defined in the following table:

*Table 389. Trusted block information section X'14'*

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: <b>X'14'</b> Trusted block information
001	001	Section version number (X'00').
002	002	Section length in bytes (10+xxx).
004	002	Reserved, binary zero.
006	004	Flags: <b>X'00000000'</b> Trusted block is in the inactive state <b>X'00000001'</b> Trusted block is in the active state
010	xxx	Information section subsections (tag-length-value objects). One or two objects in TLV format.

**Note:** See “Number representation in trusted blocks” on page 879.

Section X'14' has two information subsections (tag-length-value objects) defined. These subsections are summarized in the following table:

Table 390. Summary of trusted block information subsections

Rule subsection tag	TLV object	Optional or required	Comments
X'0001'	Protection information	Required	Contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key, the ISO 16609 TDES CBC MAC value, and the MKVP of the PKA master key (computed using MDC4).
X'0002'	Activation and expiration dates	Optional	Contains flags indicating whether or not the coprocessor is to validate dates, and contains the activation and expiration dates that are considered valid for the trusted block.

**Note:** See “Number representation in trusted blocks” on page 879.

**Trusted block section X'14' subsection X'0001':** Subsection X'0001' of the trusted block information section (X'14') is the protection information TLV object. This subsection is required. It contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key, the ISO-16609 TDES CBC MAC value, and the MKVP of the PKA master key (computed using MDC4).

This subsection is defined in the following table:

Table 391. Protection information subsection (X'0001') of trusted block information section (X'14')

Offset (bytes)	Length (bytes)	Description										
000	002	Subsection tag: <b>X'0001'</b> Trusted block information TLV object										
002	002	Subsection length in bytes (62).										
004	001	Subsection version number (X'00').										
005	001	Reserved, must be binary zero.										
006	032	Encrypted MAC key.  Contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key in the following format: <table><tr><th>Offset</th><th>Description</th></tr><tr><td>00 - 07</td><td>Confounder</td></tr><tr><td>08 - 15</td><td>Left key</td></tr><tr><td>16 - 23</td><td>Middle key</td></tr><tr><td>24 - 31</td><td>Right key</td></tr></table>	Offset	Description	00 - 07	Confounder	08 - 15	Left key	16 - 23	Middle key	24 - 31	Right key
Offset	Description											
00 - 07	Confounder											
08 - 15	Left key											
16 - 23	Middle key											
24 - 31	Right key											
038	008	MAC.  Contains the ISO-16609 TDES CBC message authentication code value.										
046	016	MKVP.  Contains the PKA master key verification pattern, computed using MDC4, when the trusted block is in internal form, otherwise contains binary zero.										

**Note:** See “Number representation in trusted blocks” on page 879.

**Trusted block section X'14' subsection X'0002':** Subsection X'0002' of the trusted block information section (X'14') is the activation and expiration dates TLV object. This subsection is optional. It contains flags indicating whether or not the coprocessor is to validate dates, and contains the activation and expiration dates that are considered valid for the trusted block.

This subsection is defined in the following table:

*Table 392. Activation and expiration dates subsection (X'0002') of trusted block information section (X'14')*

Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: <b>X'0002'</b> Activation and expiration dates TLV object
002	002	Subsection length in bytes (16).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.
006	002	Flags: <b>X'0000'</b> The coprocessor does not check dates. <b>X'0001'</b> The coprocessor checks dates.  Compare the activation date (offset 008) and the expiration date (offset 012) to the coprocessor's internal real-time clock. Return an error if the coprocessor date is before the activation date or after the expiration date.
008	004	Activation date.  Contains the first date that the trusted block can be used for generating or exporting keys. Format of the date is YYMD, where: <b>YY</b> Big-endian year (return an error if greater than 9999) <b>M</b> Month (return an error if any value other than X'01' - X'0C') <b>D</b> Day of month (return an error if any value other than X'01' - X'1F'; day must be valid for given month and year, including leap years)  Return an error if the activation date is after the expiration date or is not valid.
012	004	Expiration date.  Contains the last date that the trusted block can be used. Same format as activation date (offset 008). Return an error if date is not valid.

**Note:** See “Number representation in trusted blocks” on page 879.

**Trusted block section X'15':** Trusted block section X'15' contains application-defined data. The trusted block application-defined data section can be used to include application-defined data in the trusted block. The purpose of the data in this section is defined by the application; it is neither examined nor used by CCA in any way.

Section X'15' is optional. No multiple sections are allowed. It has no subsections defined. This section is defined in the following table:

*Table 393. Trusted block application-defined data section X'15'*

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: <b>X'15'</b> Application-defined data
001	001	Section version number (X'00').
002	002	Section length (6+xxx)

Table 393. Trusted block application-defined data section X'15' (continued)

Offset (bytes)	Length (bytes)	Description
004	002	Application data length ( <i>xxx</i> )  The value of <i>xxx</i> can be from 0 bytes to a length that does not cause the trusted block to exceed its maximum size of 3500 bytes.
006	<i>xxx</i>	Application-defined data  May be used to hold a public-key certificate for the trusted public key.

**Note:** See “Number representation in trusted blocks” on page 879.



---

## Appendix C. Control Vectors and Changing Control Vectors with the CVT Callable Service

This section contains a control vector table which displays the default value of the control vector that is associated with each type of key. It also describes how to change control vectors with the control vector translate callable service.

---

### Control Vector Table

**Note:** The Control Vectors used in ICSF are exactly the same as documented in CCA and the TSS documents.

The master key enciphers all keys operational on your system. A transport key enciphers keys that are distributed off your system. Before a master key or transport key enciphers a key, ICSF exclusive ORs both halves of the master key or transport key with a control vector. The same control vector is exclusive ORed to the left and right half of a master key or transport key.

Also, if you are entering a key part, ICSF exclusive ORs each half of the key part with a control vector before placing the key part into the CKDS.

Each type of key on ICSF (except the master key) has either one or two unique control vectors associated with it. The control vector that ICSF exclusive ORs the master key or transport key with depends on the type of key the master key or transport key is enciphering. For double-length keys, a unique control vector exists for each half of a specific key type. For example, there is a control vector for the left half of an input PIN-encrypting key, and a control vector for the right half of an input PIN-encrypting key.

If you are entering a key part into the CKDS, ICSF exclusive ORs the key part with the unique control vector(s) associated with the key type. ICSF also enciphers the key part with two master key variants for a key part. One master key variant enciphers the left half of the key part, and another master key variant enciphers the right half of the key part. ICSF creates the master key variants for a key part by exclusive ORing the master key with the control vectors for key parts. These procedures protect key separation.

Table 394 displays the default value of the control vector that is associated with each type of key. Some key types do not have a default control vector. For keys that are double-length, ICSF enciphers a unique control vector on each half. Control vectors indicated with an "\*" are supported by the Cryptographic Coprocessor Feature.

*Table 394. Default Control Vector Values*

Key Type	Control Vector Value (Hexadecimal Value for Left Half of Double-length Key)	Control Vector Value (Hexadecimal Value for Right Half of Double-length Key)
*AKEK	00 00 00 00 00 00 00 00	
CIPHER	00 03 71 00 03 00 00 00	
CIPHER (double length)	00 03 71 00 03 41 00 00	00 03 71 00 03 21 00 00



Table 394. Default Control Vector Values (continued)

Key Type	Control Vector Value (Hexadecimal Value for Left Half of Double-length Key)	Control Vector Value (Hexadecimal Value for Right Half of Double-length Key)
CIPHERXI	00 0C 50 00 03 C0 00 00	00 0C 50 00 03 A0 00 00
CIPHERXO	00 0C 60 00 03 C0 00 00	00 0C 60 00 03 A0 00 00
CIPHERXL	00 0C 71 00 03 C0 00 00	00 0C 71 00 03 A0 00 00
CVARDEC	00 3F 42 00 03 00 00 00	
CVARENC	00 3F 48 00 03 00 00 00	
CVARPINE	00 3F 41 00 03 00 00 00	
CVARXCVL	00 3F 44 00 03 00 00 00	
CVARXCVR	00 3F 47 00 03 00 00 00	
DATA (external)	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
DATA (internal)	00 00 7D 00 03 41 00 00	00 00 7D 00 03 21 00 00
*DATA	00 00 00 00 00 00 00 00	
DATAC	00 00 71 00 03 41 00 00	00 00 71 00 03 21 00 00
*DATAM generation key (external)	00 00 4D 00 03 41 00 00	00 00 4D 00 03 21 00 00
*DATAM key (internal)	00 05 4D 00 03 00 00 00	00 05 4D 00 03 00 00 00
*DATAMV MAC verification key (external)	00 00 44 00 03 41 00 00	00 00 44 00 03 21 00 00
*DATAMV MAC verification key (internal)	00 05 44 00 03 00 00 00	00 05 44 00 03 00 00 00
*DATAXLAT	00 06 71 00 03 00 00 00	
DECIPHER	00 03 50 00 03 00 00 00	
DECIPHER (double-length)	00 03 50 00 03 41 00 00	00 03 50 00 03 21 00 00
DKYGENKY	00 71 44 00 03 41 00 00	00 71 44 00 03 21 00 00
ENCIPHER	00 03 60 00 03 00 00 00	
ENCIPHER (double-length)	00 03 60 00 03 41 00 00	00 03 60 00 03 21 00 00
*EXPORTER	00 41 7D 00 03 41 00 00	00 41 7D 00 03 21 00 00
IKEYXLAT	00 42 42 00 03 41 00 00	00 42 42 00 03 21 00 00
*IMP-PKA	00 42 05 00 03 41 00 00	00 42 05 00 03 21 00 00
*IMPORTER	00 42 7D 00 03 41 00 00	00 42 7D 00 03 21 00 00
*IPINENC	00 21 5F 00 03 41 00 00	00 21 5F 00 03 21 00 00
*MAC	00 05 4D 00 03 00 00 00	
MAC (double-length)	00 05 4D 00 03 41 00 00	00 05 4D 00 03 21 00 00
*MACVER	00 05 44 00 03 00 00 00	
MACVER (double-length)	00 05 44 00 03 41 00 00	00 05 44 00 03 21 00 00
OKEYXLAT	00 41 42 00 03 41 00 00	00 41 42 00 03 21 00 00
*OPINENC	00 24 77 00 03 41 00 00	00 24 77 00 03 21 00 00
*PINGEN	00 22 7E 00 03 41 00 00	00 22 7E 00 03 21 00 00
*PINVER	00 22 42 00 03 41 00 00	00 22 42 00 03 21 00 00

**Note:** The external control vectors for DATAC, DATAM MAC generation and DATAMV MAC verification keys are also referred to as data compatibility control vectors.

## Control-Vector Base Bits

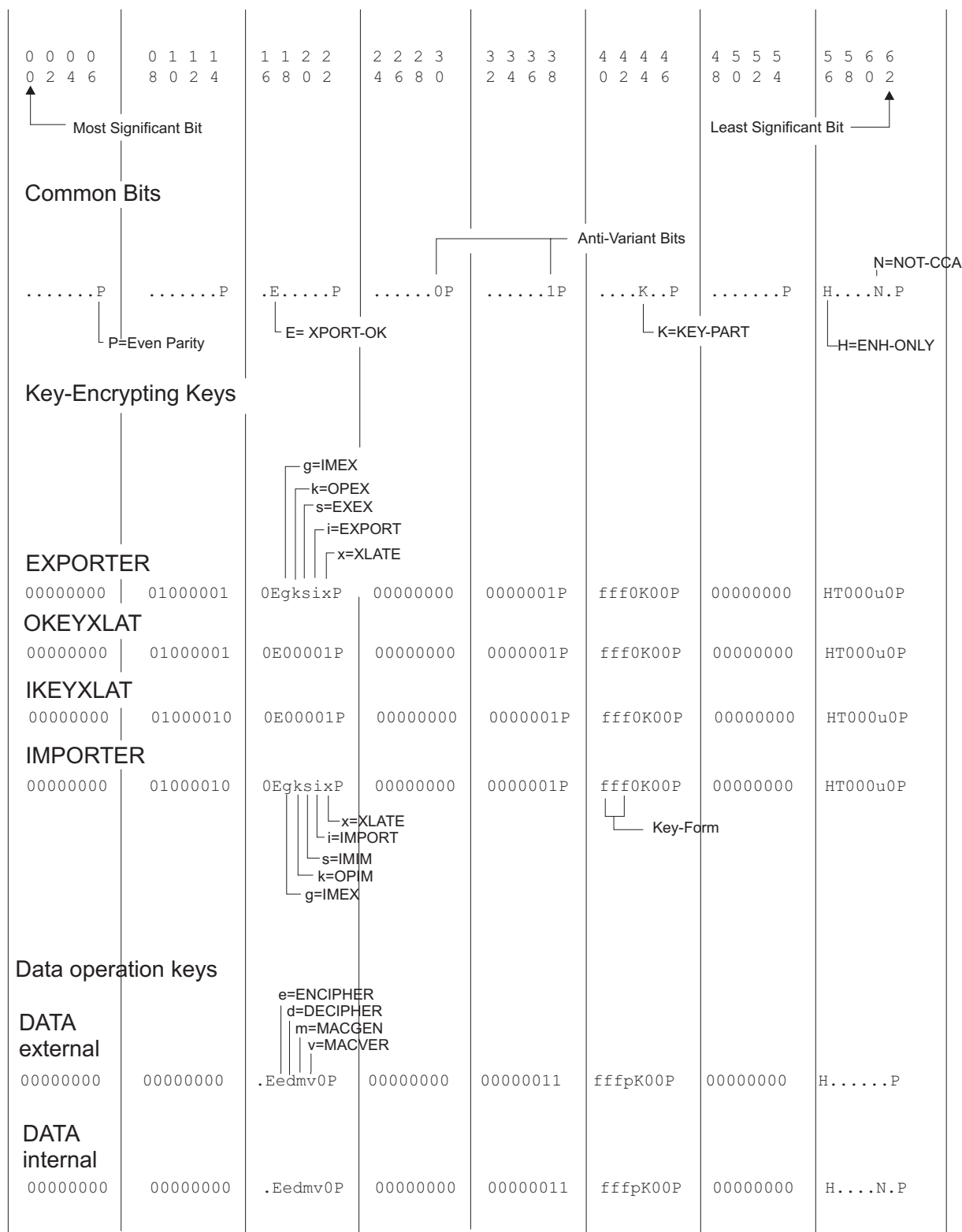


Figure 8. Control Vector Base Bit Map (Common Bits and Key-Encrypting Keys)

## Control-Vector Base Bits

0 0 0 0 0 2 4 6	0 1 1 1 8 0 2 4	1 1 2 2 6 8 0 2	2 2 2 3 4 6 8 0	3 3 3 3 2 4 6 8	4 4 4 4 0 2 4 6	4 5 5 5 8 0 2 4	5 5 6 6 6 8 0 2
Most Significant Bit			Least Significant Bit				
Data Operation Keys							
		e=ENCIPHER d=DECIPHER m=MACGEN v=MACVER					
DATA							
00000000	00000000	0Eedmv0P	00000000	00000011	fff0K00P	00000000	HT000u0P
DATA C							
00000000	00000000	0E11000P	00000000	00000011	fff0K00P	00000000	HT000u0P
DATAM							
00000000	00000000	0E00110P	00000000	00000011	fff0K00P	00000000	HT000u0P
DATAMV							
00000000	00000000	0E00010P	00000000	00000011	fff0K00P	00000000	HT000u0P
CIPHER							
00000000	00000011	0E11000P	00000000	00000011	fff0K00P	00000000	HT000u0P
CIPHERXI							
0000uu00	00001100	0E010001	00000000	00000011	fff0K00P	00000000	HT000u0P
CIPHERXO							
0000uu00	00001100	0E100001	00000000	00000011	fff0K00P	00000000	HT000u0P
CIPHERXI							
0000uu00	00001100	0E110001	00000000	00000011	fff0K00P	00000000	HT000u0P
		XDout Xdin					
DECIPHER							
00000000	00000011	0E01000P	00000000	00000011	fff0K00P	00000000	HT000u0P
ENCIPHER							
00000000	00000011	0E10000P	00000000	00000011	fff0K00P	00000000	HT000u0P
SECMSG							
00000000	00001010	0E...000P	00000000	00000011	fff0K00P	00000000	HT000u0P
		01 PIN encryption 10 Key encryption					
MAC							
cccc0000	00000101	0E00110P	00000000	00000011	fff0K00P	00000000	HT000u0P
MACVER							
cccc0000	00000101	0E00010P	00000000	00000011	fff0K00P	00000000	HT000u0P
0000 ANY 0001 ANSI X9.9 0010 CVV KEY-A 0011 CVV KEY-B 0100 AMEX-CSC				Key-Form			

Figure 9. Control Vector Base Bit Map (Data Operation Keys)

## Control-Vector Base Bits

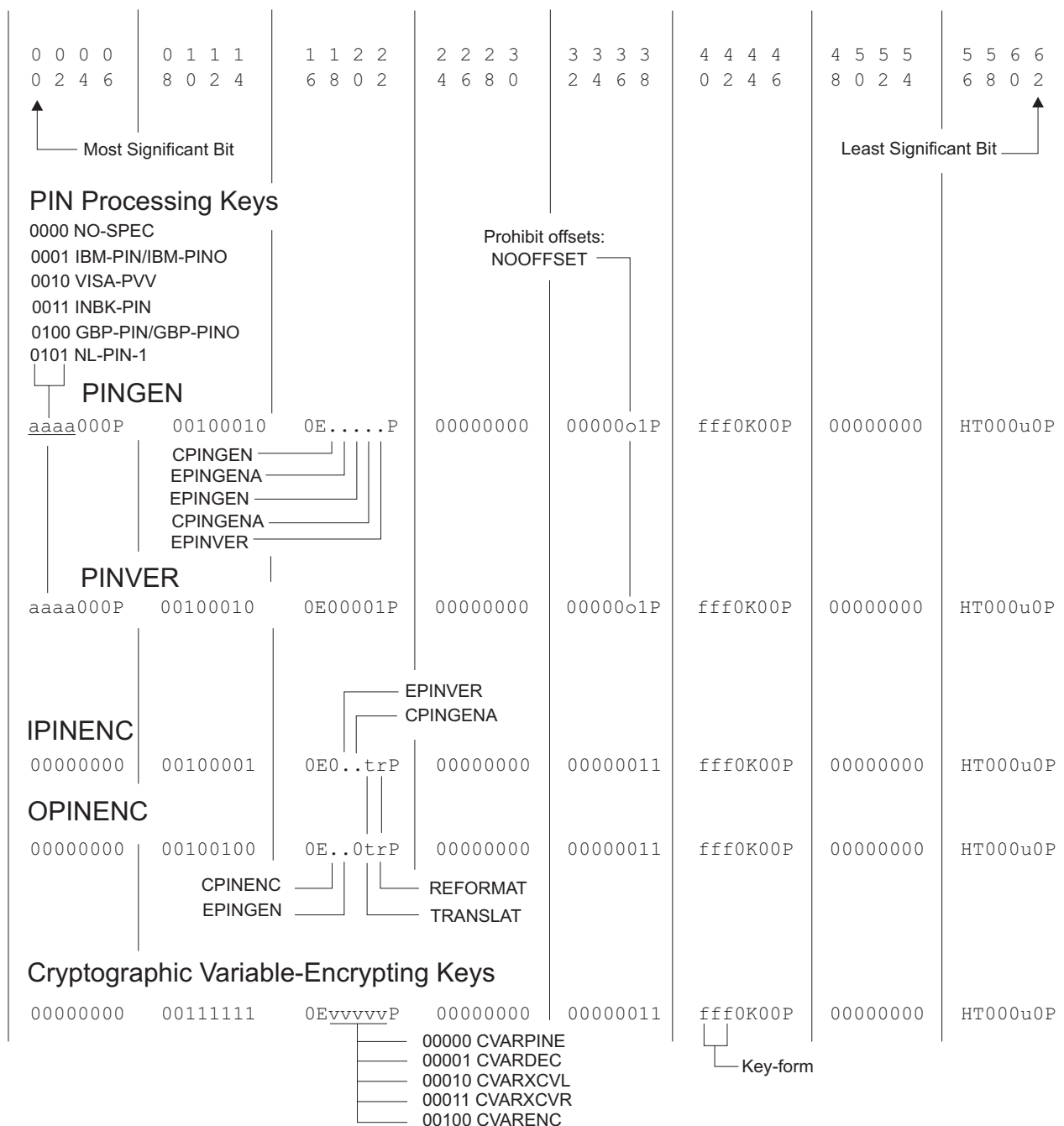


Figure 10. Control Vector Base Bit Map (PIN Processing Keys and Cryptographic Variable-Encrypting Keys)

## Control-Vector Base Bits

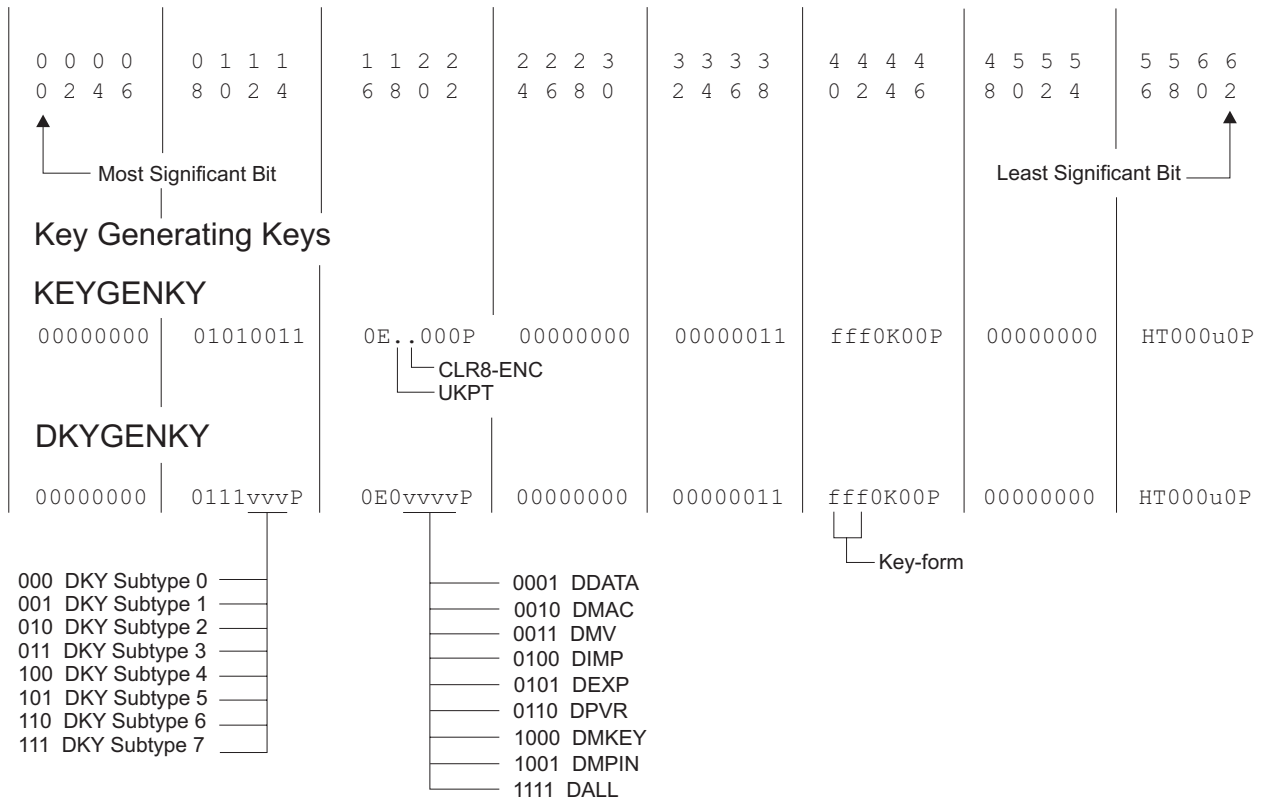


Figure 11. Control Vector Base Bit Map (Key Generating Keys)

**Key Form Bits, 'fff' -** The key form bits, 40-42, and for a double-length key, bits 104-106, are designated 'fff' in the preceding illustration. These bits can have these values:

000	Single length key
010	Double length key, left half
001	Double length key, right half

The following values may exist in some CCA implementations:

110	Double-length key, left half, halves guaranteed unique
101	Double-length key, right half, halves guaranteed unique

## Specifying a Control-Vector-Base Value

You can determine the value of a control vector by working through the following series of questions:

- Begin with a field of 64 bits (eight bytes) set to B'0'. The most significant bit is referred to as bit 0. Define the key type and subtype (bits 8 to 14), as follows:
  - The main key type bits (bits 8 to 11). Set bits 8 to 11 to one of the following values:

Bits 8 to 11	Main Key Type
0000	Data operation keys
0010	PIN keys
0011	Cryptographic variable-encrypting keys
0100	Key-encrypting keys
0101	Key-generating keys
0111	Diversified key-generating keys

- The key subtype bits (bits 12 to 14). Set bits 12 to 14 to one of the following values:

**Note:** For Diversified Key Generating Keys, the subtype field specifies the hierarchical level of the DKYGENKY. If the subtype is non-zero, then the DKYGENKY can only generate another DKYGENKY key with the hierarchy level decremented by one. If the subtype is zero, the DKYGENKY can only generate the final diversified key ( a non-DKYGENKY key) with the key type specified by the usage bits.

Bits 12 to 14	Key Subtype
<i>Data Operation Keys</i>	
000	Compatibility key (DATA)
001	Confidentiality key (CIPHER, DECIPHER, or ENCIPHER)
010	MAC key (MAC or MACVER)
011	Data translate key (DATAXLAT)
101	Secure messaging keys
110	Cipher text translate key (CIPHERXI, CIPHERXL, CIPHERXO)
<i>Key-Encrypting Keys</i>	
000	Transport-sending keys (EXPORTER and OKEYXLAT)
001	Transport-receiving keys (IMPORTER and IKEYXLAT)
<i>PIN Keys</i>	
001	PIN-generating key (PINGEN, PINVER)
000	Inbound PIN-block decrypting key (IPINENC)
010	Outbound PIN-block encrypting key (OPINENC)
<i>Cryptographic Variable-Encrypting Keys</i>	
111	Cryptographic variable-encrypting key (CVAR....)
<i>Diversified Key Generating Keys</i>	
000	DKY Subtype 0
001	DKY Subtype 1
010	DKY Subtype 2
011	DKY Subtype 3
100	DKY Subtype 4
101	DKY Subtype 5
110	DKY Subtype 6
111	DKY Subtype 7

2. For key-encrypting keys, set the following bits:
  - The key-generating usage bits (gks, bits 18 to 20). Set the gks bits to B'111' to indicate that the Key Generate callable service can use the associated key-encrypting key to encipher generated keys when the Key Generate callable service is generating various key-pair key-form combinations (see the Key-Encrypting Keys section of Figure 8). Without any of the gks bits set to 1, the Key Generate callable service cannot use the associated key-encrypting key. The Key Token Build callable service can set the gks bits to 1 when you supply the **OPIM**, **IMEX**, **IMIM**, **OPEX**, and **EXEX** keywords.
  - The IMPORT and EXPORT bit and the XLATE bit (ix, bits 21 and 22). If the 'i' bit is set to 1, the associated key-encrypting key can be used in the Data Key Import, Key Import, Data Key Export, and Key Export callable services. If the 'x' bit is set to 1, the associated key-encrypting key can be used in the Key Translate callable service.
  - The key-form bits (fff, bits 40 to 42). The key-form bits indicate how the key was generated and how the control vector participates in multiple-enciphering. To indicate that the parts can be the same value, set these bits to B'010'. For information about the value of the key-form bits in the right half of a control vector, see Step 8.
3. For MAC and MACVER keys, set the following bits:
  - The MAC control bits (bits 20 and 21). For a MAC-generate key, set bits 20 and 21 to B'11'. For a MAC-verify key, set bits 20 and 21 to B'01'.
  - The key-form bits (fff, bits 40 to 42). For a single-length key, set the bits to B'000'. For a double-length key, set the bits to B'010'.
4. For PINGEN and PINVER keys, set the following bits:
  - The PIN calculation method bits (aaaa, bits 0 to 3). Set these bits to one of the following values:

Bits 0 to 3	Calculation Method Keyword	Description
0000	NO-SPEC	A key with this control vector can be used with any PIN calculation method.
0001	IBM-PIN or IBM-PINO	A key with this control vector can be used only with the IBM PIN or PIN Offset calculation method.
0010	VISA-PVV	A key with this control vector can be used only with the VISA-PVV calculation method.
0100	GBP-PIN or GBP-PINO	A key with this control vector can be used only with the German Banking Pool PIN or PIN Offset calculation method.
0011	INBK-PIN	A key with this control vector can be used only with the Interbank PIN calculation method.
0101	NL-PIN-1	A key with this control vector can be used only with the NL-PIN-1, Netherlands PIN calculation method.



- The prohibit-offset bit (o, bit 37) to restrict operations to the PIN value. If set to 1, this bit prevents operation with the IBM 3624 PIN Offset calculation method and the IBM German Bank Pool PIN Offset calculation method.
5. For PINGEN, IPINENC, and OPINENC keys, set bits 18 to 22 to indicate whether the key can be used with the following callable services

Service Allowed	Bit Name	Bit
Clear PIN Generate	CPINGEN	18
Encrypted PIN Generate Alternate	EPINGENA	19
Encrypted PIN Generate	EPINGEN	20 for PINGEN 19 for OPINENC
Clear PIN Generate Alternate	CPINGENA	21 for PINGEN 20 for IPINENC
Encrypted Pin Verify	EPINVER	19
Clear PIN Encrypt	CPINENC	18

6. For the IPINENC (inbound) and OPINENC (outbound) PIN-block ciphering keys, do the following:
- Set the TRANSLAT bit (t, bit 21) to 1 to permit the key to be used in the PIN Translate callable service. The Control Vector Generate callable service can set the TRANSLAT bit to 1 when you supply the **TRANSLAT** keyword.
  - Set the REFORMAT bit (r, bit 22) to 1 to permit the key to be used in the PIN Translate callable service. The Control Vector Generate callable service can set the REFORMAT bit and the TRANSLAT bit to 1 when you supply the **REFORMAT** keyword.
7. For the cryptographic variable-encrypting keys (bits 18 to 22), set the variable-type bits (bits 18 to 22) to one of the following values:

Bits 18 to 22	Generic Key Type	Description
00000	CVARPINE	Used in the Encrypted PIN Generate Alternate service to encrypt a clear PIN.
00010	CVARXCVL	Used in the Control Vector Translate callable service to decrypt the left mask array.
00011	CVARXCVR	Used in the Control Vector Translate callable service to decrypt the right mask array.
00100	CVARENC	Used in the Cryptographic Variable Encipher callable service to encrypt an unformatted PIN.

8. For key-generating keys, set the following bits:
- For KEYGENKY, set bit 18 for UKPT usage and bit 19 for CLR8-ENC usage.
  - For DKYGENKY, bits 12–14 will specify the hierarchical level of the DKYGENKY key. If the subtype CV bits are non-zero, then the DKYGENKY can only generate another DKYGENKY key with the hierarchical level

decremented by one. If the subtype CV bits are zero, the DKYGENKY can only generate the final diversified key (a non-DKYGENKY key) with the key type specified by usage bits.

To specify the subtype values of the DKYGENKY, keywords DKYL0, DKYL1, DKYL2, DKYL3, DKYL4, DKYL5, DKYL6 and DKYL7 will be used.

- For DKYGENKY, bit 18 is reserved and must be zero.
- Usage bits 18-22 for the DKYGENKY key type are defined as follows. They will be encoded as the final key type that the DKYGENKY key generates.

Bits 19 to 22	Keyword	Usage
0001	DDATA	DATA, DATAC, single or double length
0010	DMAC	MAC, DATAM
0011	DMV	MACVER, DATAMV
0100	DIMP	IMPORTER, IKEYXLAT
0101	DEXP	EXPORTER, OKEYXLAT
0110	DPVR	PINVER
1000	DMKEY	Secure message key for encrypting keys
1001	DMPIN	Secure message key for encrypting PINs
1111	DALL	All key types may be generated except DKYGENKY and KEYGENKY keys. Usage of the DALL keyword is controlled by a separate access control point.

9. For secure messaging keys, set the following bits:

- Set bit 18 to 1 if the key will be used in the secure messaging for PINs service. Set bit 19 to 1 if the key will be used in the secure messaging for keys service.

10. For all keys, set the following bits:

- The export bit (E, bit 17). If set to 0, the export bit prevents a key from being exported. By setting this bit to 0, you can prevent the receiver of a key from exporting or translating the key for use in another cryptographic subsystem. Once this bit is set to 0, it cannot be set to 1 by any service other than Control Vector Translate. The Prohibit Export callable service can reset the export bit.
- The key-part bit (K, bit 44). Set the key-part bit to 1 in a control vector associated with a key part. When the final key part is combined with previously accumulated key parts, the key-part bit in the control vector for the final key part is set to 0. The Control Vector Generate callable service can set the key-part bit to 1 when you supply the **KEY-PART** keyword.
- The anti-variant bits (bit 30 and bit 38). Set bit 30 to 0 and bit 38 to 1. Many cryptographic systems have implemented a system of variants where a 7-bit value is exclusive-ORed with each 7-bit group of a key-encrypting key before enciphering the target key. By setting bits 30 and 38 to opposite values, control vectors do not produce patterns that can occur in variant-based systems.

- Control vector bits 64 to 127. If bits 40 to 42 are B'000' (single-length key), set bits 64 to 127 to 0. Otherwise, copy bits 0 to 63 into bits 64 to 127 and set bits 105 and 106 to B'01'.
- Set the parity bits (low-order bit of each byte, bits 7, 15, ..., 127). These bits contain the parity bits (P) of the control vector. Set the parity bit of each byte so the number of zero-value bits in the byte is an even number.
- For secure messaging keys, usage bit 18 on will enable the encryption of keys in a secure message and usage bit 19 on will enable the encryption of PINs in a secure message.
- The ENH-ONLY bit (H, bit 56). Set the ENH-ONLY bit to 1 in a control vector to require the key value be encrypted with the enhanced wrapping method. The Control Vector Generate callable service can set the ENH-ONLY bit to 1 when you supply the ENH-ONLY keyword.

---

## Changing Control Vectors with the Control Vector Translate Callable Service

Do the following when using the Control Vector Translate callable service:

- Provide the control information for testing the control vectors of the source, target, and key-encrypting keys to ensure that only sanctioned changes can be performed
- Select the key-half processing mode.

## Providing the Control Information for Testing the Control Vectors

To minimize your security exposure, the Control Vector Translate callable service requires control information (*mask array* information) to limit the range of allowable control vector changes. To ensure that this service is used only for authorized purposes, the source-key control vector, target-key control vector, and key-encrypting key (KEK) control vector must pass specific tests. The tests on the control vectors are performed within the secured cryptographic engine.

The tests consist of evaluating four logic expressions, the results of which must be a string of binary zeros. The expressions operate bitwise on information that is contained in the mask arrays and in the portions of the control vectors associated with the key or key-half that is being processed. If any of the expression evaluations do not result in all zero bits, the callable service is ended with a *control vector violation* return and reason code (8/39). See Figure 12. Only the 56 bit positions that are associated with a key value are evaluated. The low-order bit that is associated with key parity in each key byte is not evaluated.

## Mask Array Preparation

A mask array consists of seven 8-byte elements: A<sub>1</sub>, B<sub>1</sub>, A<sub>2</sub>, B<sub>2</sub>, A<sub>3</sub>, B<sub>3</sub>, and B<sub>4</sub>. You choose the values of the array elements such that each of the following four expressions evaluates to a string of binary zeros. (See Figure 12 on page 906.) Set the A bits to the value that you require for the corresponding control vector bits. In expressions 1 through 3, set the B bits to select the control vector bits to be evaluated. In expression 4, set the B bits to select the source and target control vector bits to be evaluated. Also, use the following control vector information:

C<sub>1</sub> is the control vector associated with the left half of the KEK.

C<sub>2</sub> is the control vector associated with the source key, or selected source-key half/halves.

$C_3$  is the control vector associated with the target key or selected target-key half/halves.

1. ( $C_1$  exclusive-OR  $A_1$ ) logical-AND  $B_1$

This expression tests whether the KEK used to encipher the key meets your criteria for the desired translation.

2. ( $C_2$  exclusive-OR  $A_2$ ) logical-AND  $B_2$

This expression tests whether the control vector associated with the source key meets your criteria for the desired translation.

3. ( $C_3$  exclusive-OR  $A_3$ ) logical-AND  $B_3$

This expression tests whether the control vector associated with the target key meets your criteria for the desired translation.

4. ( $C_2$  exclusive-OR  $C_3$ ) logical-AND  $B_4$

This expression tests whether the control vectors associated with the source key and the target key meet your criteria for the desired translation.

Encipher two copies of the mask array, each under a different cryptographic-variable key (key type CVARENC). To encipher each copy of the mask array, use the Cryptographic Variable Encipher callable service. Use two different keys so that the enciphered-array copies are unique values. When using the Control Vector Translate callable service, the *mask\_array\_left* parameter and the *mask\_array\_right* parameter identify the enciphered mask arrays. The *array\_key\_left* parameter and the *array\_key\_right* parameter identify the internal keys for deciphering the mask arrays. The *array\_key\_left* key must have a key type of CVARXCVL and the *array\_key\_right* key must have a key type of CVARXCVR. The cryptographic process decipheres the arrays and compares the results; for the service to continue, the deciphered arrays must be equal. If the results are not equal, the service returns the return and reason code for data that is not valid (8/385).

Use the Key Generate callable service to create the key pairs CVARENC-CVARXCVL and CVARENC-CVARXCVR. Each key in the key pair must be generated for a different node. The CVARENC keys are generated for, or imported into, the node where the mask array will be enciphered. After enciphering the mask array, you should destroy the enciphering key. The CVARXCVL and CVARXCVR keys are generated for, or imported into, the node where the Control Vector Translate callable service will be performed.

If using the **BOTH** keyword to process both halves of a double-length key, remember that bits 41, 42, 104, and 105 are different in the left and right halves of the CCA control vector and must be ignored in your mask-array tests (that is, make the corresponding  $B_2$  and/or  $B_3$  bits equal to zero).

When the control vectors pass the masking tests, the verb does the following:

- Deciphers the source key. In the decipher process, the service uses a key that is formed by the exclusive-OR of the KEK and the control vector in the key token variable the *source\_key\_token* parameter identifies.
- Enciphers the deciphered source key. In the encipher process, the service uses a key that is formed by the exclusive-OR of the KEK and the control vector in the key token variable the *target\_key\_token* parameter identifies.
- Places the enciphered key in the key field in the key token variable the *target\_key\_token* parameter identifies.

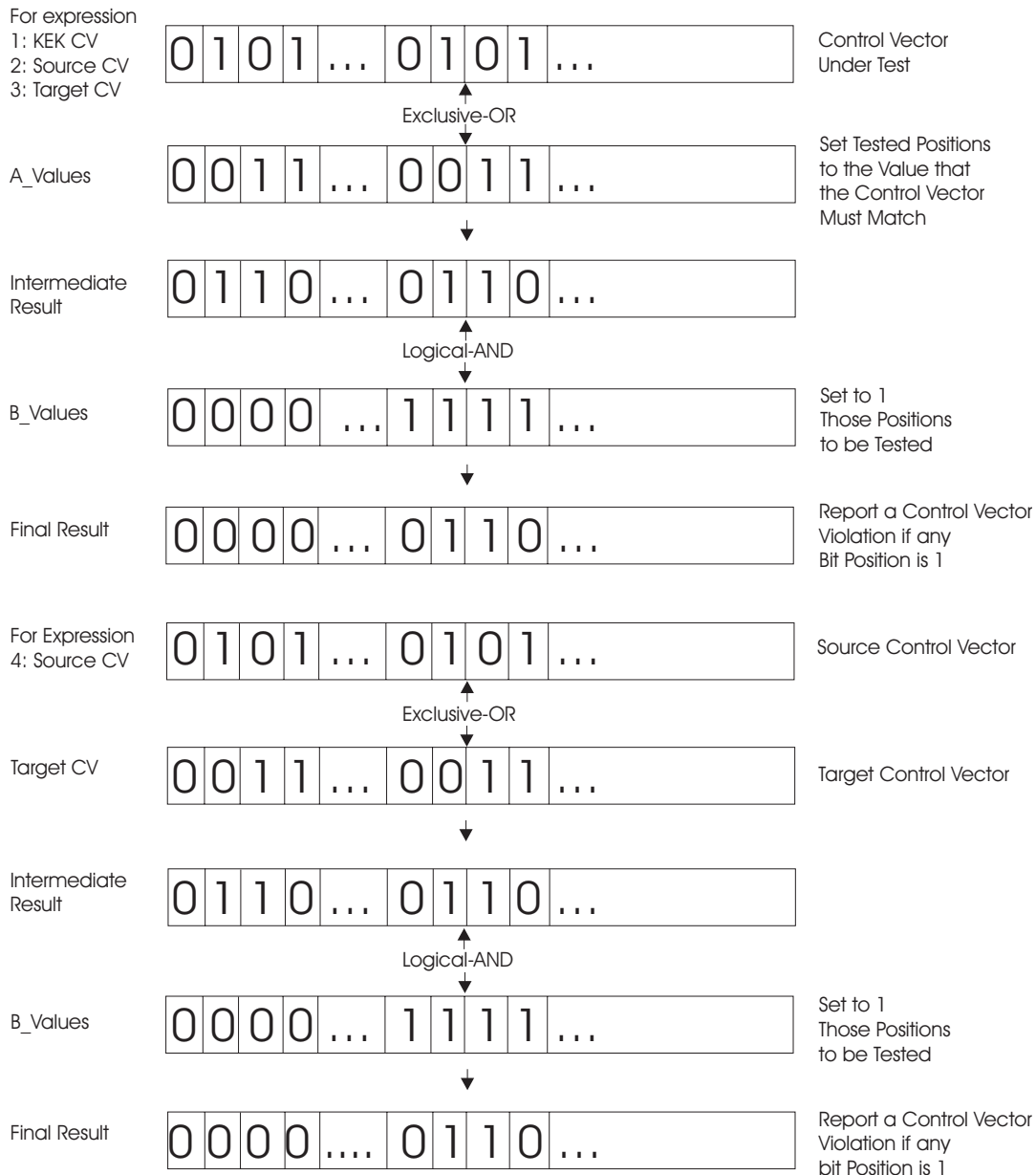
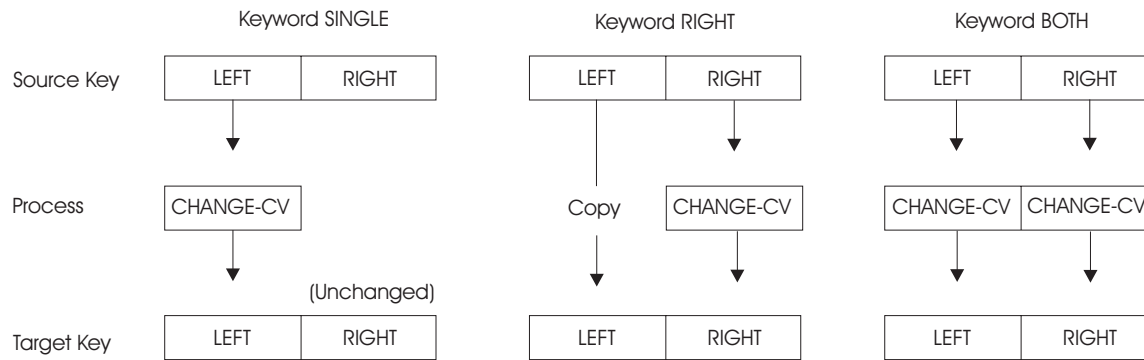


Figure 12. Control Vector Translate Callable Service Mask\_Array Processing

## Selecting the Key-Half Processing Mode

Use the Control Vector Translate callable service to change a control vector associated with a key. Rule-array keywords determine which key halves are processed in the call, as shown in Figure 13 on page 907.



**Figure 13. Control Vector Translate Callable Service.** In this figure, CHANGE-CV means the requested control vector translation change; LEFT and RIGHT mean the left and right halves of a key and its control vector.

Keyword	Meaning
<b>SINGLE</b>	<p>This keyword causes the control vector of the left half of the source key to be changed. The updated key half is placed into the left half of the target key in the target key token. The right half of the target key is unchanged.</p> <p>The <b>SINGLE</b> keyword is useful when processing a single-length key, or when first processing the left half of a double-length key (to be followed by processing the right half).</p>
<b>RIGHT</b>	<p>This keyword causes the control vector of the right half of the source key to be changed. The updated key half is placed into the right half of the target key of the target key token. The left half of the source key is copied unchanged into the left half of the target key in the target key token.</p>
<b>BOTH</b>	<p>This keyword causes the control vector of both halves of the source key to be changed. The updated key is placed into the target key in the target key token.</p> <p>A single set of control information must permit the control vector changes applied to each key half. Normally, control vector bit positions 41, 42, 105, and 106 are different for each key half. Therefore, set bits 41 and 42 to B'00' in mask array elements B<sub>1</sub>, B<sub>2</sub>, and B<sub>3</sub>.</p> <p>You can verify that the source and target key tokens have control vectors with matching bits in bit positions 40-42 and 104-106, the "form field" bits. Ensure that bits 40-42 of mask array B<sub>4</sub> are set to B'111'.</p>
<b>LEFT</b>	<p>This keyword enables you to supply a single-length key and obtain a double-length key. The source key token must contain:</p> <ul style="list-style-type: none"> <li>• The KEK-enciphered single-length key</li> <li>• The control vector for the single-length key (often this is a null value)</li> <li>• A control vector, stored in the source token where the right-half control vector is normally stored, used in decrypting the single-length source key when the key is being processed for the target right half of the key.</li> </ul> <p>The service first processes the source and target tokens as with the <b>SINGLE</b> keyword. Then the source token is processed using the single-length enciphered key and the source token right-half</p>

control vector to obtain the actual key value. The key value is then enciphered using the KEK and the control vector in the target token for the right-half of the key.

This approach is frequently of use when you must obtain a double-length CCA key from a system that only supports a single-length key, for example when processing PIN keys or key-encrypting keys received from non-CCA systems.

To prevent the service from ensuring that each key byte has odd parity, you can specify the **NOADJUST** keyword. If you do not specify the **NOADJUST** keyword, or if you specify the **ADJUST** keyword, the service ensures that each byte of the target key has odd parity.

## When the Target Key Token CV Is Null

When you use any of the **LEFT**, **BOTH**, or **RIGHT** keywords, and when the control vector in the target key token is null (all B'0'), then bit 3 in byte 59 will be set to B'1' to indicate that this is a double-length DATA key.

## Control Vector Translate Example

As an example, consider the case of receiving a single-length PIN-block encrypting key from a non-CCA system. Often such a key will be encrypted by an unmodified transport key (no control vector or variant is used). In a CCA system, an inbound PIN encrypting key is double-length.

First use the Key Token Build callable service to insert the single-length key value into the left-half key-space in a key token. Specify **USE-CV** as a key type and a control vector value set to 16 bytes of X'00'. Also specify **EXTERNAL**, **KEY**, and **CV** keywords in the rule array. This key token will be the source key key token.

Second, the target key token can also be created using the Key Token Build callable service. Specify a key type of **IPINENC** and the **NO-EXPORT** rule array keyword.

Then call the Control Vector Translate callable service and specify a rule-array keyword of **LEFT**. The mask arrays can be constructed as follows:

- $A_1$  is set to the value of the KEK's control vector, most likely the value of an **IMPORTER** key, perhaps with the **NO-EXPORT** bit set.  $B_1$  is set to eight bytes of X'FF' so that all bits of the KEK's control vector will be tested.
- $A_2$  is set to eight bytes of X'00', the (null) value of the source key control vector.  $B_2$  is set to eight bytes of X'FF' so that all bits of the source-key "control vector" will be tested.
- $A_3$  is set to the value of the target key's left-half control vector.  $B_3$  is set to X'FFFF FFFF FF9F FFFF'. This will cause all bits of the control vector to be tested except for the two ("fff") bits used to distinguish between the left-half and right-half target-key control vector.
- $B_4$  is set to eight bytes of X'00' so that no comparison is made between the source and target control vectors.



---

## Appendix D. Coding Examples

This appendix provides sample routines using the ICSF callable services for these languages:

- C
- COBOL
- Assembler
- PL/1

The C, COBOL and Assembler H examples that follow use the key generate, encipher, and decipher callable services to determine whether the deciphered text matches the starting text.

---

### C

C programs must include the header file csfbext.h, which contains stubs for calling the ICSF services. This file is installed in the HFS directory /usr/include and is copied to SYS1.SIEAHDR.H(CSFBEXT).

Information on creating C applications that call ICSF PKCS #11 services is available in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

In addition, C applications that include csfbext.h must be link edited with the appropriate DLL sidedeck for the addressing model:

#### Standard 31-bit

Link with /usr/lib/CSFDLL31.x or SYS1.SIEASID(CSFDLL31)

#### 31-bit with XPLINK

Link with /usr/lib/CSFDLL3X.x or SYS1.SIEASID(CSFDLL3X)

**64-bit** Link with /usr/lib/CSFDLL64.x or SYS1.SIEASID(CSFDLL64)

Information on creating C applications that call ICSF PKCS #11 services is available in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

```
/*-----*
 * Example using C:                                     *
 *   Invokes CSNBKGN (key generate), CSNBENC (DES encipher) and *
 *   CSNBDEC (DES decipher)                               *
 *-----*/
#include <stdio.h>
#include "csfbext.h"

/*-----*
 * Prototypes for functions in this example               *
 *-----*/

/*-----*
 * Utility for printing hex strings                       *
 *-----*/
void printHex(unsigned char *, unsigned int);

/*****
 * Main Function                                          */
/*****
int main(void) {

    /*-----*/
```



```

* Constant inputs to ICSF services                                     *
*-----*/
static int textLen = 24;
static unsigned char clearText[24]="ABCDEFGH IJKLMN0987654321";
static unsigned char cipherProcessRule[8]="CUSP    ";
static unsigned char keyForm[4]="OP  ";
static unsigned char keyLength[8]="SINGLE  ";
static unsigned char dataKeyType[8]="DATA  ";
static unsigned char nullKeyType[8]="      ";
static unsigned char ICV[8]={0};
static int *pad=0;
static int exitDataLength = 0;
static unsigned char exitData[4]={0};
static int ruleArrayCount = 1;

/*-----*
* Variable inputs/outputs for ICSF services                          *
*-----*/
unsigned char cipherText[24]={0};
unsigned char compareText[24]={0};
unsigned char dataKeyId[64]={0};
unsigned char nullKeyId[64]={0};
unsigned char dummyKEKKeyId1[64]={0};
unsigned char dummyKEKKeyId2[64]={0};
int returnCode = 0;
int reasonCode = 0;
unsigned char OCV[18]={0};

/*-----*
* Begin executable code                                             *
*-----*/
do {
/*-----*
* Call key generate                                                 *
*-----*/
if ((returnCode = CSNBKGN(&returnCode,
                        &reasonCode,
                        &exitDataLength,
                        exitData,
                        keyForm,
                        keyLength,
                        dataKeyType,
                        nullKeyType,
                        dummyKEKKeyId1,
                        dummyKEKKeyId2,
                        dataKeyId,
                        nullKeyId)) != 0) {
    printf("\nKey Generate failed:\n");
    printf("    Return Code = %04d\n",returnCode);
    printf("    Reason Code = %04d\n",reasonCode);
    break;
}

/*-----*
* Call encipher                                                     *
*-----*/
printf("\nClear Text\n");
printHex(clearText,sizeof(clearText));

if ((returnCode = CSNBENC(&returnCode,
                        &reasonCode,
                        &exitDataLength,
                        exitData,
                        dataKeyId,
                        &textLen,
                        clearText,
                        ICV,
                        &ruleArrayCount,

```

```

                                cipherProcessRule,
                                &pad,
                                OCV,
                                cipherText)) != 0) {
    printf("\nReturn from Encipher:\n");
    printf("    Return Code = %04d\n",returnCode);
    printf("    Reason Code = %04d\n",reasonCode);
    if (returnCode > 4)
        break;
    }

/*-----*
 * Call decipher                                *
 *-----*/
printf("\nCipher Text\n");
printHex(cipherText,sizeof(cipherText));

if ((returnCode = CSNBDEC(&returnCode,
                        &reasonCode,
                        &exitDataLength,
                        exitData,
                        dataKeyId,
                        &textLen,
                        cipherText,
                        ICV,
                        &ruleArrayCount,
                        cipherProcessRule,
                        OCV,
                        compareText)) != 0) {
    printf("\nReturn from Decipher:\n");
    printf("    Return Code = %04d\n",returnCode);
    printf("    Reason Code = %04d\n",reasonCode);
    if (returnCode > 4)
        break;
    }

/*-----*
 * End                                          *
 *-----*/
printf("\nClear Text after decipher\n");
printHex(compareText,sizeof(compareText));

} while(0);

return returnCode;

} /* end main */

void printHex (unsigned char * text, unsigned int len)
/*-----*
 * Prints a string as hex characters          *
 *-----*/
{
    unsigned int i;

    for (i = 0; i < len; ++i)
        if ( ((i & 7) == 7) || (i == (len - 1)) )
            printf (" %02x\n", text[i]);
        else
            printf (" %02x", text[i]);
    printf ("\n");
} /* end printHex */

```

# COBOL

```
*****
IDENTIFICATION DIVISION.
*****
PROGRAM-ID. COBOLXMP.
*****
ENVIRONMENT DIVISION.
*****
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
*****
DATA DIVISION.
*****
FILE SECTION.
WORKING-STORAGE SECTION.
77 INPUT-TEXT PIC X(24)
   VALUE 'ABCDEFGHIJKLMN0987654321'.
77 OUTPUT-TEXT PIC X(24)
   VALUE LOW-VALUES.
77 COMPARE-TEXT PIC X(24)
   VALUE LOW-VALUES.
77 CIPHER-PROCESSING-RULE PIC X(08)
   VALUE 'CUSP '.
77 KEY-FORM PIC X(08)
   VALUE 'OP '.
77 KEY-LENGTH PIC X(08)
   VALUE 'SINGLE '.
77 KEY-TYPE-1 PIC X(08)
   VALUE 'DATA '.
77 KEY-TYPE-2 PIC X(08)
   VALUE ' '.
77 ICV PIC X(08)
   VALUE LOW-VALUES.
77 PAD PIC X(01)
   VALUE LOW-VALUES.
*****
DEFINE SAPI INPUT/OUTPUT PARAMETERS *****
01 SAPI-REC.
   05 RETURN-CODE-S PIC 9(08) COMP.
   05 REASON-CODE-S PIC 9(08) COMP.
   05 EXIT-DATA-LENGTH-S PIC 9(08) COMP.
   05 EXIT-DATA-S PIC X(04).
   05 KEK-KEY-ID-1-S PIC X(64)
      VALUE LOW-VALUES.
   05 KEK-KEY-ID-2-S PIC X(64)
      VALUE LOW-VALUES.
   05 DATA-KEY-ID-S PIC X(64)
      VALUE LOW-VALUES.
   05 NULL-KEY-ID-S PIC X(64)
      VALUE LOW-VALUES.
   05 KEY-FORM-S PIC X(08).
   05 KEY-LENGTH-S PIC X(08).
   05 DATA-KEY-TYPE-S PIC X(08).
   05 NULL-KEY-TYPE-S PIC X(08).
   05 TEXT-LENGTH-S PIC 9(08) COMP.
   05 TEXT-S PIC X(24).
   05 ICV-S PIC X(08).
   05 PAD-S PIC X(01).
   05 CPHR-TEXT-S PIC X(24).
   05 COMP-TEXT-S PIC X(24).
   05 RULE-ARRAY-COUNT-S PIC 9(08) COMP.
   05 RULE-ARRAY-S.
      10 RULE-ARRAY PIC X(08).
   05 CHAINING-VECTOR-S PIC X(18).
*****
PROCEDURE DIVISION.
```

```

*****
MAIN-RTN.
***** CALL KEY GENERATE *****
MOVE 0 TO EXIT-DATA-LENGTH-S.
MOVE KEY-FORM TO KEY-FORM-S.
MOVE KEY-LENGTH TO KEY-LENGTH-S.
MOVE KEY-TYPE-1 TO DATA-KEY-TYPE-S.
MOVE KEY-TYPE-2 TO NULL-KEY-TYPE-S.
CALL 'CSNBKGN' USING RETURN-CODE-S
REASON-CODE-S
EXIT-DATA-LENGTH-S
EXIT-DATA-S
KEY-FORM-S
KEY-LENGTH-S
DATA-KEY-TYPE-S
NULL-KEY-TYPE-S
KEK-KEY-ID-1-S
KEK-KEY-ID-2-S
DATA-KEY-ID-S
NULL-KEY-ID-S.

IF RETURN-CODE-S NOT = 0 OR
REASON-CODE-S NOT = 0 THEN
DISPLAY '*** KEY-GENERATE ***'
DISPLAY '*** RETURN-CODE = ' RETURN-CODE-S
DISPLAY '*** REASON-CODE = ' REASON-CODE-S
ELSE
MOVE 24 TO TEXT-LENGTH-S
MOVE INPUT-TEXT TO TEXT-S
MOVE 1 TO RULE-ARRAY-COUNT-S
MOVE CIPHER-PROCESSING-RULE TO RULE-ARRAY-S
MOVE LOW-VALUES TO CHAINING-VECTOR-S
MOVE ICV TO ICV-S.
MOVE PAD TO PAD-S.
***** CALL ENCIPHER *****
CALL 'CSNBENC' USING RETURN-CODE-S
REASON-CODE-S
EXIT-DATA-LENGTH-S
EXIT-DATA-S
DATA-KEY-ID-S
TEXT-LENGTH-S
TEXT-S
ICV-S
RULE-ARRAY-COUNT-S
RULE-ARRAY-S
PAD-S
CHAINING-VECTOR-S
CPHR-TEXT-S

IF RETURN-CODE-S NOT = 0 OR
REASON-CODE-S NOT = 0 THEN
DISPLAY '*** ENCIPHER ***'
DISPLAY '*** RETURN-CODE = ' RETURN-CODE-S
DISPLAY '*** REASON-CODE = ' REASON-CODE-S
ELSE
***** CALL DECIPHER *****
CALL 'CSNBDEC' USING RETURN-CODE-S
REASON-CODE-S
EXIT-DATA-LENGTH-S
EXIT-DATA-S
DATA-KEY-ID-S
TEXT-LENGTH-S
CPHR-TEXT-S
ICV-S
RULE-ARRAY-COUNT-S
RULE-ARRAY-S
CHAINING-VECTOR-S
COMP-TEXT-S

IF RETURN-CODE-S NOT = 0 OR

```

```

        REASON-CODE-S NOT = 0 THEN
        DISPLAY '*** DECIPHER ***'
        DISPLAY '*** RETURN-CODE = ' RETURN-CODE-S
        DISPLAY '*** REASON-CODE = ' REASON-CODE-S
    ELSE
        IF COMP-TEXT-S = TEXT-S THEN
            DISPLAY '*** DECIPHERED TEXT = PLAIN TEXT ***'
        ELSE
            DISPLAY '*** DECIPHERED TEXT @= PLAIN TEXT ***'.
    DISPLAY '*** TEST PROGRAM ENDED ***'
    STOP RUN.

```

---

## Assembler H

```

        TITLE 'SAMPLE ENCIPHER/DECIPHER S/370 PROGRAM.'
*=====
*      SYSTEM/370 ASSEMBLER H EXAMPLE
*
*=====
SAMPLE  SPACE
        START 0
        DS     0H
        STM    14,12,12(13)    SAVE REGISTERS
        BALR   12,0            USE R12 AS BASE REGISTER
        USING  *,12            PROVIDE SAVE AREA FOR SUBROUTINE
        LA     14,SAVE         PERFORM SAVE AREA CHAINING
        ST     13,4(14)        "
        ST     14,8(13)        "
        LR     13,14           "
*
        CALL   CSFKGN,(RETC,   *
        RESCD, *
        EXDAT,  *
        EXDATA, *
        KEY_FORM, *
        KEY_LEN, *
        KEYTYP1, *
        KEYTYP2, *
        KEK_ID1, *
        KEK_ID2, *
        DATA_ID, *
        NULL_ID) *
        CLC    RETC,=F'0'      CHECK RETURN CODE
        BNE    BACK           OUTPUT RETURN/REASON CODE AND STOP
        CLC    RESC,=F'0'      CHECK REASON CODE
        BNE    BACK           OUTPUT RETURN/REASON CODE AND STOP
*
* CALL ENCIPHER WITH THE KEY JUST GENERATED
* OPERATIONAL FORM
*
        MVC    RULEAC,=F'1'    SET RULE ARRAY COUNT
        MVC    RULEA,=CL8'CUSP ' BUILD RULE ARRAY
        CALL   CSFENC,(RETC,   *
        RESCD, *
        EXDAT,  *
        EXDATA, *
        DATA_ID, *
        TEXTL,  *
        TEXT,   *
        ICV,    *
        RULEAC, *
        RULEA,  *
        PAD_CHAR, *
        OCV,    *
        CIPHER_TEXT) *
        CLC    RETC,=F'0'      CHECK RETURN CODE
        BNE    BACK           OUTPUT RETURN/REASON CODE AND STOP

```

```

CLC  RESCD,=F'0'      CHECK REASON CODE
BNE  BACK             OUTPUT RETURN/REASON CODE AND STOP
CALL CSFDEC,(RETCD,
                RESCD,
                EXDATA,
                EXDATA,
                DATA_ID,
                TEXTL,
                CIPHER_TEXT,
                ICV,
                RULEAC,
                RULEA,
                OCV,
                NEW_TEXT)
CLC  RETCD,=F'0'      CHECK RETURN CODE
BNE  BACK             OUTPUT RETURN/REASON CODE AND STOP
CLC  RESCD,=F'0'      CHECK REASON CODE
BNE  BACK             OUTPUT RETURN/REASON CODE AND STOP
*
COMPARE EQU  *                COMPARE START AND END TEXT
CLC  TEXT,NEW_TEXT
BE   GOODENC
WTO  'DECIPHERED TEXT DOES NOT MATCH STARTING TEXT'
B    BACK
GOODENC WTO  'DECIPHERED TEXT MATCHES STARTING TEXT'
*
*
WTO  'TEST PROGRAM TERMINATING'
B    RETURN
*
*-----
* CONVERT RETURN/REASON CODES FROM BINARY TO EBCDIC
*-----
BACK  DS  0F                OUTPUT RETURN & REASON CODE
      L   5,RETCD           LOAD RETURN CODE
      L   6,RESCD           LOAD REASON CODE
      CVD 5,BCD1            CONVERT TO PACK-DECIMAL
      CVD 6,BCD2
      UNPK ORETCD,BCD1      CONVERT TO EBCDIC
      UNPK ORESCD,BCD2
      OI  ORETCD+7,X'F0'    CORRECT LAST DIGIT
      OI  ORESCD+7,X'F0'
*
      MVC ERROUT+21(4),ORETCD+4
      MVC ERROUT+41(4),ORESCD+4
ERROUT WTO  'ERROR CODE =      , REASON CODE =      '
RETURN EQU  *
      L   13,4(13)         SAVE AREA RESTORATION
      MVC 16(4,13),RETCD    SAVE RETURN CODE
      LM  14,12,12(13)
      BR  14               RETURN TO CALLER
*
BCD1  DS  D                CONVERT TO BCD TEMP AREA
BCD2  DS  D                CONVERT TO BCD TEMP AREA
ORETCD DS  CL8'0'          OUTPUT RETURN CODE
ORESCD DS  CL8'0'          OUTPUT REASON CODE
*
KEY_FORM DC  CL8'OP      '  KEY FORM
KEY_LEN  DC  CL8'SINGLE  '  KEY LENGTH
KEYTYP1  DC  CL8'DATA   '  KEY TYPE 1
KEYTYP2  DC  CL8'      '  KEY TYPE 2
TEXT     DC  C'ABCDEFGHJKLMNOPQRSTUVWXYZ0987654321'
TEXTL    DC  F'32'      TEXT LENGTH
CIPHER_TEXT DC CL32' '
NEW_TEXT DC  CL32' '
DATA_ID  DC  XL64'00'    DATA KEY TOKEN
NULL_ID  DC  XL64'00'    NULL KEY TOKEN - UNFILLED

```

KEK_ID1	DC	XL64'00'	KEK1 KEY TOKEN
KEK_ID2	DC	XL64'00'	KEK2 KEY TOKEN
RETC	DS	F'0'	RETURN CODE
RESCD	DS	F'0'	REASON CODE
EXDATA	DC	F'0'	EXIT DATA LENGTH
EXDATA	DS	0C	EXIT DATA
RULEA	DS	1CL8	RULE ARRAY
RULEAC	DS	F'0'	RULE ARRAY COUNT
ICV	DC	XL8'00'	INITIAL CHAINING VECTOR
OCV	DC	XL18'00'	OUTPUT CHAINING VECTOR
PAD_CHAR	DC	F'0'	PAD CHARACTER
SAVE	DS	18F	SAVE REGISTER AREA
	END	SAMPLE	

## PL/1

```

/*****
/*
/* Sample program to call the one-way hash service to generate
/* the SHA-1 hash of the input text and call digital signature
/* generate with an RSA key using the ISO 9796 text formatting. The
/* RSA key token is built from supplied data and imported for the
/* signature generate service to use.
/*
/* INPUT: TEXT Message digest to be signed
/*
/* OUTPUT: SIGNATURE_LENGTH Length of the signature in bytes
/* Written to a dataset.
/*
/* SIGNATURE Signature for hash. Written to a
/* dataset.
/*
/*
*****/
DSIGEXP:PROCEDURE( TEXT ) OPTIONS( MAIN );

/* Declarations - Parameters */

DCL TEXT CHAR( 64 ) VARYING;

/* Declarations - API parameters */

DCL CHAINING_VECTOR_LENGTH FIXED BINARY( 31, 0 ) INIT( 128 );
DCL CHAINING_VECTOR CHAR( 128 );
DCL DUMMY_KEY CHAR( 64 );
DCL EXIT_DATA CHAR( 4 );
DCL EXIT_LEN FIXED BINARY( 31, 0 ) INIT( 0 );

DCL HASH CHAR( 20 );
DCL HASH_LENGTH FIXED BINARY( 31, 0 ) INIT( 20 );

DCL INTERNAL_PKA_TOKEN CHAR( 1024 );
DCL INTERNAL_PKA_TOKEN_LENGTH FIXED BINARY( 31, 0 );

DCL KEY_VALUE_STRUCTURE CHAR(139)
INIT(( '02000040000300408000000000000000'X
'01AE28DA4606D885EB7E0340D6BAAC51'X
'991C0CD0EAE835AFD9CFF3CD7E7EA741'X
'41DADD24A6331BEDF41A6626522CCF15'X
'767D167D01A16F970100010252BDAD42'X
'52BDAD425A8C6045D41AFAF746BED5F'X
'085D574FCD9C07F0B38C2C45017C2A1A'X
'B919ED2551350A76606BFA6AF2F1609A'X
'00A0A48DD719A55E9CA801'X ));

DCL KEY_VALUE_LENGTH FIXED BINARY( 31, 0 ) INIT( 139 );

DCL OWH_TEXT CHAR( 64 );

```

```

DCL PKA_KEY_TOKEN          CHAR( 1024 );
DCL PKA_TOKEN_LENGTH      FIXED BINARY( 31, 0 );

DCL PRIVATE_NAME          CHAR( 64 ) INIT( 'PL1.EXAMPLE.FOR.APG' );
DCL PRIVATE_NAME_LENGTH   FIXED BINARY( 31, 0 ) INIT( 0 );

DCL RETURN_CODE           FIXED BINARY( 31, 0 ) INIT( 0 );
DCL REASON_CODE           FIXED BINARY( 31, 0 ) INIT( 0 );

DCL RESERVED_FIELD_LENGTH FIXED BINARY( 31, 0 ) INIT( 0 );
DCL RESERVED_FIELD        CHAR( 1 );

DCL RULE_ARY_CNT_DSG       FIXED BINARY( 31, 0 ) INIT( 1 );
DCL RULE_ARY_CNT_PKB       FIXED BINARY( 31, 0 ) INIT( 1 );
DCL RULE_ARY_CNT_PKI       FIXED BINARY( 31, 0 ) INIT( 0 );
DCL RULE_ARY_CNT_OWH       FIXED BINARY( 31, 0 ) INIT( 2 );
DCL RULE_ARY_DSG           CHAR( 8 ) INIT( 'ISO-9796' );
DCL RULE_ARY_PKB           CHAR( 8 ) INIT( 'RSA-PRIV' );
DCL RULE_ARY_PKI           CHAR( 8 );
DCL RULE_ARY_OWH           CHAR( 16 ) INIT( 'SHA-1 ONLY ' );

DCL SIGNATURE_LENGTH       FIXED BINARY( 31, 0 );
DCL SIGNATURE              CHAR( 128 );
DCL SIG_BIT_LENGTH         FIXED BINARY( 31, 0 );

DCL TEXT_LENGTH            FIXED BINARY( 31, 0 );

/* Declarations - Files and entry points */

DCL SYSPRINT  FILE OUTPUT;
DCL SIGOUT    FILE RECORD OUTPUT;

DCL CSNDPKB   ENTRY EXTERNAL OPTIONS( ASM, INTER );
DCL CSNDPKI   ENTRY EXTERNAL OPTIONS( ASM, INTER );
DCL CSNBOWH   ENTRY EXTERNAL OPTIONS( ASM, INTER );
DCL CSNDDSG   ENTRY EXTERNAL OPTIONS( ASM, INTER );

/* Declarations - Internal variables */

DCL DSG_HEADER      CHAR( 32 )
                      INIT( '* DIGITAL SIGNATURE GENERATION *' );
DCL FILE_OUT_LINE   CHAR( 128 );
DCL OWH_HEADER      CHAR( 16 )
                      INIT( '* ONE WAY HASH *' );
DCL PKB_HEADER      CHAR( 16 )
                      INIT( '* PKA TOKEN BUILD *' );
DCL PKI_HEADER      CHAR( 16 )
                      INIT( '* PKA TOKEN IMPORT *' );
DCL RC_STRING       CHAR( 14 ) INIT( 'RETURN CODE = ' );
DCL RS_STRING       CHAR( 14 ) INIT( 'REASON CODE = ' );
DCL SIG_STRING      CHAR( 12 ) INIT( 'SIGNATURE = ' );
DCL SIG_LEN_STRING  CHAR( 26 ) INIT( 'SIGNATURE LENGTH(BYTES) = ' );

/* Declarations - Built-in functions */

DCL (SUBSTR, LENGTH) BUILTIN;

/*****
/* Call one-way hash to get the SHA-1 hash of the text. */
*****/
TEXT_LENGTH = LENGTH( TEXT );
OWH_TEXT = SUBSTR( TEXT, 1, TEXT_LENGTH );

CALL CSNBOWH( RETURN_CODE,
              REASON_CODE,
              EXIT_LEN,
              EXIT_DATA,

```



```

        RULE_ARY_CNT_OWH,
        RULE_ARY_OWH,
        TEXT_LENGTH,
        OWH_TEXT,
        CHAINING_VECTOR_LENGTH,
        CHAINING_VECTOR,
        HASH_LENGTH,
        HASH );

PUT SKIP LIST( OWH_HEADER );
PUT SKIP LIST( RC_STRING || RETURN_CODE );
PUT SKIP LIST( RS_STRING || REASON_CODE );

/*****
/* Create the PKA RSA private external token.
*****/
IF RETURN_CODE = 0 THEN
    DO;

        PKA_TOKEN_LENGTH = 1024;

        CALL CSNDPKB( RETURN_CODE,
                      REASON_CODE,
                      EXIT_LEN,
                      EXIT_DATA,
                      RULE_ARY_CNT_PKB,
                      RULE_ARY_PKB,
                      KEY_VALUE_LENGTH,
                      KEY_VALUE_STRUCTURE,
                      PRIVATE_NAME_LENGTH,
                      PRIVATE_NAME,
                      RESERVED_FIELD_LENGTH,
                      RESERVED_FIELD,
                      RESERVED_FIELD_LENGTH,
                      RESERVED_FIELD,
                      RESERVED_FIELD_LENGTH,
                      RESERVED_FIELD,
                      RESERVED_FIELD_LENGTH,
                      RESERVED_FIELD,
                      RESERVED_FIELD_LENGTH,
                      RESERVED_FIELD,
                      RESERVED_FIELD_LENGTH,
                      RESERVED_FIELD,
                      PKA_TOKEN_LENGTH,
                      PKA_KEY_TOKEN );

        PUT SKIP LIST( PKB_HEADER );
        PUT SKIP LIST( RC_STRING || RETURN_CODE );
        PUT SKIP LIST( RS_STRING || REASON_CODE );

    END;

/*****
/* Import the clear RSA private external token.
*****/
IF RETURN_CODE = 0 THEN
    DO;

        INTERNAL_PKA_TOKEN_LENGTH = 1024;

        CALL CSNDPKI( RETURN_CODE,
                      REASON_CODE,
                      EXIT_LEN,
                      EXIT_DATA,
                      RULE_ARY_CNT_PKI,
                      RULE_ARY_PKI,
                      PKA_TOKEN_LENGTH,
                      PKA_KEY_TOKEN,
                      DUMMY_KEY,

```

```

INTERNAL_PKA_TOKEN_LENGTH,
INTERNAL_PKA_TOKEN );

PUT SKIP LIST( PKI_HEADER );
PUT SKIP LIST( RC_STRING || RETURN_CODE );
PUT SKIP LIST( RS_STRING || REASON_CODE );

END;
/*****
/* Call digital signature generate. */
*****/
IF RETURN_CODE = 0 THEN
DO;

SIGNATURE_LENGTH = 128;

CALL CSNDDSG( RETURN_CODE,
              REASON_CODE,
              EXIT_LEN,
              EXIT_DATA,
              RULE_ARY_CNT_DSG,
              RULE_ARY_DSG,
              INTERNAL_PKA_TOKEN_LENGTH,
              INTERNAL_PKA_TOKEN,
              HASH_LENGTH,
              HASH,
              SIGNATURE_LENGTH,
              SIG_BIT_LENGTH,
              SIGNATURE );

PUT SKIP LIST( DSG_HEADER );
PUT SKIP LIST( RC_STRING || RETURN_CODE );
PUT SKIP LIST( RS_STRING || REASON_CODE );

IF RETURN_CODE = 0 THEN
DO;

/*****
/* Write the signature and its length to the output file. */
*****/
FILE_OUT_LINE = SIG_LEN_STRING || SIGNATURE_LENGTH;
WRITE FILE(SIGOUT) FROM( FILE_OUT_LINE );
FILE_OUT_LINE = SIG_STRING || SIGNATURE;
WRITE FILE(SIGOUT) FROM( FILE_OUT_LINE );
END;

END;

END DSIGEXP;

```



---

## Appendix E. Using ICSF with BSAFE

ICSF works in conjunction with RSA Security, Inc.'s BSAFE toolkit (BSAFE 3.1 or later). If you are currently using applications developed with BSAFE, we strongly recommend you take advantage of the increased security and performance available with ICSF interfaces. The BHAPI interface has been stabilized since ICSF FMID HCR770B and may be removed in a future release.

Through BSAFE 3.1 you can access the ICSF services to:

- Compute message digests or hashes
- Generate random numbers
- Encipher and decipher data using the DES algorithm
- Generate and verify RSA digital signatures

---

### Some BSAFE Basics

BSAFE has many algorithm information types (called AIs). Many of the AIs can perform several cryptographic functions. For this reason, you must specify the algorithmic method (AM) to be used by supplying a chooser. If the cryptographic function requires a key, you supply key information to the BSAFE application with a key information (KI) type. For the most current information on the BSAFE user interface and a complete description of algorithm information types, algorithm methods, choosers, and key information types, refer to BSAFE User's Guide and BSAFE Library Reference Manual.

### Computing Message Digests and Hashes

MD5 and SHA1 hashing are both available from ICSF via BSAFE. If your BSAFE application uses the AM\_MD5 or the AM\_SHA algorithm methods, you can add a couple of BSAFE function calls and the application will use ICSF and the Cryptographic Coprocessor Feature instead of the BSAFE algorithm method.

The following list shows BSAFE AI types with choosers that may include AM\_MD5:

- AI\_MD5
- AI\_MD5\_BER
- AI\_MD5WithDES\_CBCPad
- AI\_MD5WithDES\_CBCPadBER
- AI\_MD5WithRC2\_CBCPad
- AI\_MD5WithRC2\_CBCPadBER
- AI\_MD5WithRSAEncryption
- AI\_MD5WithRSAEncryptionBER
- AI\_MD5WithXOR
- AI\_MD5WithXOR\_BER

The following list shows BSAFE AI types with choosers that may include AM\_SHA:

- AI\_SHA1
- AI\_SHA1\_BER
- AI\_SHA1WithDES\_CBCPad
- AI\_SHA1WithDES\_CBCPadBER

## Generating Random Numbers

If your BSAFE application uses the algorithm method `AM_MD5_RANDOM`, you can add a chooser definition containing the algorithm method `AM_HW_RANDOM` (new with BSAFE 3.1) and a couple of BSAFE function calls and your program can use ICSF and the Cryptographic Coprocessor Feature to generate random numbers instead of the BSAFE algorithm method.

BSAFE 3.1 provides a new algorithm information type, `AI_HWRandom`. You need to set your random number generation object with `AI_HWRandom`, and initialize the object with a chooser containing `AM_HW_RANDOM`, in order to use ICSF with the Cryptographic Coprocessor Feature for generating random numbers. You do not, however, have to make a `B_RandomUpdate` call, since the S/390 and IBM @server zSeries cryptographic solution does not require a seed.

The only AI type with choosers that may include `AM_HW_RANDOM` is `AI_HWRandom`.

## Encrypting and Decrypting with DES

If your BSAFE application uses either the `AM_DES_CBC_ENCRYPT` or the `AM_DES_CBC_DECRYPT` algorithm methods, you can add a chooser containing the algorithm methods `AM_TOKEN_DES_CBC_ENCRYPT` and/or `AM_TOKEN_DES_CBC_DECRYPT` (both new with BSAFE 3.1) and a couple of BSAFE function calls and your program can use ICSF and the Cryptographic Coprocessor Feature to encrypt and/or decrypt data using the DES algorithm.

For your encryption or decryption key, you can use either a clear key in the form of a `KI_8Byte` or `KI_DES8` or `KI_Item` (8 bytes long), or a CCA DES Key Token in the form of a `KI_TOKEN` (64 bytes long). `KI_TOKEN` is a new key information type in BSAFE 3.1.

The following list shows BSAFE AI types with choosers that may include either `AM_TOKEN_DES_CBC_ENCRYPT`, `AM_TOKEN_DES_CBC_DECRYPT`, or both:

- `AI_DES_CBC_BSAFE1`
- `AI_DES_CBC_IV8`
- `AI_DES_CBCPadBER`
- `AI_DES_CBCPadIV8`
- `AI_DES_CBCPadPEM`
- `AI_MD5WithDES_CBCPad`
- `AI_MD5WithDES_CBCPadBER`
- `AI_SHA1WithDES_CBCPad`
- `AI_SHA1WithDES_CBCPadBER`

## Generating and Verifying RSA Digital Signatures

You can use algorithm method `AM_TOKEN_RSA_PRV_ENCRYPT` with `AM_MD5` or `AM_SHA` to have ICSF and the Cryptographic Coprocessor Feature generate RSA digital signatures. To verify the RSA digital signature using the S/390 or IBM @server zSeries cryptographic solution, you can use `AM_TOKEN_RSA_PUB_DECRYPT` (with `AM_MD5` or `AM_SHA`). Your BSAFE application must contain a couple of new BSAFE function calls to access the S/390 and IBM @server zSeries services. `AM_TOKEN_RSA_PRV_ENCRYPT` and `AM_TOKEN_RSA_PUB_DECRYPT` are new in BSAFE 3.1. For more information, see "Using the New Function Calls in Your BSAFE Application" on page 923.

For signature generation, you can use either a clear private key in the form of a `KI_PKCS_RSAPrivate` or a CCA RSA private key token in the form of a `KI_TOKEN`. For signature verification, you can use either a public RSA key in the form of a `KI_RSAPublic` or a CCA RSA public key token in the form of a `KI_TOKEN`. `KI_TOKEN` is a new key information type in BSAFE. For more information about `KI_TOKEN`, see “Using the BSAFE `KI_TOKEN`” on page 925.

The following list shows BSAFE AI types with choosers that may include `AM_TOKEN_RSA_PRIV_ENCRYPT`:

- `AI_MD5WithRSAEncryption`
- `AI_MD5WithRSAEncryptionBER`
- `AI_SHA1WithRSAEncryption`
- `AI_SHA1WithRSAEncryptionBER`

The following list shows BSAFE AI types with choosers that may include `AM_TOKEN_RSA_PUB_DECRYPT`:

- `AI_MD5WithRSAEncryption`
- `AI_SHA1WithRSAEncryption`

---

## Encrypting and Decrypting with RSA

You can use algorithm method `AM_TOKEN_RSA_ENCRYPT` to have ICSF encrypt a symmetric key (or other string of 48 bytes or fewer). To decrypt the string using ICSF, you can use `AM_TOKEN_RSA_CRT_DECRYPT`. You'll need a couple of new BSAFE function calls to access the S/390 and IBM @server zSeries services (see “Using the New Function Calls in Your BSAFE Application.”)

To encrypt a string, you can use either a public key in the form `KI_RSAPublic` or a CCA RSA public key token in the form of a `KI_TOKEN`.

To decrypt a string, you can use either a private key in the form `KI_PKCS_RSAPrivate` or a CCA RSA private key token in the form of a `KI_TOKEN`.

---

## Using the New Function Calls in Your BSAFE Application

To have your BSAFE application access the ICSF, S/390, and IBM @server zSeries Cryptographic Coprocessor Feature services, you need to add several new elements to your program. These elements are explained with examples in the steps that follow.

1. At the beginning of your program, declare one or more session choosers and also the hardware table list. For information about choosers and the hardware table list, see BSAFE User's Guide.

```
/*-----*
 * SESSION_CHOOSER will replace OLD_CHOOSER.                *
 *-----*/
B_ALGORITHM_METHOD **SESSION_CHOOSER = NULL_PTR;

/*-----*
 * CCA_VTABLE is a vector table of functions that will be    *
 * substituted for BSAFE equivalents. It is supplied by IBM *
 * and will be loaded into your application when you invoke *
 * QueryCrypto.                                              *
 *-----*/
HW_TABLE_LIST CCA_VTABLE = (HW_TABLE_LIST)NULL_PTR;
```

2. Declare a tag list. The content of the tag list is supplied by BSAFE at the B\_CreateSessionChooser call, which is discussed in a later step.  

```
unsigned char **taglist = (unsigned char **)NULL_PTR;
```
3. For random number generation, DES encryption or decryption or RSA encryption or decryption, you need to define and declare an additional chooser wherever your current chooser is defined and declared. For instance, suppose your application is doing an RSA encryption, and OLD\_CHOOSER is defined as follows:

```
/*-----*
 * OLD_CHOOSER is used for this application when ICSF and      *
 * the crypto hardware is not available.                      *
 *-----*/
B_ALGORITHM_METHOD *OLD_CHOOSER[] = {
    &AM_SHA,
    &AM_RSA_ENCRYPT,
    (B_ALGORITHM_METHOD *)NULL_PTR
};

/*-----*
 * ICSF_CHOOSER is a 'skeleton' for SESSION_CHOOSER.         *
 * SESSION_CHOOSER will be used for this application if      *
 * ICSF and the crypto hardware are not available.           *
 *-----*/
B_ALGORITHM_METHOD *ICSF_CHOOSER[] = {
    &AM_SHA,
    &AM_TOKEN_RSA_PUB_ENCRYPT,
    (B_ALGORITHM_METHOD *)NULL_PTR
};
```

4. At the beginning of the main function in your application, add a call to the ICSF QueryCrypto function followed by a conditional call to the BSAFE B\_CreateSessionChooser function.

```
/*-----*
 * Check for the existence of crypto hardware.  If it's there, *
 * QueryCrypto will supply CCA_VTABLE                      *
 *-----*/
if ((status = QueryCrypto(CRYPTO_Q_DES_AND_RSA,&CCA_VTABLE)) == 0)
/*-----*
    * B_CreateSessionChooser will replace the                *
    * BSAFE software functions with their CCA                *
    * hardware equivalents.                                  *
    *                                                         *
    * Note that the last three parameters are not            *
    * used with CCA                                           *
    *-----*/
    if ((status = B_CreateSessionChooser(ICSF_CHOOSER,
                                         &SESSION_CHOOSER,
                                         CCA_VTABLE,
                                         (ITEM *)NULL_PTR,
                                         (POINTER *)NULL_PTR,
                                         &taglist)) != 0)
        break;
```

5. Set up the conditions under which any alternate choosers are used to initialize the appropriate algorithm object. For information about initializing algorithm objects, see BSAFE User's Guide.

```
/*-----*
 * Initialize the algorithm object with the appropriate      *
 * chooser.                                                  *
 *-----*/
if (SESSION_CHOOSER != NULL_PTR)
    if ((status = B_XXXXXInit
        (xxxxxxObject,SESSION_CHOOSER,
         (A_SURRENDER_CTX *)NULL_PTR)) != 0)
```

```

        break;
    else ;
else
    if ((status = B_XXXXXXInit
        (XXXXXXObject,OLD_CHOOSER,
        (A_SURRENDER_CTX *)NULL_PTR)) != 0)
        break;
    else ;

```

6. When your application no longer needs the session chooser, program a call to the BSAFE B\_FreeSessionChooser function.

```

if (SESSION_CHOOSER != NULL_PTR)
    B_FreeSessionChooser(&SESSION_CHOOSER,&taglist);

```

---

## Using the BSAFE KI\_TOKEN

Those ICSF functions that require a key, like encipher and decipher, expect the key in the form of a CCA token. If you already have a CCA token, perform the following steps before you try to set your algorithm object. For information about how to perform the following tasks, see BSAFE User's Guide and BSAFE Library Reference Manual.

1. Create a key object.
2. Declare a KEY\_TOKEN\_INFO and fill it in.

KEY\_TOKEN\_INFO is defined as follows in the BSAFE User's Guide:

```

typedef struct {
    ITEM manufacturerID;
    ITEM internalKeyInfo;
} KEY_TOKEN_INFO;

```

The first ITEM is the address and length of one of the following three strings, depending on the CCA key token type you are using:

- com.ibm.CCADES
- com.ibm.CCARSAPublic
- com.ibm.CCARSAPrivate

The second ITEM is the address and length of your CCA key token.

3. Set the key information (B\_SetKeyInfo) into the key object using the item and a key information type of KI\_TOKEN as input.

If you don't already have a CCA token, you can supply a clear key to the function using one of the key information types mentioned in the section discussing the function you are using. BSAFE will convert the key to a CCA token. If you supply a clear BSAFE KI type to one of the ICSF functions, and the function is performed successfully, you can retrieve the key as a CCA token by invoking B\_GetKeyInfo with KI\_TOKEN as the key information type. A KEY\_TOKEN\_INFO struct is returned.

---

## ICSF Triple DES via BSAFE

ICSF performs single, double, or triple DES depending on the length of the DES key; if you're using BSAFE to access ICSF triple DES, you should use the algorithm methods AM\_TOKEN\_DES\_CBC\_ENCRYPT and AM\_TOKEN\_DES\_CBC\_DECRYPT.

If you've already have an ICSF token, follow the instructions in the section titled "Using the BSAFE KI\_TOKEN."



If you're using a clear key, follow the same procedure, except use your clear key padded on the right with binary zeroes to a length of 64 as the internalKeyInfo part of your KI\_TOKEN\_INFO. ICSF will convert your clear key to an internal ICSF key token.

Here's an example:

```
B_KEY_OBJ desKey = (B_KEY_OBJ)NULL_PTR;
KEY_TOKEN_INFO myTokenInfo;
unsigned char myToken[64] = {0};
unsigned char * myTokenP;
unsigned char myDoubleKey[16]; /* Input to this function */
unsigned char mfgID[] = "com.ibm.CCADES";
unsigned char * mfgIDP;
.
.
.
myTokenP = myToken;
mfgIDP = mfgID;
T_memcpy(myToken, myDoubleKey, sizeof(myDoubleKey));
myTokenInfo.manufacturerID.len = strlen(mfgID);
myTokenInfo.manufacturerID.data = mfgIDP;
myTokenInfo.internalKeyInfo.len = sizeof(myToken);
myTokenInfo.internalKeyInfo.data = myTokenP;

/* Create a key object. */
if ((status = B_CreateKeyObject (&desKey)) != 0)
    break;

/* Set the key object. */
if ((status = B_SetKeyInfo
    (desKey, KI_TOKEN, myTokenInfo )) != 0)
    break;
.
.
.
```

---

## Retrieving ICSF Error Information

When using the ICSF and Cryptographic Coprocessor Feature, Init, Update, and Final calls can result in BSAFE returning a status of BE\_HARDWARE (0x020B). When this occurs, you can derive the ICSF return and reason codes by using a new BSAFE operation, B\_GetExtendedErrorInfo. For an explanation of the return codes and reason codes, see Appendix A, "ICSF and TSS Return and Reason Codes," on page 781.

A coding example follows.

```
.
.
#include "balg.h"
#include "alobj.h"
#include "cca.h"
.
.
{
.
.
.
B_ALGORITHM_OBJECT * aop;
ITEM * errp;
unsigned char * algorithmMethod;
CCA_ERROR_DATA * edp;
unsigned int CCAreturnCode=0;
unsigned int CCAreasonCode=0;
```

```

unsigned char algorithmName[40]={0x00};
.
.
.
if (status==BE_HARDWARE) {
    B_GetExtendedErrorInfo(aop,errp,algorithmMethod);
    edp = errp->data;
    CCAReturnCode = (unsigned int) edp->returnCode;
    CCAreasonCode = (unsigned int) edp->reasonCode;
}
.
.
}

```

The prototype for B\_GetExtendedErrorInfo is in balg.h, as shown in the example that follows.

```

B_GetExtendedErrorInfo (
    B_ALGORITHM_OBJ algorithmObject, /* in--algorithm object */
    ITEM * errorData, /* out--address and length of error data */
    POINTER algorithmMethod /* out--address of faulting AM */
);

```



---

## Appendix F. Cryptographic Algorithms and Processes

This appendix describes the personal identification number (PIN) formats and algorithms.

---

### PIN Formats and Algorithms

For PIN calculation procedures, see IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference.

#### PIN Notation

This section describes various PIN block formats. The following notations describe the contents of PIN blocks:

- P** = A 4-bit decimal digit that is one digit of the PIN value.
- C** = A 4-bit hexadecimal control value. The valid values are X'0', X'1', and X'2'.
- L** = A 4-bit hexadecimal value that specifies the number of PIN digits. The value ranges from 4 to 12, inclusive.
- F** = A 4-bit field delimiter of value X'F'.
- f** = A 4-bit delimiter filler that is either P or F, depending on the length of the PIN.
- D** = A 4-bit decimal padding value. All pad digits in the PIN block have the same value.
- X** = A 4-bit hexadecimal padding value. All pad digits in the PIN block have the same value.
- x** = A 4-bit hexadecimal filler that is either P or X, depending on the length of the PIN.
- R** = A 4-bit hexadecimal random digit. The sequence of R digits can each take a different value.
- r** = A 4-bit random filler that is either P or R, depending on the length of the PIN.
- Z** = A 4-bit hexadecimal zero (X'0').
- z** = A 4-bit zero filler that is either P or Z, depending on the length of the PIN.
- S** = A 4-bit hexadecimal digit that constitutes one digit of a sequence number.
- A** = A 4-bit decimal digit that constitutes one digit of a user-specified constant.

#### PIN Block Formats

This section describes the PIN block formats and assigns a code to each format.

##### ANSI X9.8

This format is also named ISO format 0, VISA format 1, VISA format 4, and ECI format 1.

P1 = CLPPPPffffffffFF

P2 = ZZZZAAAAAAAAAA

PIN Block = P1 XOR P2

where C = X'0'  
L = X'4' to X'C'

**Programming Note:** The rightmost 12 digits (excluding the check digit) in P2 are the rightmost 12 digits of the account number for all formats except VISA format 4. For VISA format 4, the rightmost 12 digits (excluding the check digit) in P2 are the leftmost 12 digits of the account number.

### ISO Format 1

This format is also named ECI format 4.

PIN Block = CLPPPPrrrrrrrrRR

where C = X'1'  
L = X'4' to X'C'

### ISO Format 2

PIN Block = CLPPPPffffffffFF

where C = X'2'  
L = X'4' to X'C'

### VISA Format 2

PIN Block = LPPPPzzDDDDDDDD

where L = X'4' to X'6'

### VISA Format 3

This format specifies that the PIN length can be 4-12 digits, inclusive. The PIN starts from the leftmost digit and ends by the delimiter ('F'), and the remaining digits are padding digits.

An example of a 6-digit PIN:

PIN Block = PPPPPFXXXXXXXX

### IBM 4700 Encrypting PINPAD Format

This format uses the value X'F' as the delimiter for the PIN.

PIN Block = LPPPPffffffffFSS

where L = X'4' to X'C'

### IBM 3624 Format

This format requires the program to specify the delimiter, X, for determining the PIN length.

PIN Block = PPPPxxxxxxxxXXXX

### IBM 3621 Format

This format requires the program to specify the delimiter, X, for determining the PIN length.

PIN Block = SSSPPPPxxxxxxxx

### ECI Format 2

This format defines the PIN to be 4 digits.

PIN Block = PPPRRRRRRRRRRR

## ECI Format 3

PIN Block = LPPPPzzRRRRRRRRR

where L = X'4' to X'6'

## PIN Extraction Rules

This section describes the PIN extraction rules for the Encrypted PIN verify and Encrypted PIN translate callable services.

### Encrypted PIN Verify Callable Service

The service extracts the customer-entered PIN from the input PIN block according to the following rules:

- If the input PIN block format is ANSI X9.8, ISO format 0, VISA format 1, VISA format 4, ECI format 1, ISO format 1, ISO format 2, VISA format 2, IBM Encrypting PINPAD format, or ECI format 3, the service extracts the PIN according to the length specified in the PIN block.
- If the input PIN block format is VISA format 3, the specified delimiter (padding) determines the PIN length. The search starts at the leftmost digit in the PIN block. If the input PIN block format is 3624, the specification of a PIN extraction method for the 3624 is supported through rule array keywords. If no PIN extraction method is specified in the rule array, the specified delimiter (padding) determines the PIN length.
- If the input PIN block format is 3621, the specification of a PIN extraction method for the 3621 is supported through rule array keywords. If no PIN extraction method is specified in the rule array, the specified delimiter (padding) determines the PIN length.
- If the input PIN block format is ECI format 2, the PIN is the leftmost 4 digits.

For the VISA algorithm, if the extracted PIN length is less than 4, the services sets a reason code that indicates that verification failed. If the length is greater than or equal to 4, the service uses the leftmost 4 digits as the referenced PIN.

For the IBM German Banking Pool algorithm, if the extracted PIN length is not 4, the service sets a reason code that indicates that verification failed.

For the IBM 3624 algorithm, if the extracted PIN length is less than the PIN check length, the service sets a reason code that indicates that verification failed.

### Clear PIN Generate Alternate Callable Service

The service extracts the customer-entered PIN from the input PIN block according to the following rules:

- This service supports the specification of a PIN extraction method for the 3624 and 3621 PIN block formats through the use of the *rule\_array* keyword. *Rule\_array* points to an array of one or two 8-byte elements. The first element in the rule array specifies the PIN calculation method. The second element in the rule array (if specified) indicates the PIN extraction method. Refer to the “Clear PIN Generate Alternate (CSNBCPA and CSNECPA)” on page 485 for an explanation of PIN extraction method keywords.

### Encrypted PIN Translate Callable Service

The service extracts the customer-entered PIN from the input PIN block according to the following rules:

- If the input PIN block format is ANSI X9.8, ISO format 0, VISA format 1, VISA format 4, ECI format 1, ISO format 1, ISO format 2, VISA format 2, IBM

Encrypting PINPAD format, or ECI format 3, and if the specified PIN length is less than 4, the service sets a reason code to reject the operation. If the specified PIN length is greater than 12, the operation proceeds to normal completion with unpredictable contents in the output PIN block. Otherwise, the service extracts the PIN according to the specified length.

- If the input PIN block format is VISA format 3, the specified delimiter (padding) determines the PIN length. The search starts at the leftmost digit in the PIN block. If the input PIN block format is 3624, the specification of a PIN extraction method for the 3624 is supported through rule array keywords. If no PIN extraction method is specified in the rule array, the specified delimiter (padding) determines the PIN length.
- If the input PIN block format is 3621, the specification of a PIN extraction method for the 3621 is supported through rule array keywords. If no PIN extraction method is specified in the rule array, the specified delimiter (padding) determines the PIN length.
- If the input block format is ECI format 2, the PIN is always the leftmost 4 digits.

If the maximum PIN length allowed by the output PIN block is shorter than the extracted PIN, only the leftmost digits of the extracted PIN that form the allowable maximum length are placed in the output PIN block. The PIN length field in the output PIN block, if it exists, specifies the allowable maximum length.

### **PIN Change/Unblock Callable Service**

The PIN Block calculation PIN Change/Unblock:

1. Form three 8-byte, 16-digit blocks, -1, -2, and -3, and set all digits to X'0'
2. Replace the rightmost four bytes of block-1 with the authentication code described in the previous section.
3. Set the second digit of block-2 to the length of the new PIN (4 to 12), followed by the new PIN, and padded to the right with X'F'
4. Include any current PIN by placing it into the leftmost digits of block-3.
5. Exclusive-OR blocks -1, -2, and -3 to form the 8-byte PIN block.
6. Pad the PIN block with other portions of the message for the smart card:
  - Prepend X'08'
  - Append X'80'
  - Append an additional six bytes of X'00'

The resulting message is ECB-mode triple-encrypted with an appropriate session key.

## **IBM PIN Algorithms**

This section describes the IBM PIN generation algorithms, IBM PIN offset generation algorithm, and IBM PIN verification algorithms.

### **3624 PIN Generation Algorithm**

This algorithm generates a n-digit PIN based on an account-related data or person-related data, namely the validation data. The assigned PIN length parameter specifies the length of the generated PIN.

The algorithm requires the following input parameters:

- A 64-bit validation data
- A 64-bit decimalization table
- A 4-bit assigned PIN length

- A 128-bit PIN-generation key

The service uses the PIN generation key to encipher the validation data. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of the enciphered validation data. The result is an intermediate PIN. The leftmost  $n$  digits of the intermediate PIN are the generated PIN, where  $n$  is specified by the assigned PIN length.

Figure 14 illustrates the 3624 PIN generation algorithm.

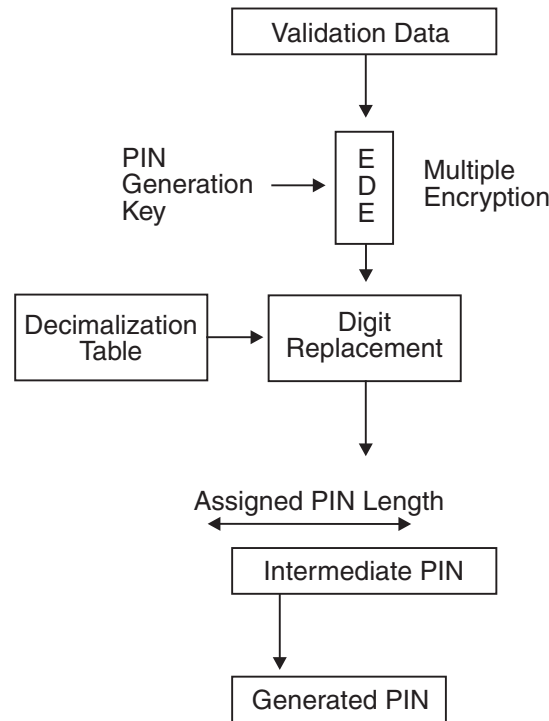


Figure 14. 3624 PIN Generation Algorithm

### German Banking Pool PIN Generation Algorithm

This algorithm generates a 4-digit PIN based on an account-related data or person-related data, namely the validation data.

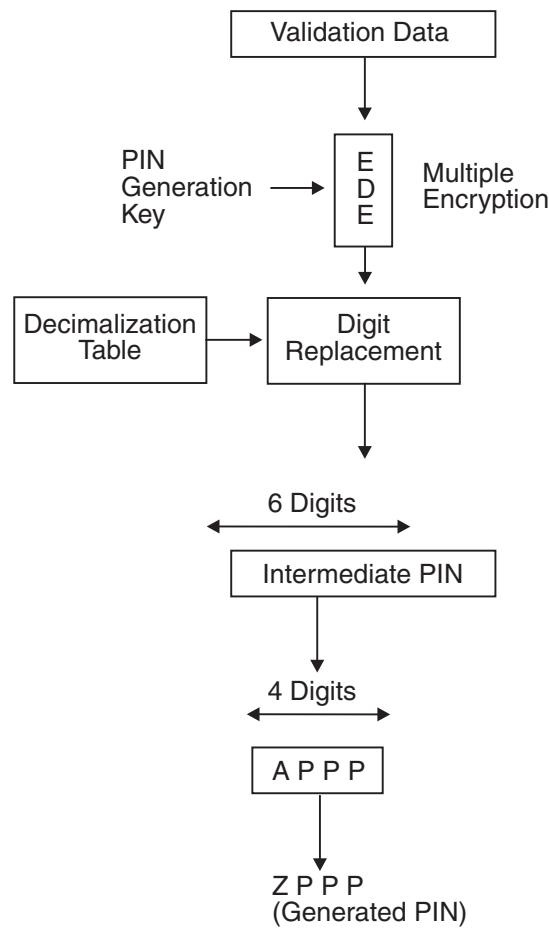
The algorithm requires the following input parameters:

- A 64-bit validation data
- A 64-bit decimalization table
- A 128-bit PIN-generation key

The validation data is enciphered using the PIN generation key. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of enciphered validation data. The result is an intermediate PIN. The rightmost 4 digits of the leftmost 6 digits of the intermediate PIN are extracted. The leftmost digit of the extracted 4 digits is checked for zero. If the digit is zero, the digit is changed to one; otherwise, the digit remains unchanged. The resulting four digits is the generated PIN.



Figure 15 illustrates the German Banking Pool (GBP) PIN generation algorithm.



If A = 0, then Z = 1; otherwise, Z = A.

Figure 15. GBP PIN Generation Algorithm

### PIN Offset Generation Algorithm

To allow the customer to select his own PIN, a PIN offset is used by the IBM 3624 and GBP PIN generation algorithms to relate the customer-selected PIN to the generated PIN.

The PIN offset generation algorithm requires two parameters in addition to those used in the 3624 PIN generation algorithm. They are a customer-selected PIN and a 4-bit PIN check length. The length of the customer-selected PIN is equal to the assigned-PIN length, n.

The 3624 PIN generation algorithm described in the previous section is performed. The offset data value is the result of subtracting (modulo 10) the leftmost n digits of the intermediate PIN from the customer-selected PIN. The modulo 10 subtraction ignores borrows. The rightmost m digits of the offset data form the PIN offset, where m is specified by the PIN check length. Note that n cannot be less than m. To generate a PIN offset for a GBP PIN, m is set to 4 and n is set to 6.

Figure 16 illustrates the PIN offset generation algorithm.



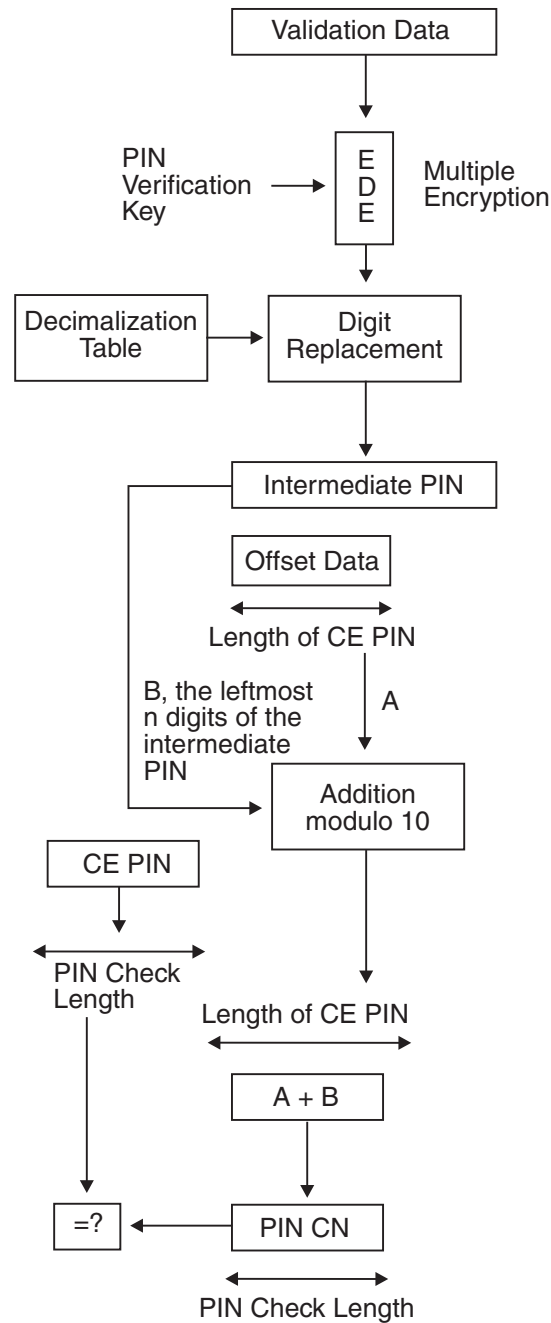
- An offset data
- A customer-entered PIN

The rightmost  $m$  digits of the offset data form the PIN offset, where  $m$  is the PIN check length.

1. The validation data is enciphered using the PIN verification key. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of enciphered validation data.
2. The leftmost  $n$  digits of the result is added (modulo 10) to the offset data value, where  $n$  is the length of the customer-entered PIN. The modulo 10 addition ignores carries.
3. The rightmost  $m$  digits of the result of the addition operation form the PIN check number. The PIN check number is compared with the rightmost  $m$  digits of the customer-entered PIN. If they match, PIN verification is successful; otherwise, verification is unsuccessful.

When a nonzero PIN offset is used, the length of the customer-entered PIN is equal to the assigned PIN length.

Figure 17 illustrates the PIN verification algorithm.



PIN CN: PIN Check Number  
CE PIN: Customer-entered PIN

Figure 17. PIN Verification Algorithm

### German Banking Pool PIN Verification Algorithm

This algorithm generates an intermediate PIN based on the specified validation data. A part of the intermediate PIN is adjusted by adding an offset data. A part of the result is extracted. The extracted value may or may not be modified before it compares with the customer-entered PIN.

The algorithm requires the following input parameters:

- A 64-bit validation data
- A 64-bit decimalization table
- A 128-bit PIN verification key
- An offset data
- A customer-entered PIN

The rightmost 4 digits of the offset data form the PIN offset.

1. The validation data is enciphered using the PIN verification key. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of enciphered validation data.
2. The leftmost 6 digits of the result is added (modulo 10) to the offset data. The modulo 10 addition ignores carries.
3. The rightmost 4 digits of the result of the addition (modulo 10) are extracted.
4. The leftmost digit of the extracted value is checked for zero. If the digit is zero, the digit is set to one; otherwise, the digit remains unchanged. The resulting four digits are compared with the customer-entered PIN. If they match, PIN verification is successful; otherwise, verification is unsuccessful.

Figure 18 illustrates the GBP PIN verification algorithm.

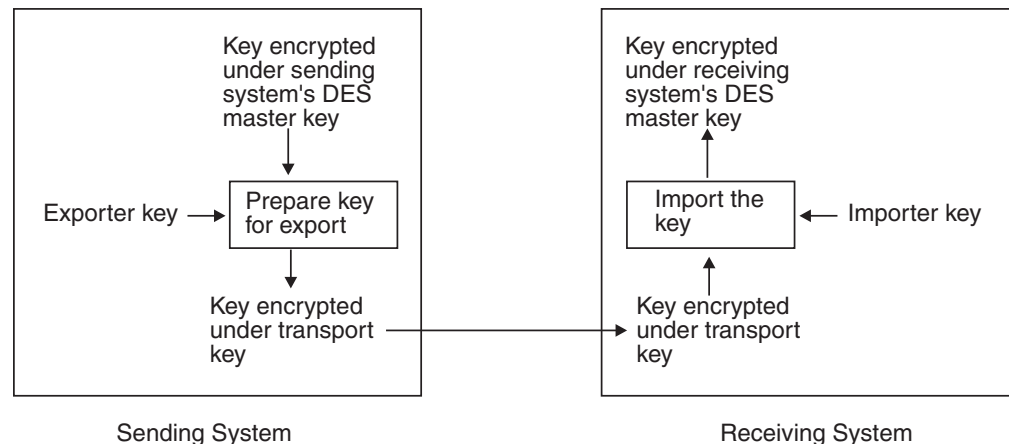


Figure 18. GBP PIN Verification Algorithm

## VISA PIN Algorithms

The VISA PIN verification algorithm performs a multiple encipherment of a value, called the transformed security parameter (TSP), and an extraction of a 4-digit PIN verification value (PVV) from the ciphertext. The calculated PVV is compared with the referenced PVV and stored on the plastic card or data base. If they match, verification is successful.

### PVV Generation Algorithm

The algorithm generates a 4-digit PIN verification value (PVV) based on the transformed security parameter (TSP).

The algorithm requires the following input parameters:

- A 64-bit TSP
- A 128-bit PVV generation key

1. A multiple encipherment of the TSP using the double-length PVV generation key is performed.
2. The ciphertext is scanned from left to right. Decimal digits are selected during the scan until four decimal digits are found. Each selected digit is placed from left to right according to the order of selection. If four decimal digits are found, those digits are the PVV.
3. If, at the end of the first scan, less than four decimal digits have been selected, a second scan is performed from left to right. During the second scan, all decimal digits are skipped and only nondecimal digits can be processed. Nondecimal digits are converted to decimal digits by subtracting 10. The process proceeds until four digits of PVV are found.

Figure 19 illustrates the PVV generation algorithm.

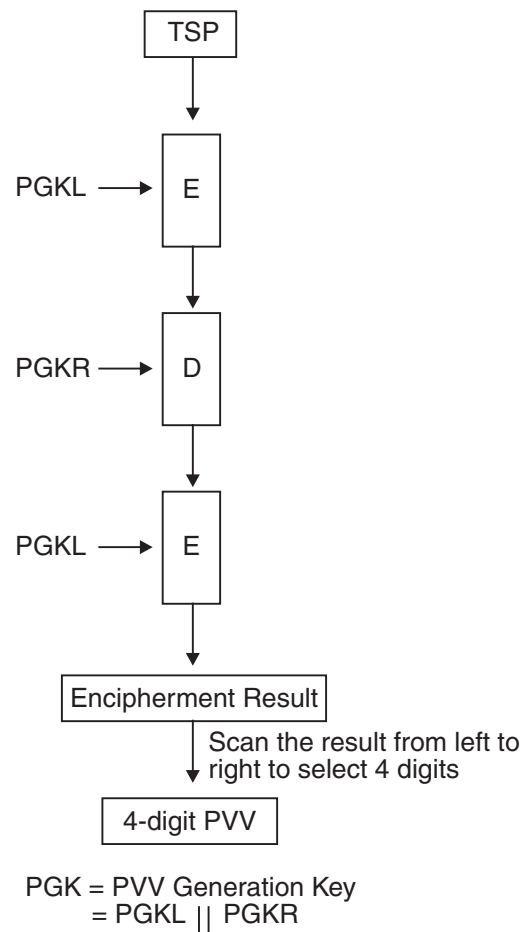


Figure 19. PVV Generation Algorithm

**Programming Note:** For VISA PVV algorithms, the leftmost 11 digits of the TSP are the personal account number (PAN), the leftmost 12th digit is a key table index to select the PVV generation key, and the rightmost 4 digits are the PIN. The key table index should have a value between 1 and 6, inclusive.

### PVV Verification Algorithm

The algorithm requires the following input parameters:

- A 64-bit TSP

- A 16-bit referenced PVV
- A 128-bit PVV verification key

A PVV is generated using the PVV generation algorithm, except a PVV verification key rather than a PVV generation key is used. The generated PVV is compared with the referenced PVV. If they match, verification is successful.

### Interbank PIN Generation Algorithm

The Interbank PIN calculation method consists of the following steps:

1. Let X denote the transaction\_security parameter element converted to an array of 16 4-bit numeric values. This parameter consists of (in the following sequence) the 11 rightmost digits of the customer PAN (excluding the check digit), a constant of 6, a 1-digit key indicator, and a 3-digit validation field.
2. Encrypt X with the double-length PINGEN (or PINVER) key to get 16 hexadecimal digits (64 bits).
3. Perform decimalization on the result of the previous step by scanning the 16 hexadecimal digits from left to right, skipping any digit greater than X'9' until 4 decimal digits (for example, digits that have values from X'0' to X'9') are found. If all digits are scanned but 4 decimal digits are not found, repeat the scanning process, skipping all digits that are X'9' or less and selecting the digits that are greater than X'9'. Subtract 10 (X'A') from each digit selected in this scan. If the 4 digits that were found are all zeros, replace the 4 digits with 0100.
4. Concatenate and use the resulting digits for the Interbank PIN. The 4-digit PIN consists of the decimal digits in the sequence in which they are found.

---

## Cipher Processing Rules

DES defines operations on 8-byte data strings. Although the fundamental concepts of ciphering (enciphering and deciphering) and data verification are simple, there are different approaches to processing data strings that are not a multiple of 8 bytes in length. These approaches are defined in various standards and IBM products.

### CBC and ANSI X3.106

ANSI standard X3.106 defines four methods of operation for ciphering. One of these modes, cipher block chaining (CBC), defines the basic method for performing ciphering on multiple blocks. A plaintext data string, which must be a multiple of the block size, is processed as a series of blocks. The ciphered result from processing a block is exclusive ORed with the next block. The last block of the ciphered result is defined as an output chaining vector (OCV). ICSF stores the output chaining vector value in the *chaining\_vector* parameter.

An initial chaining vector is exclusive ORed with the first group of 8 input bytes.

In summary:

- An input chaining vector (ICV) is required.
- If the *text\_length* is not an exact multiple of 8 bytes, the request fails.
- The plaintext is not padded, for example, the output text length is not increased.

ICSF provides an enhancement to CBC mode called ciphertext-stealing. This allows for a text length that is not a multiple of the block size. This is accomplished by manipulating the last two blocks in a certain way. The second to last block is encrypted in the normal manner, but then some of the bits are "stolen" and added

to the last (partial) block. These bits can be recovered by decrypting the last block. This enhancement is currently proposed to NIST as *Proposal To Extend CBC Mode By "Ciphertext Stealing"*, dated May 6, 2007.

## ANSI X9.23 and IBM 4700

An enhancement to the basic cipher block chaining mode of ANSI X3.106 is defined so the data lengths that are not an exact multiple of 8 bytes can be processed. The ANSI X9.23 method *always* adds from 1 byte to 8 bytes to the plaintext before encipherment. The last added byte is the count of the added bytes and is in the range of X'01' to X'08'. The standard defines that the other added bytes, the pad characters, are random.

When ICSF enciphers the plaintext, the resulting ciphertext is always 1 to 8 bytes longer than the plaintext.

When ICSF decipheres the ciphertext, ICSF uses the last byte of the deciphered data as the number of bytes to be removed (the pad bytes and the count byte). The resulting plaintext is the same as the original plaintext.

The output chaining vector can be used as feedback with this method in the same way as with the X3.106 method.

In summary, for the ANSI X9.23 method:

- X9.23 processing requires the caller to supply an ICV.
- X9.23 encipher does not allow specification of a pad character.

The 4700 padding rule is similar to the X9.23 rule. The only difference is that in the X9.23 method, the padding character is not user-selected, but the padding string is selected by the encipher process.

## Segmenting

The callable services can operate on large data objects. *Segmenting* is the process of dividing the function into more than one processing step. Your application can divide the process into multiple steps without changing the final outcome.

To provide segmenting capability, the MAC generation, MAC verification, and MDC generation callable services require an 18-byte system work area in the application address space that is provided as the chaining vector parameter to the callable service. The application program must not change the system work area.

## Cipher Last-Block Rules

The DES defines cipher-block chaining as operating on multiples of 8 bytes, and AES uses multiples of 16 bytes. Various algorithms are used to process strings that are multiples of the block size. The algorithms are generically named "last-block rules". You select the supported last-block rules by using these keywords:

- X9.23
- IPS
- CUSP (also used with PCF)
- 4700-PAD
- CBC-CS

You specify which cipher last-block rule you want to use in the *rule\_array* parameter of the callable service.



## CUSP

If the length of the data to be enciphered is an exact multiple of 8 bytes, the ICV is exclusive ORed with the first 8-byte block of plaintext, and the resulting 8 bytes are passed to the DES with the specified key. The resulting 8-byte block of ciphertext is then exclusive ORed with the second 8-byte block of plaintext, and the value is enciphered. This process continues until the last 8-byte block of plaintext is to be enciphered. Because the length of this last block is exactly 8 bytes, the last block is processed in an identical manner to all the preceding blocks.

To produce the OCV, the last block of *ciphertext* is enciphered again (thus producing a double-enciphered block). The user can pass this value of the OCV as the ICV in his next encipher call to produce chaining between successive calls. The caller can alternatively pass the same ICV on every call to the callable service.

If the length of data to be enciphered is greater than 7 bytes, and is *not* an exact multiple of 8 bytes, the process is the same as that above, until the last partial block of 1 to 7 bytes is reached. To encipher the last short block, the previous 8-byte block of ciphertext is passed to the DES with the specified key. The first 1 to 7 bytes of this double-enciphered block has two uses. The first use is to exclusive OR this block with the last short block of plaintext to form the last short block of the ciphertext. The second use is to pass it back as the OCV. Thus, the OCV is the last complete 8-byte block of plaintext, doubly enciphered.

If the length of the data to be enciphered is less than 8 bytes, the ICV is enciphered under the specified key. The first 1 to 7 bytes of the enciphered ICV is exclusive ORed with the plaintext to form the ciphertext. The OCV is the enciphered ICV.

## The Information Protection System (IPS)

The Information Protection System (IPS) offers two forms of chaining: block and record. Under record chaining, the OCV for each enciphered data string becomes the ICV for the next. Under block chaining, the same ICV is used for each encipherment.

Files that are enciphered directly with the ICSF encipher callable service cannot be properly deciphered using the IPS/CMS CIPHER command or the IPS/CMS subroutines. Both IPS/CMS CIPHER and AMS REPRO ENCIPHER write headers to their files that contain information (principally the ICV and chaining method) needed for decipherment. The encipher callable service does not generate these headers. Specialized techniques are described in IPS/CMS documentation to overcome some, if not all, of these limitations, depending on the chaining mode. As a rough test, you can attempt a decipherment with the CIPHER command HDWARN option, which causes CIPHER to continue processing even though the header is absent.

The encipher callable service returns an OCV used by IPS for record chaining. This allows cryptographic applications using ICSF to be compatible with IPS record chaining.

Record chaining provides a superior method of handling successive short blocks, and has better error recovery features when the caller passes successive short blocks.

The principle used by record chaining is that *the OCV is the last 8 bytes of ciphertext*. This is handled as follows:

- If the length of the data to be enciphered is an exact multiple of 8 bytes, the ICV is exclusive ORed with the first 8 byte block of plaintext, and the resulting 8 bytes are passed to the DES with the specified key. The resulting 8-byte block of ciphertext is then exclusive ORed with the second 8-byte block of plaintext, and the resulting value is enciphered. This process continues until the last 8-byte block of plaintext is to be enciphered. Because the length of this last block is exactly 8 bytes, the last block is processed in an identical manner to all the preceding blocks.

The OCV is the last 8 bytes of ciphertext.

The user can pass this value as the ICV in the next encipher call to produce chaining between successive calls.

- If the length of data to be enciphered is greater than 7 bytes, and is *not* an exact multiple of 8 bytes, the process is the same as that above, until the last partial block of 1 to 7 bytes is reached. To encipher the last short block, the previous 8-byte block of ciphertext is passed to the DES with the specified key. The first 1 to 7 bytes of this doubly enciphered block is then exclusive ORed with the last short block of plaintext to form the last short block of the ciphertext. The OCV is the last 8 bytes of ciphertext.
- If the length of the data to be enciphered is less than 8 bytes, then the ICV is enciphered under the specified key. The first 1 to 7 bytes of the enciphered ICV is exclusive ORed with the plaintext to form the ciphertext. The OCV is the rightmost 8 bytes of the plaintext ICV concatenated with the short block of ciphertext. For example:

```

ICV           = ABCDEFGH
ciphertext    = XYZ
OCV           = DEFGHXYZ

```

## PKCS Padding Method

This section describes the algorithm used to pad clear text when the PKCS-PAD method is specified. Padding is applied before encryption when this keyword is specified with the Symmetric Algorithm Encipher callable service, and it is removed from decrypted data when the keyword is specified with the Symmetric Algorithm Decipher callable service.

The rules for PKCS padding are very simple:

- Padding bytes are always added to the clear text before it is encrypted.
- Each padding byte has a value equal to the total number of padding bytes that are added. For example, if 6 padding bytes must be added, each of those bytes will have the value 0x06.
- The total number of padding bytes is at least one, and is the number that is required in order to bring the data length up to a multiple of the cipher algorithm block size.

The callable services described in this document use AES, which has a cipher block size of 16 bytes. The total number of padding bytes added to the clear text will always be between 1 and 16. The table below indicates exactly how many padding bytes are added according to the data length, and also shows the value of the padding bytes that are applied.

Value of clear text length (mod 16)	Number of padding bytes added	Value of each padding byte
0	16	0x10
1	15	0x0F

Value of clear text length (mod 16)	Number of padding bytes added	Value of each padding byte
2	14	0x0E
3	13	0x0D
4	12	0x0C
5	11	0x0B
6	10	0x0A
7	9	0x09
8	8	0x08
9	7	0x07
10	6	0x06
11	5	0x05
12	4	0x04
13	3	0x03
14	2	0x02
15	1	0x01

Note that the PKCS standards that define this padding method describe it in a way that limits the maximum padding length to 8 bytes. This is a consequence of the fact that the algorithms at that time used 8-byte blocks. We extend the definition to apply to 16-byte AES cipher blocks.

### PKCS Padding Method (Example 1)

Clear text consists of the following 18 bytes:

F14ADBDA019D6DB7 EFD91546E3FF8444 9BCB

In order to make this a multiple of 16 bytes (the AES block size), we must add 14 bytes. Each byte will contain the value 0x0E, which is 14, the total number of padding bytes added. The result is that the padded clear text is as follows:

F14ADBDA019D6DB7 EFD91546E3FF8444 9BCB0E0E0E0E0E0E  
0E0E0E0E0E0E0E0E

The padded value is 32 bytes in length, which is two AES blocks. This padded string is encrypted in CBC mode, and the resulting ciphertext will also be 32 bytes in length.

### PKCS Padding Method (Example 2)

Clear text consists of the following 16 bytes:

971ACD01C9C7ADEA CC83257926F490FF

This is already a multiple of the AES block size, but PKCS padding rules say that padding is always applied. Thus, we add 16 bytes of padding to bring the total length to 32, the next multiple of the AES block size. Each pad byte has the value 0x10, which is 16, the total number of padding bytes added. The result is that the padded clear text is as follows:

971ACD01C9C7ADEA CC83257926F490FF 1010101010101010  
1010101010101010

The padded value is 32 bytes in length, which is two AES blocks. This padded string is encrypted in CBC mode, and the resulting cipher text will also be 32 bytes in length.

---

## Wrapping Methods for Symmetric Key Tokens

This section explains how symmetric keys are wrapped with master and key-encrypting keys. For DES and AES keys, two methods are detailed. These use the 64-byte token. HMAC keys will use a variable length token with associated data and the payload wrapping method. In the future, all symmetric keys will be able to use the variable length token and the payload wrapping method.

### ECB Wrapping of DES Keys (Original Method)

The wrapping of a double-length key (\*K) using a double-length \*KEK is defined as follows:

$$e*KEK(KL) \parallel e*KEK(KR) = eKEK(dKEK(eKEK(KL))) \parallel eKEK(dKEK(eKEK(KR)))$$

Where:

- KL is the left 64 bits of \*K.
- KR is the right 64 bits of \*K.
- KEKL is the left 64 bits of \*KEK.
- KEKR is the right 64 bits of \*KEK.
- $\parallel$  means concatenation

### CBC Wrapping of AES Keys

The key value in AES tokens are wrapped using the AES algorithm and cipher block chaining (CBC) mode of encryption. The key value is left justified in a 32-byte block, padded on the right with zero and encrypted.

The enhanced wrapping of an AES key (\*K) using an AES \*MK is defined as follows:  $e*MK(*K) = ecbcMK(*K)$

### Enhanced CBC Wrapping of DES Keys (Enhanced Method)

The enhanced CBC wrapping method uses triple DES encryption, an internal chaining of the key value and CBC mode.

The enhanced wrapping of a double-length key (\*K) using a double-length \*KEK is defined as follows:

$$e*KEK(*KL) = ecbcKEK(dcbcKEK(ecbcKEK(KLPRIME \parallel KR)))$$

$$KLPRIME = KL \text{ XOR } SHA1(KR)$$

Where:

- KL is the left 64 bits of \*K.
- KR is the right 64 bits of \*K.
- KLPRIME is the 64 bit modified value of KL
- KEKL is the left 64 bits of \*KEK.
- KEKR is the right 64 bits of \*KEK.
- SHA1(X) is the 160-bit SHA-1 hash of X
- $\parallel$  means concatenation.
- XOR means bitwise exclusive OR

- ecbc means encryption using cipher block chaining mode
- dcbc means decryption using cipher block chaining mode

## Wrapping key derivation for enhanced wrapping of DES keys

The wrapping key is exactly the same key that is used by CCA today, with one exception. Instead of using the base key itself (master key or key-encrypting key), ICSF will use a key that is derived from that base key. The derived key will have the control vector applied to it in the standard CCA manner, and then use the resulting key to wrap the new-format target key token. The reason for using a derived key is to ensure that no attacks against this wrapping scheme are possible using the existing CCA functions. For example, it was observed that an attack was possible by copying the wrapped key into an ECB CCA key token, if the wrapping key was used instead of a derivative of that key.

The key will be derived using a method defined in the NIST standard SP 800-108, "Recommendation for Key Derivation Using Pseudorandom Functions" (October, 2009). Derivation will use the method "KDF in Counter Mode" using pseudorandom function (PRF) HMAC-SHA256. This method provides sufficient strength for deriving keys for any algorithm used.

The HMAC algorithm is defined as follows:

$$\text{HMAC}(K, \text{text}) = H((K0 \text{ XOR opad}) \parallel H((K0 \text{ XOR ipad}) \parallel \text{text}))$$

where opad is the constant 0x5C repeated to form a string the same length as K0, and ipad is the constant 0x36 repeated to form a string the same length as K0. If the key K is equal in length to the input block size of the hash function (512 bits for SHA-256), then K0 is set to the value of K. Otherwise, K0 is formed from K by hashing and/or padding.

The KDF specification calls for inputs optionally including two byte strings, Label and Context. The context will not be used. The label will contain information on the usage of this key, to distinguish it from other derivations that CCA may use in the future for different purposes. Since the security of the derivation process is rooted in the security of the derivation key and in the HMAC and KDF functions themselves, it is not necessary for this label string to be of any particular minimum size. The separation indicator byte of 0x00 specified in the NIST document will follow the label.

The label value will be defined so that it will be unique to derivation for this key wrapping process. This means that in any future designs which use the same KDF, ICSF must use a different value for the label. The label will be the 16 byte value consisting of the following ASCII characters:

```
ENHANCEDWRAP2010 (X'454E4841 4E434544 57524150 32303130')
```

The parameters for the counter mode KDF defined in SP 800-108 are as follows:

- Fixed values:
  - h (length of output of PRF) = 256 bits
  - r (length of the counter, in bits) = 32 - the counter will be an unsigned 4-byte value
- Inputs:
  - KI (input key) will be the key we are deriving from
  - Label will be the value shown above (ASCII ENHANCEDWRAP2010)
  - Separator byte of 0x00 will follow the label value

- Context will be a null string (no context is used)
- L will be the length of the derived key to be produced, rounded up to the next multiple of 256
- PRF (pseudorandom function) will be HMAC-SHA256

The KDF function will produce a pseudorandom bit string that is a multiple of 256 and will use as many bits of that as are required for the key to be produced. Bits for the key will be taken starting from the leftmost bit of the pseudorandom string, and any unused bits at the right will be discarded.

## Variable length token (AESKW method)

The wrapping method for the variable-length key tokens will be AESKW as defined in ANSI X9.102.

The wrapping of the payload of a variable length key (\*K) using an AES \*MK is defined as follows:

$$e^{*MK}(*K) = e^{AESKW^{*MK}}(P)$$

$$P = \text{ICV} || \text{Pad Length} || \text{Hash Length} || \text{Hash options} || \text{Data Hash} || *K || \text{Padding}$$

Where:

- ICV is the 6 byte constant 0xA6A6A6A6A6A6
- Pad length is the length of the Padding in bits
- Hash length is the length of the Data Hash in bytes
- Hash options is a 4-byte field
- Data Hash is the hash of the associated data block
- Padding is the number of bytes, 0x00, to make of the overall length of P a multiple of 16
- eAESKW means encryption using the AESKW method

---

## PKA92 Key Format and Encryption Process

The PKA Symmetric Key Generate and the PKA Symmetric Key Import callable services optionally support a **PKA92** method of encrypting a DES or CDMF key with an RSA public key. This format is adapted from the IBM Transaction Security System (TSS) 4753 and 4755 product's implementation of "PKA92". The callable services do not create or accept the complete PKA92 AS key token as defined for the TSS products. Rather, the callable services only support the actual RSA-encrypted portion of a TSS PKA92 key token, the *AS External Key Block*.

**Forming an AS External Key Block** - The PKA96 implementation forms an AS External Key Block by RSA-encrypting a key block using a public key. The key block is formed by padding the key record detailed in Table 395 on page 948 with zero bits on the left, high-order end of the key record. The process completes the key block with three sub-processes: masking, overwriting, and RSA encrypting.

Table 395. PKA96 Clear DES Key Record

Offset (Bytes)	Length (Bytes)	Description
Zero-bit padding to form a structure as long as the length of the public key modulus. The implementation constrains the public key modulus to a multiple of 64 bits in the range of 512 to 1024 bits. Note that government export or import regulations can impose limits on the modulus length. The maximum length is validated by a check against a value in the Function Control Vector.		
000	005	Header and flags: X'01 0000 0000'
005	016	Environment Identifier (EID), encoded in ASCII
021	008	Control vector base for the DES key
029	008	Repeat of the CV data at offset 021
037	008	The single-length DES key or the left half of a double-length DES key
045	008	The right half of a double-length DES key or a random number. This value is locally designated "K."
053	008	Random number, "IV"
061	001	Ending byte, X'00'

#### Masking Sub-process

1. Form the initial key block by padding the PKR with zero bits on the left, high-order end to the length of the modulus.
2. Create a mask by CBC encrypting a multiple of 8 bytes of binary zeros using K as the key and the length of the modulus, and IV as the initialization vector as defined in the key record at offsets 45 and 53. Exclusive-OR the mask with the key record and call the result PKR.
3. Exclusive-OR the mask with the key block.

#### Overwriting Sub-process

1. Set the high-order bits of PKR to B'01', and set the low-order bits to B'0110'.
2. Exclusive-OR K and IV and write the result at offset 45 in PKR.
3. Write IV at offset 53 in PKR. This causes the masked and overwritten PKR to have IV at its original position.

*Encrypting Sub-process* - RSA encrypt the overwritten PKR masked key record using the public key of the receiving node. This is the last step in creating an AS external key block

**Recovering a Key from an AS External Key Block** - Recover the encrypted DES key from an AS External Key Block by performing decrypting, validating, unmasking, and extraction sub-processes.

*Decrypting Sub-process* - RSA decrypt the AS External Key Block using an RSA private key and call the result of the decryption PKR. The private key must be usable for key management purposes.

*Validating Sub-process* - Verify that the high-order two bits of the decrypted key block are valued to B'01' and that the low-order four bits of the PKR record are valued to B'0110'.



*Unmasking Sub-process* - Set IV to the value of the 8 bytes at offset 53 of the PKR record. Note that there is a variable quantity of padding prior to offset 0. See Table 395 on page 948.

Set K to the exclusive-OR of IV and the value of the 8 bytes at offset 45 of the PKR record.

Create a mask that is equal in length to the key block by CBC encrypting a multiple of 8 bytes of binary zeros using K as the key and IV as the initialization vector. Exclusive-OR the mask with PKR and call the result the key record.

Copy K to offset 45 in the PKR record.

*Extraction Sub-process.* Confirm that:

- The four bytes at offset 1 in the PKR are valued to X'0000 0000'
- The two control vector fields at offsets 21 and 29 are identical
- If the control vector is an IMPORTER or EXPORTER key class, that the EID in the key record is not the same as the EID stored in the cryptographic engine.

The control vector base of the recovered key is the value at offset 21. If the control vector base bits 40 to 42 are valued to B'010' or B'110', the key is double length. Set the right half of the received key's control vector equal to the left half and reverse bits 41 and 42 in the right half.

The recovered key is at offset 37 and is either 8 or 16 bytes long based on the control vector base bits 40 to 42. If these bits are valued to B'000', the key is single length. If these bits are valued to B'010' or B'110', the key is double length.

---

## ANSI X9.17 Partial Notarization Method

The ANSI X9.17 notarization process can be divided into two procedures:

1. *Partial notarization*, in which the ANSI key-encrypting key (AKEK) is cryptographically combined with the origin and destination identifiers.

**Note:** IBM defines this step as partial notarization. The ANSI X9.17 standard does not use the term partial notarization.

2. *Offsetting*, in which the result of the first step is exclusive-ORed with a counter value. ICSF performs the offset procedure to complete the notarization process when you use a partially notarized AKEK.

This appendix describes partial notarization for the ANSI X9.17 notarization process.

### Partial Notarization

Partial notarization improves performance when you use an AKEK for many cryptographic service messages, each with a different counter value.

This section describes the steps in partial notarization. For more information about partial notarization, see “ANSI X9.17 Key Management Services” on page 53. For a description of the steps ICSF uses to complete the notarization of an AKEK or to notarize a key in one process, see *ANSI X9.17 - 1985, Financial Institution Key Management (Wholesale)*.

### Notations Used in the Calculations

\*KK The 16-byte AKEK to be partially notarized



**KKL**    The leftmost 8 bytes of \*KK  
**KKR**    The rightmost 8 bytes of \*KK  
**KK**      The 8-byte AKEK to be partially notarized  
  
**KK1**    An 8-byte intermediate result  
**KK2**    An 8-byte intermediate result  
  
**FMID**   The 16-byte origin identifier  
**FMID1**    The leftmost 8 bytes of FMID  
**FMID2**    The rightmost 8 bytes of FMID  
  
**TOID**   The 16-byte destination identifier  
**TOID1**    The leftmost 8 bytes of TOID  
**TOID2**    The rightmost 8 bytes of TOID  
  
**NSL**    An 8-byte intermediate result  
**NSL1**   The leftmost 4 bytes of NSL  
  
**NSR**    An 8-byte intermediate result  
**NSR2**   The rightmost 4 bytes of NSR  
  
**\*KKNI**    The 16-byte partially notarized AKEK  
**KKNIL**    The leftmost 8 bytes of \*KKNI  
**KKNIR**    The rightmost 8 bytes of \*KKNI  
**KKNI**    The 8-byte partially notarized AKEK  
  
**XOR**    Denotes the exclusive-OR operation  
**TOID1<<1**    Denotes the ASCII TOID1 left-shifted one bit  
**FMID1<<1**    Denotes the ASCII FMID1 left-shifted one bit  
**eK(X)**   Denotes DES encryption of plaintext X using key K  
**||**        Denotes the concatenation operation

### Partial Notarization Calculation for a Double-Length AKEK

For a double-length AKEK, the partial notarization calculation consists of the following steps:

1. Set  $KK1 = KKL \text{ XOR } TOID1<<1$
2. Set  $KK2 = KKR \text{ XOR } FMID1<<1$
3. Set  $NSL = e_{KK2}(TOID2)$
4. Set  $NSR = e_{KK1}(FMID2)$
5. Set  $KKNIL = KKL \text{ XOR } NSL$
6. Set  $KKNIR = KKR \text{ XOR } NSR$
7. Set  $*KKNI = KKNIL || KKNIR$

### Partial Notarization Calculation for a Single-Length AKEK

For a single-length AKEK, the partial notarization calculation consists of the following steps:

1. Set  $KK1 = KK \text{ XOR } TOID1<<1$

2. Set  $KK2 = KK \text{ XOR } FMID1 \ll 1$
3. Set  $NSL = eKK2(TOID2)$
4. Set  $NSR = eKK1(FMID2)$
5. Set  $NSL = NSL1 \parallel NSR2$
6. Set  $KKNI = KK \text{ XOR } NSL$

---

## Transform CDMF Key Algorithm

The CDMF key transformation algorithm uses a 64-bit cryptographic key.

1. Set parity bits of the key to zero by ANDing the key with  $X'FEFEFEFEFEFEFEFE'$  to produce  $Kx$ .
2. Using DES, encipher  $Kx$  under the constant  $K1$ .
3. XOR this value with  $Kx$  to produce  $Ky$ .
4. AND  $Ky$  with  $X'0EFE0EFE0EFE0EFE'$  to produce  $Kz$ .
5. Using DES, encipher  $Kz$  under  $K2$  to produce  $eK2(Kz)$ .
6. Adjust  $eK2(Kz)$  to odd parity in each byte. The result is the transformed key.

The following figure illustrates these steps. (e indicates DES encryption.)

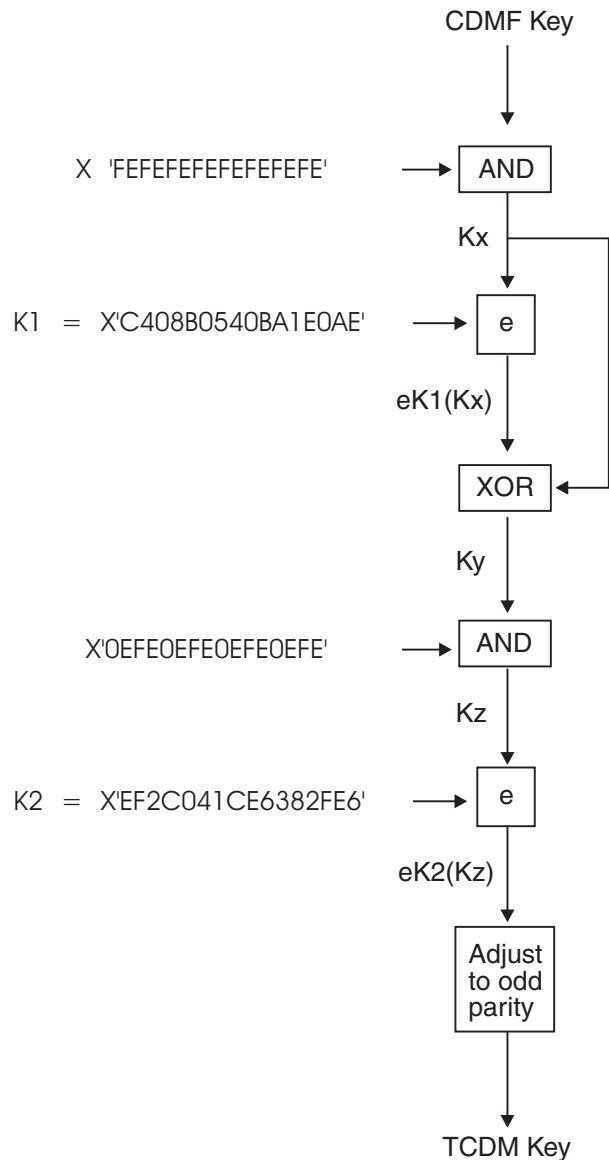


Figure 20. The CDMF Key Transformation Algorithm

## Formatting Hashes and Keys in Public-Key Cryptography

The digital signature generate and digital signature verify callable services support several methods for formatting a hash, and in some cases a descriptor for the hashing method, into a bit-string to be processed by the cryptographic algorithm. This topic discusses the ANSI X9.31 and PKCS #1 methods. The ISO 9796-1 method can be found in the ISO standard.

This topic also describes the PKCS #1, version 1, 1.5, and 2.0, methods for placing a key in a bit string for RSA ciphering in a key exchange.

### ANSI X9.31 Hash Format

With ANSI X9.31, the string that is processed by the RSA algorithm is formatted by the concatenation of a header, padding, the hash and a trailer, from the most

significant bit to the least significant bit, such that the resulting string is the same length as the modulus of the key. For the ICSF implementation, the modulus length must be a multiple of 8 bits.

- The header consists of X'6B'
- The padding consists of X'BB', repeated as many times as required, and terminated by X'BA'
- The hash value follows the padding
- The trailer consists of a hashing mechanism specifier and final byte. These specifiers are defined:
  - X'31': RIPEMD-160
  - X'33': SHA-1
- A final byte of X'CC'.

## PKCS #1 Formats

Version 2.0 of the PKCS #1 standard <sup>5</sup> defines methods for formatting keys and hashes prior to RSA encryption of the resulting data structures. The lower versions of the PKCS #1 standard defined block types 0, 1, and 2, but in the current standard that terminology is dropped.

ICSF implemented these processes using the terminology of the Version 2.0 standard:

- For formatting keys for secured transport (CSNDSYX, CSNDSYG, CSNDSYI):
  - RSAES-OAEP, the preferred method for key-encipherment <sup>6</sup> when exchanging DATA keys between systems. Keyword PKCSOAEP is used to invoke this formatting technique. The P parameter described in the standard is not used and its length is set to zero.
  - RSAES-PKCS1-v1\_5, is an older method for formatting keys. Keyword PKCS-1.2 is used to invoke this formatting technique.
- For formatting hashes for digital signatures (CSNDDSG and CSNDDSV):
  - RSASSA-PKCS1-v1\_5, the newer name for the block-type 1 format. Keyword PKCS-1.1 is used to invoke this formatting technique.
  - The PKCS #1 specification no longer discusses use of block-type 0. Keyword PKCS-1.0 is used to invoke this formatting technique. Use of block-type 0 is discouraged.

Using the terminology from older versions of the PKCS #1 standard, block types 0 and 1 are used to format a hash and block type 2 is used to format a DES key. The blocks consist of (|| means concatenation): X'00' || BT || PS || X'00' D where:

- BT is the block type, X'00', X'01', X'02'.
- PS is the padding of as many bytes as required to make the block the same length as the modulus of the RSA key, and is bytes of X'00' for block type 0, X'01' for block type 1, and random and non-X'00' for block type 2. The length of PS must be at least 8 bytes.
- D is the key, or the concatenation of the BER-encoded hash identifier and the hash.

---

5. PKCS standards can be retrieved from <http://www.rsasecurity.com/rsalabs/pkcs>.

6. The PKA 92 method and the method incorporated into the SET standard are other examples of the Optimal Asymmetric Encryption Padding (OAEP) technique. The OAEP technique is attributed to Bellare and Rogaway.

You can create the BER encoding of an MD5 or SHA-1 value by prepending these strings to the 16 or 20-byte hash values, respectively:

MD5     X'3020300C 06082A86 4886F70D 02050500 0410'  
SHA-1 X'30213009 06052B0E 03021A05 000414'

---

## Visa and EMV-related smart card formats and processes

The VISA and EMV specifications for performing secure messaging with an EMV compliant smart card are covered in these documents:

- *EMV 2000 Integrated Circuit Card Specification for Payment Systems Version 4.0 (EMV4.0) Book 2*
- *Design Visa Integrated Circuit Card Specification Manual*
- *Integrated Circuit Card Specification (VIS) 1.4.0 Corrections*

Book 2, Annex A1.3, describes how a smart-card, card-specific authentication code is derived from a card-issuer-supplied encryption key (ENC-MDK). The *Integrated Circuit Card Specification (VIS) 1.4.0 Corrections* indicates that the key used should be an authentication key (MAC-MDK).

Book 2, Annex A1.3 describes how a smart-card, card-specific session key is derived from a card-issuer-supplied PIN-block-encryption key (ENC-MDK). The encryption key is derived using a "tree-based-derivation" technique. IBM CCA offers two variations of the tree-based technique (TDESEMV2 and TDESEMV4), and a third technique CCA designates TDES-XOR.

In addition, Book 2 describes construction of the PIN block sent to an EMV card to initialize or update the user's PIN.

*Design Visa Integrated Circuit Card Specification Manual*, Annex B.4, contains a description of the session-key derivation technique CCA designates TDES-XOR.

Augmented by the above-mentioned documentation, the relevant processes are described in these sections:

- "Deriving the smart-card-specific authentication code"
- "Constructing the PIN-block for transporting an EMV smart-card PIN"
- "Deriving the CCA TDES-XOR session key" on page 955
- "Deriving the EMV TDESEMVn tree-based session key" on page 955
- "PIN-block self-encryption" on page 956

### Deriving the smart-card-specific authentication code

To ensure that an original or replacement PIN is received from an authorized source, the EMV PIN-transport PIN-block incorporates an authentication code. The authentication code is the rightmost four bytes resulting from the ECB-mode triple-DES encryption of (the first) eight bytes of card-specific data (that is, the rightmost four bytes of the Unique DEA Key A).

### Constructing the PIN-block for transporting an EMV smart-card PIN

The PIN block is used to transport a new PIN value. The PIN block also contains an authentication code, and optionally the "current" PIN value, enabling the smart

card to further ensure receipt of a valid PIN value. To enable incorporation of the PIN block into the a message for an EMV smart-card, the PIN block is padded to 16 bytes prior to encryption.

PINs of length 4 - 12 digits are supported.

PIN-block construction:

1. Form three 8-byte, 16-digit blocks, block-1, block-2, and block-3, and set all digits to X'0'.
2. Replace the rightmost four bytes of block-1 with the authentication code described in the previous section.
3. Set the second digit of block-2 to the length of the new PIN (4 to 12), followed by the new PIN, and padded to the right with X'F'.
4. Include any current PIN by placing it into the leftmost digits of block-3.
5. Exclusive-OR block-1, block-2, and block-3 to form the 8-byte PIN block.
6. Pad the PIN block with other portions of the message for the smart card:
  - Prepend X'08' (the length of the PIN block)
  - Append X'80', followed by 6 bytes of X'00'

The resulting message is ECB-mode triple-encrypted with an appropriate session key.

## Deriving the CCA TDES-XOR session key

In the diversified key generate and PIN change/unblock services, the TDES-XOR process first derives a smart-card-specific intermediate key from the issuer-supplied ENC-MDK key and card-specific data. (This intermediate key is also used in the TDESEMV2 and TDESEMV4 processes. See the next section.) The intermediate key is then modified using the application transaction counter (ATC) value supplied by the smart card.

The double-length session-key creation steps:

1. Obtain the left-half of an intermediate key by ECB-mode triple-DES encrypting the (first) eight bytes of card specific data using the issuer-supplied ENC-MDK key.
2. Again using the ENC-MDK key, obtain the right-half of the intermediate key by ECB-mode triple-DES encrypting:
  - The second 8 bytes of card-specific derivation data when 16 bytes have been supplied
  - The exclusive-OR of the supplied 8 bytes of derivation data with X'FFFFFFFF FFFFFFFF'
3. Pad the ATC value to the left with six bytes of X'00' and exclusive-OR the result with the left-half of the intermediate key to obtain the left-half of the session key.
4. Obtain the one's complement of the ATC by exclusive-ORing the ATC with X'FFFF'. Pad the result on the left with six bytes of X'00'. Exclusive-OR the 8-byte result with the right-half of the intermediate key to obtain the right-half of the session key.

## Deriving the EMV TDESEMVn tree-based session key

In the diversified key generate and PIN change/unblock services, the TDESEMV2 and TDESEMV4 keywords call for the creation of the session key with this process:

1. The intermediate key is obtained as explained above for the TDES-XOR process.
2. Combine the intermediate key with the two-byte Application Transaction Counter (ATC) and an optional Initial Value. The process is defined in the EMV 2000 Integrated Circuit Card Specification for Payment Systems Version 4.0 (EMV4.0) Book 2 Book 2, Annex A1.3.
  - TDESEMV2 causes processing with a branch factor of 2 and a height of 16.
  - TDESEMV4 causes processing with a branch factor of 4 and a height of 8.

## PIN-block self-encryption

In the Secure Messaging for PINs (CSNBSPN and CSNESP) service, you can use the SELFENC rule-array keyword to specify that the 8-byte PIN block shall be used as a DES key to encrypt the PIN block. The verb appends the self-encrypted PIN block to the clear PIN-block in the output message.

---

## Key Test Verification Pattern Algorithms

The key test verification pattern algorithms are:

- The DES algorithm is used by the Key Test callable service to generate and verify the verification pattern.
- The SHAVP1 algorithm is used by the Key Test2 callable service to generate and verify the verification pattern.

## DES Algorithm (single- and double-length keys)

For DES keys, the Key Test callable service uses this algorithm to generate and verify the verification pattern.

$$KK = eC( KL ) \text{ XOR } KL$$

$$VP = eKK( KR \text{ XOR } RN ) \text{ XOR } RN$$

where:

- $eK(x)$  -  $x$  is encrypted by key  $K$  using the DES algorithm
- $KL$  is the left 128-bit clear key value of the key
- $KR$  is the right 128-bit clear key value of the key (will be hex zero for a single length key)
- $C$  is X'4545454545454545'
- $KK$  is a 128-bit intermediate value
- $RN$  is a 128-bit pseudo-random number
- $VP$  is the 128-bit verification pattern

## SHAVP1 Algorithm

This algorithm is used by the Key Test2 callable service to generate and verify the verification pattern.

$$VP = \text{Trunc128}( \text{SHA256}( KA \parallel KT \parallel KL \parallel K ) )$$

Where:

- $VP$  is the 128-bit verification pattern
- $\text{TruncN}(x)$  is truncation of the string  $x$  to the left most  $N$  bits
- $\text{SHA256}(x)$  is the SHA-256 hash of the string  $x$

- KA is the one-byte CCA variable-length key token constant for the algorithm of key (HMAC X'03')
- KT is the two-byte CCA variable-length key token constant for the type of key (MAC X'0002')
- KL is the two-byte bit length of the clear key value
- K is the clear key value left justified and padded on the right with binary zeros to byte boundary || is string concatenation





---

## Appendix G. EBCDIC and ASCII Default Conversion Tables

This section presents tables showing EBCDIC to ASCII and ASCII to EBCDIC conversion tables. In the table headers, EBC refers to EBCDIC and ASC refers to ASCII.

Table 396 shows the EBCDIC to ASCII default conversion table.

*Table 396. EBCDIC to ASCII Default Conversion Table*

EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC
00	00	20	81	40	20	60	2D	80	F8	A0	C8	C0	7B	E0	5C
01	01	21	82	41	A6	61	2F	81	61	A1	7E	C1	41	E1	E7
02	02	22	1C	42	E1	62	DF	82	62	A2	73	C2	42	E2	53
03	03	23	84	43	80	63	DC	83	63	A3	74	C3	43	E3	54
04	CF	24	86	44	EB	64	9A	84	64	A4	75	C4	44	E4	55
05	09	25	0A	45	90	65	DD	85	65	A5	76	C5	45	E5	56
06	D3	26	17	46	9F	66	DE	86	66	A6	77	C6	46	E6	57
07	7F	27	1B	47	E2	67	98	87	67	A7	78	C7	47	E7	58
08	D4	28	89	48	AB	68	9D	88	68	A8	79	C8	48	E8	59
09	D5	29	91	49	8B	69	AC	89	69	A9	7A	C9	49	E9	5A
0A	C3	2A	92	4A	9B	6A	BA	8A	96	AA	EF	CA	CB	EA	A0
0B	0B	2B	95	4B	2E	6B	2C	8B	A4	AB	C0	CB	CA	EB	85
0C	0C	2C	A2	4C	3C	6C	25	8C	F3	AC	DA	CC	BE	EC	8E
0D	0D	2D	05	4D	28	6D	5F	8D	AF	AD	5B	CD	E8	ED	E9
0E	0E	2E	06	4E	2B	6E	3E	8E	AE	AE	F2	CE	EC	EE	E4
0F	0F	2F	07	4F	7C	6F	3F	8F	C5	AF	F9	CF	ED	EF	D1
10	10	30	E0	50	26	70	D7	90	8C	B0	B5	D0	7D	F0	30
11	11	31	EE	51	A9	71	88	91	6A	B1	B6	D1	4A	F1	31
12	12	32	16	52	AA	72	94	92	6B	B2	FD	D2	4B	F2	32
13	13	33	E5	53	9C	73	B0	93	6C	B3	B7	D3	4C	F3	33
14	C7	34	D0	54	DB	74	B1	94	6D	B4	B8	D4	4D	F4	34
15	B4	35	1E	55	A5	75	B2	95	6E	B5	B9	D5	4E	F5	35
16	08	36	EA	56	99	76	FC	96	6F	B6	E6	D6	4F	F6	36
17	C9	37	04	57	E3	77	D6	97	70	B7	BB	D7	50	F7	37
18	18	38	8A	58	A8	78	FB	98	71	B8	BC	D8	51	F8	38
19	19	39	F6	59	9E	79	60	99	72	B9	BD	D9	52	F9	39
1A	CC	3A	C6	5A	21	7A	3A	9A	97	BA	8D	DA	A1	FA	B3
1B	CD	3B	C2	5B	24	7B	23	9B	87	BB	D9	DB	AD	FB	F7
1C	83	3C	14	5C	2A	7C	40	9C	CE	BC	BF	DC	F5	FC	F0
1D	1D	3D	15	5D	29	7D	27	9D	93	BD	5D	DD	F4	FD	FA
1E	D2	3E	C1	5E	3B	7E	3D	9E	F1	BE	D8	DE	A3	FE	A7
1F	1F	3F	1A	5F	5E	7F	22	9F	FE	BF	C4	DF	8F	FF	FF

Table 397 shows the ASCII to EBCDIC default conversion table.

*Table 397. ASCII to EBCDIC Default Conversion Table*

ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC	ASC	EBC
00	00	20	40	40	7C	60	79	80	43	A0	EA	C0	AB	E0	30
01	01	21	5A	41	C1	61	81	81	20	A1	DA	C1	3E	E1	42
02	02	22	7F	42	C2	62	82	82	21	A2	2C	C2	3B	E2	47
03	03	23	7B	43	C3	63	83	83	1C	A3	DE	C3	0A	E3	57
04	37	24	5B	44	C4	64	84	84	23	A4	8B	C4	BF	E4	EE
05	2D	25	6C	45	C5	65	85	85	EB	A5	55	C5	8F	E5	33
06	2E	26	50	46	C6	66	86	86	24	A6	41	C6	3A	E6	B6
07	2F	27	7D	47	C7	67	87	87	9B	A7	FE	C7	14	E7	E1
08	16	28	4D	48	C8	68	88	88	71	A8	58	C8	A0	E8	CD
09	05	29	5D	49	C9	69	89	89	28	A9	51	C9	17	E9	ED
0A	25	2A	5C	4A	D1	6A	91	8A	38	AA	52	CA	CB	EA	36
0B	0B	2B	4E	4B	D2	6B	92	8B	49	AB	48	CB	CA	EB	44
0C	0C	2C	6B	4C	D3	6C	93	8C	90	AC	69	CC	1A	EC	CE
0D	0D	2D	60	4D	D4	6D	94	8D	BA	AD	DB	CD	1B	ED	CF
0E	0E	2E	4B	4E	D5	6E	95	8E	EC	AE	8E	CE	9C	EE	31
0F	0F	2F	61	4F	D6	6F	96	8F	DF	AF	8D	CF	04	EF	AA
10	10	30	F0	50	D7	70	97	90	45	B0	73	D0	34	F0	FC
11	11	31	F1	51	D8	71	98	91	29	B1	74	D1	EF	F1	9E
12	12	32	F2	52	D9	72	99	92	2A	B2	75	D2	1E	F2	AE
13	13	33	F3	53	E2	73	A2	93	9D	B3	FA	D3	06	F3	8C
14	3C	34	F4	54	E3	74	A3	94	72	B4	15	D4	08	F4	DD
15	3D	35	F5	55	E4	75	A4	95	2B	B5	B0	D5	09	F5	DC
16	32	36	F6	56	E5	76	A5	96	8A	B6	B1	D6	77	F6	39
17	26	37	F7	57	E6	77	A6	97	9A	B7	B3	D7	70	F7	FB
18	18	38	F8	58	E7	78	A7	98	67	B8	B4	D8	BE	F8	80
19	19	39	F9	59	E8	79	A8	99	56	B9	B5	D9	BB	F9	AF
1A	3F	3A	7A	5A	E9	7A	A9	9A	64	BA	6A	DA	AC	FA	FD
1B	27	3B	5E	5B	AD	7B	C0	9B	4A	BB	B7	DB	54	FB	78
1C	22	3C	4C	5C	E0	7C	4F	9C	53	BC	B8	DC	63	FC	76
1D	1D	3D	7E	5D	BD	7D	D0	9D	68	BD	B9	DD	65	FD	B2
1E	35	3E	6E	5E	5F	7E	A1	9E	59	BE	CC	DE	66	FE	9F
1F	1F	3F	6F	5F	6D	7F	07	9F	46	BF	BC	DF	62	FF	FF

## Appendix H. Access Control Points and Callable Services

Access to callable services that are executed on the PCI Cryptographic Coprocessor, PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, Crypto Express3 Coprocessor, or Crypto Express4 Coprocessor is through Access Control Points in the ICSF Role. To execute services on the coprocessor, access control points must be enabled for each service in the ICSF Role. The access control points available depend on the coprocessor you are using.

The TKE workstation allows you to enable or disable access control points. For systems that do not use the optional TKE Workstation, most access control points (current and new) are enabled in the ICSF Role with the appropriate licensed internal code on the coprocessor. The table of access control points lists the default setting of each access control point.

New TKE users and non-TKE users have the default set of access control points enabled. For existing TKE users who have changed the setting of any access control point, any new access control points will not be enabled.

**Note:** Access control points for ICSF utilities are listed in *z/OS Cryptographic Services ICSF Administrator's Guide*.

If an access control point is disabled, the corresponding ICSF callable service will fail during execution with an access denied error.

The following tables list usage information using the following abbreviations:

**AE** Always enabled, can not be disabled.

**ED** Enabled by default.

**DD** Disabled by default.

**SC** Usage of this access control point requires special consideration.

This table lists access control points that affect multiple services or have require special consideration when enabling the access control point.

*Table 398. Access control points affecting multiple services or requiring special consideration*

Name	Callable Services	Notes	Usage
Allow weak DES wrap of RSA	CSNDPKG / CSNFPKG	When enabled, a weaker DES key-encrypting key is allowed to wrap an RSA private key token. The Prohibit weak wrap – Transport keys access control point must be enabled and this access control point will override the restriction. See “Key Strength and Wrapping of Key” on page 85 for more information.	DD, SC
ANSI X9.8 PIN - Allow modification of PAN	CSNBPTR / CSNEPTR	See “ANSI X9.8 PIN Restrictions” on page 469 for a description of this control.	DD, SC
ANSI X9.8 PIN - Allow only ANSI PIN blocks	CSNBPTR / CSNEPTR	See “ANSI X9.8 PIN Restrictions” on page 469 for a description of this control.	DD, SC

Table 398. Access control points affecting multiple services or requiring special consideration (continued)

Name	Callable Services	Notes	Usage
ANSI X9.8 PIN - Enforce PIN block restrictions	CSNBCPA / CSNECPA, CSNBPTR / CSNEPTR, and CSNBSPN / CSNESP	See “ANSI X9.8 PIN Restrictions” on page 469 for a description of this control.	DD, SC
ANSI X9.8 PIN – Use stored decimalization tables only	CSNBPGN / CSNEPGN, CSNBCPA / CSNECPA, CSNBEPG / CSNEEPG and CSNBPVR / CSNEPVR	See “ANSI X9.8 PIN Restrictions” on page 469 for a description of this control.	DD, SC
DATAM Key Management Control	CSNBKGN / CSNEKGN, CSNBKIM / CSNEKIM, CSNBKEX / CSNEKEX and CSNBDKG / CSNEDKG	When enabled, the DATAM and DATAMV key types can be used. When disabled, the key types are not allowed.	ED
Disallow 24-byte DATA wrapped with 16-byte Key	All callable services that wrap key under an exporter or importer KEK or a 16-byte DES master key	When enabled, a triple-length 0 CV DATA keys can not be wrapped by a 16-byte DES Key, either the master key or a key-encrypting key. See “Key Strength and Wrapping of Key” on page 25 for more information.	DD, SC
Enhanced PIN Security	CSNBCPE / CSNECPE, CSNBCPA / CSNECPA, CSNBEPG / CSNEEPG, CSNBPTR / CSNEPTR, CSNBPVR / CSNEPVR, and CSNBPCU / CSNEPCU	“Enhanced PIN Security Mode” on page 473 for a description of this control	DD, SC
NOCV KEK usage for export-related functions	CSNBKEX / CSNEKEX, CSNBSKM / CSNESKM, and CSNBKGN / CSNEKGN	When enable, NOVC key-encrypting keys can be used by the listed services.	ED, SC

Table 398. Access control points affecting multiple services or requiring special consideration (continued)

Name	Callable Services	Notes	Usage
NOCV KEK usage for import-related functions	CSNBKIM / CSNEKIM, CSNBSKI / CSNESKI, CSNBSKM / CSNESKM, and CSNBKGN / CSNEKGN	When enable, NOVC key-encrypting keys can be used by the listed services.	ED, SC
Prohibit weak wrap – Master keys	All Services that wrap or import keys. Both symmetric and asymmetric keys are affected	When enabled, an error return code will be returned when attempting to wrap a stronger key with a weaker master key. Also, an error return code will be returned when the last part is loaded into the DES or RSA new master key register, if the complete master key is weak. See “Key Strength and Wrapping of Key” on page 25 and “Key Strength and Wrapping of Key” on page 85 for more information.	DD, SC
Prohibit weak wrap – Transport keys	All Services that wrap or import keys. Both symmetric and asymmetric keys are affected	When enabled, an error return code will be returned when attempting to wrap a stronger key with a weaker key-encrypting key. See “Key Strength and Wrapping of Key” on page 25 for more information.	DD, SC
Symmetric Key Token Change – RTCMK	Services that use symmetric key tokens	When enabled, this control allows symmetric key tokens under the old master key to be reenciphered under the current master key. These reenciphered tokens are returned from all callable service that use symmetric tokens.	AE
Symmetric Key Token Change2 – RTCMK	Services that use the variable-length symmetric key tokens	When enabled, this control allows symmetric key tokens under the old master key to be reenciphered under the current master key. These reenciphered tokens are returned from all callable service that use symmetric tokens.	AE
Symmetric token wrapping - internal enhanced method	Services that wrap internal symmetric key tokens	When enabled, this control will cause all generated or imported keys to be wrapped with the enhanced method. This control can be overridden by rule array keywords for certain services. See “Key Strength and Wrapping of Key” on page 25 for more information.	DD, SC
Symmetric token wrapping - internal original method	Services that wrap internal symmetric key tokens	When enabled, this control will cause all generated or imported keys to be wrapped with the original method. This control can be overridden by rule array keywords for certain services. See “Key Strength and Wrapping of Key” on page 25 for more information.	ED
Symmetric token wrapping - external enhanced method	Services that wrap external symmetric key tokens	When enabled, this control will cause all generated or exported keys to be wrapped with the enhanced method. This control can be overridden by rule array keywords for certain services. See “Key Strength and Wrapping of Key” on page 25 for more information.	DD, SC

Table 398. Access control points affecting multiple services or requiring special consideration (continued)

Name	Callable Services	Notes	Usage
Symmetric token wrapping - external original method	Services that wrap external symmetric key tokens	When enabled, this control will cause all generated or exported keys to be wrapped with the original method. This control can be overridden by rule array keywords for certain services. See "Key Strength and Wrapping of Key" on page 25 for more information.	ED
UKPT - PIN Verify, PIN Translate	CSNBPVR / CSNEPVR and CSNBPTR / CSNEPTR	When enabled, the listed services can use UKPT key derivation.	ED
Warn when weak wrap – Master keys	All Services that wrap or import keys. Both symmetric and asymmetric keys are affected	When enabled, an informational return code will be returned when attempting to wrap a stronger key with a master key that is weaker. Also, a warning return code will be returned when the last part is loaded into the DES or RSA new master key register, if the master key is weak. See "Key Strength and Wrapping of Key" on page 25 and "Key Strength and Wrapping of Key" on page 85 for more information.	DD. SC
Warn when weak wrap – Key-encrypting keys	All Services that wrap or import keys. Both symmetric and asymmetric keys are affected	When enabled, an informational return code will be returned when attempting to wrap a stronger key with a weaker key or when attempting to import a key token that has previously been wrapped with a weaker key, as indicated by its security history field. See "Key Strength and Wrapping of Key" on page 25 and "Key Strength and Wrapping of Key" on page 85 for more information.	DD. SC

There are relationships between certain access control points. A controlling access control point is required to be enabled before subordinate access control points can be enabled. The TKE workstation will enable the controlling access control point when a subordinate access control point is enabled.

- The Allow weak DES wrap of RSA access control point is only checked if the Prohibit weak wrap – Transport keys access control point is enabled.
- The ANSI X9.8 PIN - Allow modification of PAN and ANSI X9.8 PIN - Allow only ANSI PIN blocks access control points can only be enabled when the ANSI X9.8 PIN - Enforce PIN block restrictions access control point is enabled.

This next table lists access control points that affect specific services indicated in the access control point name. There is a description of the usage of the access control point in the Usage Notes section of the callable service description.

The following abbreviations and symbols are used in this table:

**Note:** If the ICSF role has been changed via the TKE workstation, all new access control points are disabled by default.

Table 399. Access control points – Callable Services

Name	Callable Service	Usage
Cipher Text translate2	CSNBCTT2 / CSNECTT2 and CSNBCTT3 / CSNECTT3	ED
Cipher Text translate2 – Allow translate from AES to TDES	CSNBCTT2 / CSNECTT2 and CSNBCTT3 / CSNECTT3	ED
Cipher Text translate2 – Allow translate to weaker AES	CSNBCTT2 / CSNECTT2 and CSNBCTT3 / CSNECTT3	ED
Cipher Text translate2 – Allow translate to weaker DES	CSNBCTT2 / CSNECTT2 and CSNBCTT3 / CSNECTT3	ED
Cipher Text translate2 – Allow only cipher text translate types	CSNBCTT2 / CSNECTT2 and CSNBCTT3 / CSNECTT3	DD
Clear Key Import / Multiple Clear Key Import - DES	CSNBCKI / CSNECKI and CSNBCKM / CSNECKM	ED
Clear PIN Encrypt	CSNBCPE / CSNECPE	ED
Clear PIN Generate - 3624	CSNBPGN / CSNEPGN	ED
Clear PIN Generate - GBP	CSNBPGN / CSNEPGN	ED
Clear PIN Generate - VISA PVV	CSNBPGN / CSNEPGN	ED
Clear PIN Generate - Interbank	CSNBPGN / CSNEPGN	ED
Clear Pin Generate Alternate - 3624 Offset	CSNBCPA / CSNECPA	ED
Clear PIN Generate Alternate - VISA PVV	CSNBCPA / CSNECPA	ED
Control Vector Translate	CSNBCVT / CSNECVT	ED
Cryptographic Variable Encipher	CSNBCVE / CSNECVE	ED
CVV Key Combine	CSNBCKC / CSNECKC	ED
CVV Key Combine - Allow wrapping override keywords	CSNBCKC / CSNECKC	ED
CVV Key Combine - Permit mixed key types	CSNBCKC / CSNECKC	ED
Data Key Export	CSNBDKX / CSNEDKX	ED
Data Key Export - Unrestricted	CSNBDKX / CSNEDKX	ED
Data Key Import	CSNBDKM / CSNEDKM	ED
Data Key Import - Unrestricted	CSNBDKM / CSNEDKM	ED
Decipher - DES	CSNBDEC / CSNEDEC	ED
Digital Signature Generate	CSNDDSG / CSNFDSG	ED
DSG - ZERO-PAD restriction lifted	CSNDDSG / CSNFDSG	ED
Digital Signature Verify	CSNDDSV / CSNFDSV	ED
Diversified Key Generate - CLR8-ENC	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - SESS-XOR	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - TDES-ENC	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - TDES-DEC	CSNBDKG / CSNEDKG	ED



Table 399. Access control points – Callable Services (continued)

Name	Callable Service	Usage
Diversified Key Generate - TDES-XOR	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - TDESEMV2/TDESEMV4	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - Allow wrapping override keywords	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - single length or same halves	CSNBDKG / CSNEDKG	ED
Diversified Key Generate - DKYGENKY - DALL	CSNBDKG / CSNEDKG	DD, SC
ECC Diffie-Hellman	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow Prime Curve 192	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow Prime Curve 224	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow Prime Curve 256	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow Prime Curve 384	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow Prime Curve 521	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow BP Curve 160	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow BP Curve 192	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow BP Curve 224	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow BP Curve 256	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow BP Curve 320	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow BP Curve 384	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow BP Curve 512	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow PASSTHRU	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Allow key wrap override	CSNDEDH / CSNFEDH	ED
ECC Diffie-Hellman – Prohibit weak key generate	CSNDEDH / CSNFEDH	DD, SC
Encipher - DES	CSNBENC / CSNEENC	ED
Encrypted PIN Generate - 3624	CSNBEPG / CSNEEPG	ED
Encrypted PIN Generate - GBP	CSNBEPG / CSNEEPG	ED
Encrypted PIN Generate - Interbank	CSNBEPG / CSNEEPG	ED

Table 399. Access control points – Callable Services (continued)

Name	Callable Service	Usage
Encrypted PIN Translate - Translate	CSNBPTR / CSNEPTR	ED
Encrypted PIN Translate - Reformat	CSNBPTR / CSNEPTR	ED
Encrypted PIN Verify - 3624	CSNBPVR / CSNEPVR	ED
Encrypted PIN Verify - GPB	CSNBPVR / CSNEPVR	ED
Encrypted PIN Verify - VISA PVV	CSNBPVR / CSNEPVR	ED
Encrypted PIN Verify - Interbank	CSNBPVR / CSNEPVR	ED
HMAC Generate – SHA-1	CSNBHMG / CSNBHMG1 and CSNEHMG / CSNEHMG1	ED
HMAC Generate – SHA-224	CSNBHMG / CSNBHMG1 and CSNEHMG / CSNEHMG1	ED
HMAC Generate – SHA-256	CSNBHMG / CSNBHMG1 and CSNEHMG / CSNEHMG1	ED
HMAC Generate – SHA-384	CSNBHMG / CSNBHMG1 and CSNEHMG / CSNEHMG1	ED
HMAC Generate – SHA-512	CSNBHMG / CSNBHMG1 and CSNEHMG / CSNEHMG1	ED
HMAC Verify – SHA-1	CSNBHMGV / CSNBHMGV1 and CSNEHMGV / CSNEHMGV1	ED
HMAC Verify – SHA-224	CSNBHMGV / CSNBHMGV1 and CSNEHMGV / CSNEHMGV1	ED
HMAC Verify – SHA-256	CSNBHMGV / CSNBHMGV1 and CSNEHMGV / CSNEHMGV1	ED
HMAC Verify – SHA-384	CSNBHMGV / CSNBHMGV1 and CSNEHMGV / CSNEHMGV1	ED
HMAC Verify – SHA-512	CSNBHMGV / CSNBHMGV1 and CSNEHMGV / CSNEHMGV1	ED
Key Export	CSNBKEX / CSNEKEX	ED
Key Export - Unrestricted	CSNBKEX / CSNEKEX	ED
Key Generate – OP	CSNBKGN / CSNEKGN	ED
Key Generate – Key set	CSNBKGN / CSNEKGN	ED
Key Generate – Key set extended	CSNBKGN / CSNEKGN	ED
Key Generate - SINGLE-R	CSNBKGN / CSNEKGN	ED
Key Generate2 – OP	CSNBKGN2 / CSNEKGN2	ED
Key Generate2 – Key set	CSNBKGN2 / CSNEKGN2	ED
Key Generate2 – Key set extended	CSNBKGN2 / CSNEKGN2	ED
Key Import	CSNBKIM / CSNEKIM	ED
Key Import - Unrestricted	CSNBKIM / CSNEKIM	ED
Key Part Import - First key part	CSNBKPI / CSNEKPI	ED
Key Part Import - Middle and final	CSNBKPI / CSNEKPI	ED
Key Part Import - ADD-PART	CSNBKPI / CSNEKPI	ED
Key Part Import - COMPLETE	CSNBKPI / CSNEKPI	ED
Key Part Import - Allow wrapping override keywords	CSNBKPI / CSNEKPI	ED
Key Part Import - Unrestricted	CSNBKPI / CSNEKPI	ED
Key Part Import2 – Load first key part, require 3 key parts	CSNBKPI2 / CSNEKPI2	ED
Key Part Import2 – Load first key part, require 2 key parts	CSNBKPI2 / CSNEKPI2	ED
Key Part Import2 - Load first key part, require 1 key parts	CSNBKPI2 / CSNEKPI2	ED

Table 399. Access control points – Callable Services (continued)

Name	Callable Service	Usage
Key Part Import2 - Add second of 3 or more key parts	CSNBKPI2 / CSNEKPI2	ED
Key Part Import2 - Add last required key part	CSNBKPI2 / CSNEKPI2	ED
Key Part Import2 - Add optional key part	CSNBKPI2 / CSNEKPI2	ED
Key Part Import2 – Complete key	CSNBKPI2 / CSNEKPI2	ED
Key Test and Key Test2	CSNBKYT / CSNEKYT and CSNBKYT2 / CSNEKYT2	AE
Key Test2 – AES, ENC-ZERO	CSNBKYT2 / CSNEKYT2	AE
Key Translate	CSNBKTR / CSNEKTR	ED
Key Translate2	CSNBKTR2 / CSNEKTR2	ED
Key Translate2 - Allow use of REFORMAT	CSNBKTR2 / CSNEKTR2	ED
Key Translate2 - Allow wrapping override keywords	CSNBKTR2 / CSNEKTR2	ED
Key Translate2 - Disallow AES ver 5 to ver 4 conversion	CSNBKTR2 / CSNEKTR2	DD
MAC Generate	CSNBMGN / CSNEMGN	ED
MAC Verify	CSNBMVR / CSNEMVR	ED
Multiple Clear Key Import / Multiple Secure Key Import - AES	CSNBCKM / CSNECKM and CSNBSKM / CSNESKM	ED
Multiple Clear Key Import - Allow wrapping override keywords	CSNBCKM / CSNECKM	ED
Multiple Secure Key Import - Allow wrapping override keywords	CSNBSKM / CSNESKM	ED
Operational Key Load	CSNBOKL / CSNEOKL	ED
Operational Key Load2	CSNBOKL / CSNEOKL	ED
PIN Change/Unblock - change EMV PIN with OPINENC	CSNBPCU / CSNEPCU	ED
PIN Change/Unblock - change EMV PIN with IPINENC	CSNBPCU / CSNEPCU	ED
PKA Decrypt	CSNDPKD / CSNFPKD	ED
PKA Encrypt	CSNDPKE / CSNFPKE	ED
PKA Key Generate	CSNDPKG / CSNFPKG	ED
PKA Key Generate – Clear RSA keys	CSNDPKG / CSNFPKG	ED
PKA Key Generate – Clear ECC keys	CSNDPKG / CSNFPKG	ED
PKA Key Generate - Clone	CSNDPKG / CSNFPKG	ED
PKA Key Generate - Permit Regeneration Data	CSNDPKG / CSNFPKG	ED
PKA Key Generate - Permit Regeneration Data Retain	CSNDPKG / CSNFPKG	ED
PKA Key Import	CSNDPKI / CSNFPKI	ED

Table 399. Access control points – Callable Services (continued)

Name	Callable Service	Usage
PKA Key Import - Import an External Trusted Key Block to internal form	CSNDPKI / CSNFPKI	ED
PKA Key Token Change RTCMK	CSNDKTC / CSNFKTC	ED
PKA Key Translate - from CCA RSA to SC Visa Format	CSNDPKT / CSNFPKT	ED
PKA Key Translate - from CCA RSA to SC ME Format	CSNDPKT / CSNFPKT	ED
PKA Key Translate - from CCA RSA to SC CRT Format	CSNDPKT / CSNFPKT	ED
PKA Key Translate – Translate internal key token	CSNDPKT / CSNFPKT	ED
PKA Key Translate – Translate external key token	CSNDPKT / CSNFPKT	ED
PKA Key Translate - from source EXP KEK to target EXP KEK	CSNDPKT / CSNFPKT	ED
PKA Key Translate - from source IMP KEK to target EXP KEK	CSNDPKT / CSNFPKT	ED
PKA Key Translate - from source IMP KEK to target IMP KEK	CSNDPKT / CSNFPKT	ED
Prohibit Export	CSNBPEX / CSNEPEX	ED
Prohibit Export Extended	CSNBPEXX / CSNEPEXX	ED
Remote Key Export - Generate or export a key for use by a non-CCA node	CSNDRKX / CSNFRKX	ED
RKX/TBC – Disallow triple-length MAC key	CSNDRKX / CSNFRKX and CSNDTBC / CSNFTBC	DD, SC
Restrict Key Attribute – Export Control	CSNBRKA / CSNERKA	ED
Restrict Key Attribute - Permit setting the TR-31 export bit	CSNBRKA / CSNERKA	ED
Retained Key Delete	CSNDRKD / CSNFRKD	ED
Retained Key List	CSNDRKL / CSNFRKL	ED
Secure Key Import – DES, IM	CSNBSKI / CSNESKI and CSNBSKM / CSNESKM	ED
Secure Key Import – DES, OP	CSNBSKI / CSNESKI and CSNBSKM / CSNESKM	ED
Secure Key Import2 - OP	CSNBSKI2 / CSNESKI2	ED
Secure Key Import2 - IM	CSNBSKI2 / CSNESKI2	ED
Secure Messaging for Keys	CSNBSKY / CSNESKY	ED
Secure Messaging for PINs	CSNBSPN / CSNESPEN	ED
SET Block Compose	CSNDSBC / CSNFSBC	ED
SET Block Decompose	CSNDSBD / CSNFSBD	ED
SET Block Decompose - PIN ext IPINENC	CSNDSBD / CSNFSBD	ED
SET Block Decompose - PIN ext OPINENC	CSNDSBD / CSNFSBD	ED

Table 399. Access control points – Callable Services (continued)

Name	Callable Service	Usage
Symmetric Algorithm Decipher - Secure AES	CSNBSAD / CSNESAD and CSNBSAD1 / CSNESAD1	ED
Symmetric Algorithm Encipher - Secure AES	CSNBSAE / CSNESAE and CSNBSAE1 / CSNESAE1	ED
Symmetric Key Encipher/Decipher - Encrypted DES keys	CSNBSYD / CSNBSYE and CSNBSYD1 / CSNESYD1	ED
Symmetric Key Encipher/Decipher - Encrypted AES keys	CSNBSYD / CSNBSYE and CSNBSYD1 / CSNESYD1	ED
Symmetric Key Export - AES, PKCSOAEP, PKCS-1.2	CSNDSYX / CSNFSYX	ED
Symmetric Key Export - AES, PKOAEP2	CSNDSYX / CSNFSYX	ED
Symmetric Key Export - AES, ZERO-PAD	CSNDSYX / CSNFSYX	ED
Symmetric Key Export - AESKW	CSNDSYX / CSNFSYX	ED
Symmetric Key Export - DES, PKCS-1.2	CSNDSYX / CSNFSYX	ED
Symmetric Key Export - DES, ZERO-PAD	CSNDSYX / CSNFSYX	ED
Symmetric Key Export – HMAC,PKOAEP2	CSNDSYX / CSNFSYX	ED
Symmetric Key Generate - AES, PKCSOAEP, PKCS-1.2	CSNDSYG / CSNFSYG	ED
Symmetric Key Generate - AES, ZERO-PAD	CSNDSYG / CSNFSYG	ED
Symmetric Key Generate - DES, PKCS-1.2	CSNDSYG / CSNFSYG	ED
Symmetric Key Generate - DES, ZERO-PAD	CSNDSYG / CSNFSYG	ED
Symmetric Key Generate - DES, PKA92	CSNDSYG / CSNFSYG	ED
Symmetric Key Generate - Allow wrapping override keywords	CSNDSYG / CSNFSYG	ED
Symmetric Key Import - AES, PKCSOAEP, PKCS-1.2	CSNDSYI / CSNFSYI	ED
Symmetric Key Import - AES, ZERO-PAD	CSNDSYI / CSNFSYI	ED
Symmetric Key Import - DES, PKCS-1.2	CSNDSYI / CSNFSYI	ED
Symmetric Key Import - DES, ZERO-PAD	CSNDSYI / CSNFSYI	ED
Symmetric Key Import - DES, PKA92 KEK	CSNDSYI / CSNFSYI	ED
Symmetric Key Import - Allow wrapping override keywords	CSNDSYI / CSNFSYI	ED
Symmetric Key Import2 – AES,PKOAEP2	CSNDSYI2 / CSNFSYI2	ED

Table 399. Access control points – Callable Services (continued)

Name	Callable Service	Usage
Symmetric Key Import2 – HMAC,PKOAEP2	CSNDSYI2 / CSNFSYI2	ED
Symmetric Key Import2 - AESKW	CSNDSYI2 / CSNFSYI2	ED
Symmetric Key Import2 - disallow weak import	CSNDSYI2 / CSNFSYI2	DD, SC
TR31 Export – Permit version A TR-31 key blocks	CSNBT31X / CSNET31X	ED
TR31 Export – Permit version B TR-31 key blocks	CSNBT31X / CSNET31X	ED
TR31 Export – Permit version C TR-31 key blocks	CSNBT31X / CSNET31X	ED
TR31 Export – Permit any CCA key if INCL-CV is specified	CSNBT31X / CSNET31X	ED
TR31 Export – Permit KEYGENKY:UKPT to B0	CSNBT31X / CSNET31X	ED
TR31 Export – Permit MAC/MACVER:AMEXCSC to C0:G/C/V	CSNBT31X / CSNET31X	DD
TR31 Export – Permit MAC/MACVER:CVVKEYA to C0:G/C/V	CSNBT31X / CSNET31X	DD
TR31 Export – Permit MAC/MACVER:ANYMAC to C0:G/C/V	CSNBT31X / CSNET31X	ED
TR31 Export – Permit DATA to C0:G/C	CSNBT31X / CSNET31X	ED
TR31 Export – Permit ENCIPHER/DECIPHER/CIPHER to D0:E/D/B	CSNBT31X / CSNET31X	ED
TR31 Export – Permit DATA to D0:B	CSNBT31X / CSNET31X	ED
TR31 Export – Permit EXPORTER/OKEYXLAT to K0:E	CSNBT31X / CSNET31X	DD
TR31 Export – Permit IMPORTER/IKEYXLAT to K0:D	CSNBT31X / CSNET31X	DD
TR31 Export – Permit EXPORTER/OKEYXLAT to K1:E	CSNBT31X / CSNET31X	DD
TR31 Export – Permit IMPORTER/IKEYXLAT to K1:D	CSNBT31X / CSNET31X	DD
TR31 Export – Permit MAC/DATA/DATAM to M0:G/C	CSNBT31X / CSNET31X	DD
TR31 Export – Permit MACVER/DATAMV to M0:V	CSNBT31X / CSNET31X	ED
TR31 Export – Permit MAC/DATA/DATAM to M1:G/C	CSNBT31X / CSNET31X	ED
TR31 Export – Permit MACVER/DATAMV to M1:V	CSNBT31X / CSNET31X	ED

Table 399. Access control points – Callable Services (continued)

Name	Callable Service	Usage
TR31 Export – Permit MAC/DATA/DATAM to M3:G/C	CSNBT31X / CSNET31X	ED
TR31 Export – Permit MACVER/DATAMV to M3:V	CSNBT31X / CSNET31X	ED
TR31 Export – Permit OPINENC to P0/E	CSNBT31X / CSNET31X	ED
TR31 Export – Permit IPINENC to P0/D	CSNBT31X / CSNET31X	ED
TR31 Export – Permit PINVER:NO-SPEC to V0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit PINGEN:NO-SPEC to V0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit PINVER:NO-SPEC/IBM-PIN/IBM- PINO to V1	CSNBT31X / CSNET31X	ED
TR31 Export – Permit PINGEN:NO-SPEC/IBM-PIN/ IBM-PINO to V1	CSNBT31X / CSNET31X	ED
TR31 Export – Permit PINVER:NO-SPEC/VISA-PVV to V2	CSNBT31X / CSNET31X	ED
TR31 Export – Permit PINGEN:NO-SPEC/VISA-PVV to V2	CSNBT31X / CSNET31X	ED
TR31 Export – Permit DKYGENKY:DKYL0+DMAC to E0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DMV to E0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DALL to E0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DMAC to E0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DMV to E0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DALL to E0	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DDATA to E1	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DMPIN to E1	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DALL to E1	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DDATA to E1	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DMPIN to E1	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DALL to E1	CSNBT31X / CSNET31X	DD

Table 399. Access control points – Callable Services (continued)

Name	Callable Service	Usage
TR31 Export – Permit DKYGENKY:DKYL0+DMAC to E2	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DALL to E2	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DMAC to E2	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL1+DALL to E2	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DATA/MAC/CIPHER/ENCIPHER to E3	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DDATA to E4	CSNBT31X / CSNET31X	ED
TR31 Export – Permit DKYGENKY:DKYL0+DALL to E4	CSNBT31X / CSNET31X	ED
TR31 Export – Permit DKYGENKY:DKYL0+DEXP to E5	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DMAC to E5	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DDATA to E5	CSNBT31X / CSNET31X	DD
TR31 Export – Permit DKYGENKY:DKYL0+DALL to E5	CSNBT31X / CSNET31X	ED
TR31 Export – Permit PINGEN/PINVER to V0/V1/V2:N	CSNBT31X / CSNET31X	DD
TR31 Import – Permit version A TR-31 key blocks	CSNBT31I / CSNET31I	ED
TR31 Import – Permit version B TR-31 key blocks	CSNBT31I / CSNET31I	ED
TR31 Import – Permit version C TR-31 key blocks	CSNBT31I / CSNET31I	ED
TR31 Import – Permit override of default wrapping method	CSNBT31I / CSNET31I	ED
TR31 Import – Permit C0 to MAC/MACVER:CVVKEY-A	CSNBT31I / CSNET31I	DD
TR31 Import – Permit C0 to MAC/MACVER:AMEX-CSC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K0:E to EXPORTER/OKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K0:D to IMPORTER/IKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K0:B to EXPORTER/OKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K0:B to IMPORTER/IKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K1:E to EXPORTER/OKEYXLAT	CSNBT31I / CSNET31I	DD



Table 399. Access control points – Callable Services (continued)

Name	Callable Service	Usage
TR31 Import – Permit K1:D to IMPORTER/IKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K1:B to EXPORTER/OKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit K1:B to IMPORTER/IKEYXLAT	CSNBT31I / CSNET31I	DD
TR31 Import – Permit M0/M1/M3 to MAC/MACVER:ANY-MAC	CSNBT31I / CSNET31I	ED
TR31 Import – Permit P0:E to OPINENC	CSNBT31I / CSNET31I	ED
TR31 Import – Permit P0:D to IPINENC	CSNBT31I / CSNET31I	ED
TR31 Import – Permit V0 to PINGEN:NO-SPEC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit V0 to PINVER:NO-SPEC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit V1 to PINGEN:IBM-PIN/IBM-PINO	CSNBT31I / CSNET31I	ED
TR31 Import – Permit V1 to PINVER:IBM-PIN/IBM-PINO	CSNBT31I / CSNET31I	ED
TR31 Import – Permit V2 to PINGEN:VISA-PVV	CSNBT31I / CSNET31I	ED
TR31 Import – Permit V2 to PINVER:VISA-PVV	CSNBT31I / CSNET31I	ED
TR31 Import – Permit E0 to DKYGENKY:DKYL0+DMAC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E0 to DKYGENKY:DKYL0+DMV	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E0 to DKYGENKY:DKYL1+DMAC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E0 to DKYGENKY:DKYL1+DMV	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E1 to DKYGENKY:DKYL0+DMPIN	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E1 to DKYGENKY:DKYL0+DDATA	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E1 to DKYGENKY:DKYL1+DMPIN	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E1 to DKYGENKY:DKYL1+DDATA	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E2 to DKYGENKY:DKYL0+DMAC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E2 to DKYGENKY:DKYL1+DMAC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E3 to ENCIPHER	CSNBT31I / CSNET31I	DD

Table 399. Access control points – Callable Services (continued)

Name	Callable Service	Usage
TR31 Import – Permit E4 to DKYGENKY:DKYL0+DDATA	CSNBT31I / CSNET31I	ED
TR31 Import – Permit E5 to DKYGENKY:DKYL0+DMAC	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E5 to DKYGENKY:DKYL0+DDATA	CSNBT31I / CSNET31I	DD
TR31 Import – Permit E5 to DKYGENKY:DKYL0+DEXP	CSNBT31I / CSNET31I	DD
TR31 Import – Permit V0/V1/V2:N to PINGEN/PINVER	CSNBT31I / CSNET31I	DD
Transaction Validation – Generate	CSNBTRV / CSNETRV	ED
Transaction Validation - Verify CSC-3	CSNBTRV / CSNETRV	ED
Transaction Validation - Verify CSC-4	CSNBTRV / CSNETRV	ED
Transaction Validation - Verify CSC-5	CSNBTRV / CSNETRV	ED
Trusted Block Create - Activate an Inactive Trusted Key Block	CSNDTBC / CSNFTBC	ED
Trusted Block Create - Create Trusted Key Block in Inactive Form	CSNDTBC / CSNFTBC	ED
Unique Key Derive	CSNBUKD / CSNEUKD	ED
Unique Key Derive - Allow PIN-DATA processing	CSNBUKD / CSNEUKD	DD
Unique Key Derive - Override default wrapping	CSNBUKD / CSNEUKD	ED
VISA CVV Generate	CSNBCSG / CSNECSG	ED
VISA CVV Verify	CSNBCSV / CSNECSV	ED

There are relationships between certain access control points. A controlling access control point is required to be enabled before subordinate access control points can be enabled. The TKE workstation will enable the controlling access control point when a subordinate access control point is enabled.

- To use Data Key Export - Unrestricted, the Data Key Export access control point must be enabled.
- To use Data Key Import - Unrestricted, the Data Key Import access control point must be enabled.
- Diversified Key Generate - single length or same halves requires either Diversified Key Generate - TDES-ENC or Diversified Key Generate - TDES-DEC be enabled.
- To use Key Export - Unrestricted, the Key Export access control point must be enabled.
- To use Key Import - Unrestricted, the Key Import access control point must be enabled.
- To use Key Part Import – Unrestricted, the Key Part Import - First key part and Key Part Import - Middle and final access control points must be enabled.

- To use TR31 Export - Permit PINGEN/PINVER to V0/V1/V2:N, the TR31 Export - Permit version A TR-31 key blocks access control point must be enabled.
- To use Unique Key Derive - Allow PIN-DATA processing or Unique Key Derive - Override default wrapping access control points, Unique Key Derive access control point must be enabled.
- To use SET Block Decompose - PIN ext IPINENC or PIN ex OPINENC, the SET Block Decompose access control point must be enabled.
- To use PKA Key Generate - Permit Regeneration Data, the PKA Key Generate access control point must be enabled.
- To use PKA Key Generate - Permit Regeneration Data Retain, the PKA Key Generate and PKA Key Generate – Clone access control points must be enabled.
- To use PKA Key Generate - Clear or PKA Key Generate - Clone, the PKA Key Generate access control point must be enabled.
- To use any of the following access control points, the ECC Diffie-Hellman access control point must be enabled:
  - ECC Diffie-Hellman - Allow PASSTHRU
  - ECC Diffie-Hellman - Allow key wrap override
  - ECC Diffie-Hellman - Allow Prime Curve 192
  - ECC Diffie-Hellman - Allow Prime Curve 224
  - ECC Diffie-Hellman - Allow Prime Curve 256
  - ECC Diffie-Hellman - Allow Prime Curve 384
  - ECC Diffie-Hellman - Allow Prime Curve 521
  - ECC Diffie-Hellman - Allow BP Curve 160
  - ECC Diffie-Hellman - Allow BP Curve 192
  - ECC Diffie-Hellman - Allow BP Curve 224
  - ECC Diffie-Hellman - Allow BP Curve 256
  - ECC Diffie-Hellman - Allow BP Curve 320
  - ECC Diffie-Hellman - Allow BP Curve 384
  - ECC Diffie-Hellman - Allow BP Curve 512
  - ECC Diffie-Hellman - Prohibit weak key generate

---

## Appendix I. Accessibility

Publications for this product are offered in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when using PDF files, you may view the information through the z/OS® Internet Library web site or the z/OS Information Center. If you continue to experience problems, send an email to [mhvrdfs@us.ibm.com](mailto:mhvrdfs@us.ibm.com) or write to:

IBM® Corporation  
Attention: MHVRCFS Reader Comments  
Department H6MA, Building 707  
2455 South Road  
Poughkeepsie, NY 12601-5400  
USA

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size.

---

### Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

---

### Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

---

### z/OS information

z/OS information is accessible using screen readers with the Library Server versions of z/OS books in the Internet library at:

<http://www.ibm.com/systems/z/os/zos/bkserv/>



---

## Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan, Ltd.  
1623-14, Shimotsuruma, Yamato-shi  
Kanagawa 242-8502 Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel  
IBM Corporation  
2455 South Road  
Poughkeepsie, NY 12601-5400  
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

---

## Programming Interface Information

This book documents intended Programming Interfaces that allow the customer to write programs to obtain the services of z/OS Integrated Cryptographic Service Facility.

---

## Trademarks

IBM, the IBM logo, and [ibm.com](http://ibm.com)<sup>®</sup> are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.





---

## Glossary

This glossary defines terms and abbreviations used in Integrated Cryptographic Service Facility (ICSF). If you do not find the term you are looking for, refer to the index of the appropriate Integrated Cryptographic Service Facility document or view *IBM Glossary of Computing Terms* located at:

<http://www.ibm.com/ibm/terminology>

This glossary includes terms and definitions from:

- *IBM Glossary of Computing Terms*. Definitions are identified by the symbol (D) after the definition.
- *The American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies can be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- *The Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

Definitions specific to the Integrated Cryptographic Services Facility are labeled "In ICSF."

### A

**access method services (AMS).** The facility used to define and reproduce VSAM key-sequenced data sets (KSDS). (D)

**Advanced Encryption Standard (AES).** In computer security, the National Institute of Standards and Technology (NIST) Advanced Encryption Standard

(AES) algorithm. The AES algorithm is documented in a draft Federal Information Processing Standard.

**AES.** Advanced Encryption Standard.

**American National Standard Code for Information Interchange (ASCII).** The standard code using a coded character set consisting of 7-bit characters (8 bits including parity check) that is used for information exchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

**ANSI key-encrypting key (AKEK).** A 64- or 128-bit key used exclusively in ANSI X9.17 key management applications to protect data keys exchanged between systems.

**ANSI X9.17.** An ANSI standard that specifies algorithms and messages for DES key distribution.

**ANSI X9.19.** An ANSI standard that specifies an optional double-MAC procedure which requires a double-length MAC key.

**application program.** (1) A program written for or by a user that applies to the user's work, such as a program that does inventory control or payroll. (2) A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities. (D)

**application program interface (API).** (1) A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program. (D) (2) In ICSF, a callable service.

**asymmetric cryptography.** Synonym for public key cryptography. (D)

**authentication pattern.** An 8-byte pattern that ICSF calculates from the master key when initializing the cryptographic key data set. ICSF places the value of the authentication pattern in the header record of the cryptographic key data set.

**authorized program facility (APF).** A facility that permits identification of programs authorized to use restricted functions. (D)

## C

**callable service.** A predefined sequence of instructions invoked from an application program, using a CALL instruction. In ICSF, callable services perform cryptographic functions and utilities.

**CBC.** Cipher block chaining.

**CCA.** Common Cryptographic Architecture.

**CCF.** Cryptographic Coprocessor Feature.

**CDMF.** Commercial Data Masking Facility.

**CEDA.** A CICS transaction that defines resources online. Using CEDA, you can update both the CICS system definition data set (CSD) and the running CICS system.

**CEX2A.** Crypto Express2 Accelerator

**CEX2C.** Crypto Express2 Coprocessor

| **CEX3A.** Crypto Express3 Accelerator

| **CEX3C.** Crypto Express3 Coprocessor

**checksum.** (1) The sum of a group of data associated with the group and used for checking purposes. (T) (2) In ICSF, the data used is a key part. The resulting checksum is a two-digit value you enter when you use the key-entry unit to enter a master key part or a clear key part into the key-storage unit.

**Chinese Remainder Theorem (CRT).** A mathematical theorem that defines a format for the RSA private key that improves performance.

**CICS.** Customer Information Control System.

**cipher block chaining (CBC).** A mode of encryption that uses the data encryption algorithm and requires an initial chaining vector. For encipher, it exclusively ORs the initial block of data with the initial control vector and then enciphers it. This process results in the encryption both of the input block and of the initial control vector that it uses on the next input block as the process repeats. A comparable chaining process works for decipher.

**ciphertext.** (1) In computer security, text produced by encryption. (2) Synonym for enciphered data. (D)

**CKDS.** Cryptographic Key Data Set.

**clear key.** Any type of encryption key not protected by encryption under another key.

**CMOS.** Complementary metal oxide semiconductor.

**coexistence mode.** An ICSF method of operation during which CUSP or PCF can run independently and simultaneously on the same ICSF system. A CUSP or

PCF application program can run on ICSF in this mode if the application program has been reassembled.

**Commercial Data Masking Facility (CDMF).** A data-masking algorithm using a DES-based kernel and a key that is shortened to an effective key length of 40 DES key-bits. Because CDMF is not as strong as DES, it is called a masking algorithm rather than an encryption algorithm. Implementations of CDMF, when used for data confidentiality, are generally exportable from the USA and Canada.

**Common Cryptographic Architecture: Cryptographic Application Programming Interface.** Defines a set of cryptographic functions, external interfaces, and a set of key management rules that provide a consistent, end-to-end cryptographic architecture across different IBM platforms.

**compatibility mode.** An ICSF method of operation during which a CUSP or PCF application program can run on ICSF without recompiling it. In this mode, ICSF cannot run simultaneously with CUSP or PCF.

**complementary keys.** A pair of keys that have the same clear key value, are different but complementary types, and usually exist on different systems.

**console.** A part of a computer used for communication between the operator or maintenance engineer and the computer. (A)

**control-area split.** In systems with VSAM, the movement of the contents of some of the control intervals in a control area to a newly created control area in order to facilitate insertion or lengthening of a data record when there are no remaining free control intervals in the original control area. (D)

**control block.** (1) A storage area used by a computer program to hold control information. (I) Synonymous with control area. (2) The circuitry that performs the control functions such as decoding microinstructions and generating the internal control signals that perform the operations requested. (A)

**control interval.** A fixed-length area of direct-access storage in which VSAM stores records and creates distributed free space. Also, in a key-sequenced data set or file, the set of records pointed to by an entry in the sequence-set index record. The control interval is the unit of information that VSAM transmits to or from direct access storage. A control interval always comprises an integral number of physical records. (D)

**control interval split.** In systems with VSAM, the movement of some of the stored records in a control interval to a free control interval to facilitate insertion or lengthening of a record that does not fit in the original control interval. (D)

**control statement input data set.** A key generator utility program data set containing control statements that a particular key generator utility program job will process.

**control statement output data set.** A key generator utility program data set containing control statements to create the complements of keys created by the key generator utility program.

**control vector.** In ICSF, a mask that is exclusive ORed with a master key or a transport key before ICSF uses that key to encrypt another key. Control vectors ensure that keys used on the system and keys distributed to other systems are used for only the cryptographic functions for which they were intended.

**CPACE.** CP Assist for Cryptographic Functions

**CP Assist for Cryptographic Functions.** Implemented on all z890, z990, z9 EC, z9 BC, z10 EC and z10 BC processors to provide SHA-1 secure hashing.

**cross memory mode.** Synchronous communication between programs in different address spaces that permits a program residing in one address space to access the same or other address spaces. This synchronous transfer of control is accomplished by a calling linkage and a return linkage.

**CRT.** Chinese Remainder Theorem.

**Crypto Express2 Coprocessor.** An asynchronous cryptographic coprocessor available on the z890, z990, z9 EC, z9 BC, z10 EC and z10 BC.

| **Crypto Express3 Coprocessor.** An asynchronous  
| cryptographic coprocessor available on z10 EC and z10  
| BC.

**cryptographic adapter (4755 or 4758).** An expansion board that provides a comprehensive set of cryptographic functions for the network security processor and the workstation in the TSS family of products.

**cryptographic coprocessor.** A microprocessor that adds cryptographic processing functions to specific z890, z990, z9 EC, z9 BC, z10 EC and z10 BC processors. The Cryptographic Coprocessor Feature is a tamper-resistant chip built into the processor board.

**cryptographic key data set (CKDS).** (1) A data set that contains the encrypting keys used by an installation. (D) (2) In ICSF, a VSAM data set that contains all the cryptographic keys. Besides the encrypted key value, an entry in the cryptographic key data set contains information about the key.

**cryptography.** (1) The transformation of data to conceal its meaning. (2) In computer security, the principles, means, and methods for encrypting plaintext and decrypting ciphertext. (D) (3) In ICSF, the use of

cryptography is extended to include the generation and verification of MACs, the generation of MDCs and other one-way hashes, the generation and verification of PINs, and the generation and verification of digital signatures.

**CUSP (Cryptographic Unit Support Program).** The IBM cryptographic offering, program product 5740-XY6, using the channel-attached 3848. CUSP is no longer in service.

**CUSP/PCF conversion program.** A program, for use during migration from CUSP or PCF to ICSF, that converts a CUSP or PCF cryptographic key data set into a ICSF cryptographic key data set.

**Customer Information Control System (CICS).** An IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user written application programs. It includes facilities for building, using, and maintaining databases.

**CVC.** Card verification code used by MasterCard.

**CVV.** Card verification value used by VISA.

## D

**data encryption algorithm (DEA).** In computer security, a 64-bit block cipher that uses a 64-bit key, of which 56 bits are used to control the cryptographic process and 8 bits are used for parity checking to ensure that the key is transmitted properly. (D)

**data encryption standard (DES).** In computer security, the National Institute of Standards and Technology (NIST) Data Encryption Standard, adopted by the U.S. government as Federal Information Processing Standard (FIPS) Publication 46, which allows only hardware implementations of the data encryption algorithm. (D)

**data key or data-encrypting key.** (1) A key used to encipher, decipher, or authenticate data. (D) (2) In ICSF, a 64-bit encryption key used to protect data privacy using the DES algorithm or the CDMF algorithm. AES data keys are now supported by ICSF.

**data set.** The major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access. (D)

**data-translation key.** A 64-bit key that protects data transmitted through intermediate systems when the originator and receiver do not share the same key.

**DEA.** Data encryption algorithm.

**decipher.** (1) To convert enciphered data in order to restore the original data. (T) (2) In computer security, to convert ciphertext into plaintext by means of a cipher

system. (3) To convert enciphered data into clear data. Contrast with encipher. Synonymous with decrypt. (D)

**decode.** (1) To convert data by reversing the effect of some previous encoding. (I) (A) (2) In ICSF, to decipher data by use of a clear key.

**decrypt.** See decipher.

**DES.** Data Encryption Standard.

**diagnostics data set.** A key generator utility program data set containing a copy of each input control statement followed by a diagnostic message generated for each control statement.

**digital signature.** In public key cryptography, information created by using a private key and verified by using a public key. A digital signature provides data integrity and source nonrepudiation.

**Digital Signature Algorithm (DSA).** A public key algorithm for digital signature generation and verification used with the Digital Signature Standard.

**Digital Signature Standard (DSS).** A standard describing the use of algorithms for digital signature purposes. One of the algorithms specified is DSA (Digital Signature Algorithm).

**domain.** (1) That part of a network in which the data processing resources are under common control. (T) (2) In ICSF, an index into a set of master key registers.

**double-length key.** A key that is 128 bits long. A key can be either double- or single-length. A single-length key is 64 bits long.

**DSA.** Digital Signature Algorithm.

**DSS.** Digital Signature Standard.

## E

**ECB.** Electronic codebook.

**ECI.** Eurochèque International S.C., a financial institution consortium that has defined three PIN block formats.

**EID.** Environment Identification.

**electronic codebook (ECB) operation.** (1) A mode of operation used with block cipher cryptographic algorithms in which plaintext or ciphertext is placed in the input to the algorithm and the result is contained in the output of the algorithm. (D) (2) A mode of encryption using the data encryption algorithm, in which each block of data is enciphered or deciphered without an initial chaining vector. It is used for key management functions and the encode and decode callable services.

**electronic funds transfer system (EFTS).** A computerized payment and withdrawal system used to transfer funds from one account to another and to obtain related financial data. (D)

**encipher.** (1) To scramble data or to convert data to a secret code that masks the meaning of the data to any unauthorized recipient. Synonymous with encrypt. (2) Contrast with decipher. (D)

**enciphered data.** Data whose meaning is concealed from unauthorized users or observers. (D)

**encode.** (1) To convert data by the use of a code in such a manner that reconversion to the original form is possible. (T) (2) In computer security, to convert plaintext into an unintelligible form by means of a code system. (D) (3) In ICSF, to encipher data by use of a clear key.

**encrypt.** See encipher.

**exit.** (1) To execute an instruction within a portion of a computer program in order to terminate the execution of that portion. Such portions of computer programs include loops, subroutines, modules, and so on. (T) (2) In ICSF, a user-written routine that receives control from the system during a certain point in processing—for example, after an operator issues the START command.

**exportable form.** A condition a key is in when enciphered under an exporter key-encrypting key. In this form, a key can be sent outside the system to another system. A key in exportable form cannot be used in a cryptographic function.

**exporter key-encrypting key.** A 128-bit key used to protect keys sent to another system. A type of transport key.

## F

**file.** A named set of records stored or processed as a unit. (T)

## G

**GBP.** German Bank Pool.

**German Bank Pool (GBP).** A German financial institution consortium that defines specific methods of PIN calculation.

## H

**hashing.** An operation that uses a one-way (irreversible) function on data, usually to reduce the length of the data and to provide a verifiable authentication value (checksum) for the hashed data.



**header record.** A record containing common, constant, or identifying information for a group of records that follows. (D)

## I

**ICSF.** Integrated Cryptographic Service Facility.

**importable form.** A condition a key is in when it is enciphered under an importer key-encrypting key. A key is received from another system in this form. A key in importable form cannot be used in a cryptographic function.

**importer key-encrypting key.** A 128-bit key used to protect keys received from another system. A type of transport key.

**initial chaining vector (ICV).** A 64-bit random or pseudo-random value used in the cipher block chaining mode of encryption with the data encryption algorithm.

**initial program load (IPL).** (1) The initialization procedure that causes an operating system to commence operation. (2) The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction. (3) The process of loading system programs and preparing a system to run jobs. (D)

**input PIN-encrypting key.** A 128-bit key used to protect a PIN block sent to another system or to translate a PIN block from one format to another.

**installation exit.** See exit.

**Integrated Cryptographic Service Facility (ICSF).** A licensed program that runs under MVS/System Product 3.1.3, or higher, or OS/390 Release 1, or higher, or z/OS, and provides access to the hardware cryptographic feature for programming applications. The combination of the hardware cryptographic feature and ICSF provides secure high-speed cryptographic services.

**International Organization for Standardization.** An organization of national standards bodies from many countries, established to promote the development of standards to facilitate the international exchange of goods and services and to develop cooperation in intellectual, scientific, technological, and economic activity. ISO has defined certain standards relating to cryptography and has defined two PIN block formats.

**ISO.** International Organization for Standardization.

## J

**job control language (JCL).** A control language used to identify a job to an operating system and to describe the job's requirements. (D)

## K

**key-encrypting key (KEK).** (1) In computer security, a key used for encryption and decryption of other keys. (D) (2) In ICSF, a master key or transport key.

**key generator utility program (KGUP).** A program that processes control statements for generating and maintaining keys in the cryptographic key data set.

**key output data set.** A key generator utility program data set containing information about each key that the key generator utility program generates except an importer key for file encryption.

**key part.** A 32-digit hexadecimal value that you enter for ICSF to combine with other values to create a master key or clear key.

**key part register.** A register in the key storage unit that stores a key part while you enter the key part.

**key store policy.** Ensures that only authorized users and jobs can access secure key tokens that are stored in one of the ICSF key stores - the CKDS or the PKDS.

**key store policy controls.** Resources that are defined in the XFACILIT class. A control can verify the caller has authority to use a secure token and identify the action to take when the secure token is not stored in the CKDS or PKDS.

## L

**linkage.** The coding that passes control and parameters between two routines.

**load module.** All or part of a computer program in a form suitable for loading into main storage for execution. A load module is usually the output of a linkage editor. (T)

**LPAR mode.** The central processor mode that enables the operator to allocate the hardware resources among several logical partitions.

## M

**MAC generation key.** A 64-bit or 128-bit key used by a message originator to generate a message authentication code sent with the message to the message receiver.

**MAC verification key.** A 64-bit or 128-bit key used by a message receiver to verify a message authentication code received with a message.

**magnetic tape.** A tape with a magnetizable layer on which data can be stored. (T)

**master key.** (1) In computer security, the top-level key in a hierarchy of key-encrypting keys. (2) ICSF uses

master keys to encrypt operational keys. Master keys are known only to the cryptographic coprocessors and are maintained in tamper proof cryptographic coprocessors. Examples of cryptographic coprocessors are CCF, PCICC, PCIXCC, CEX2C, and CEX3C. Some of the master keys that ICSF supports are a 128-bit DES master key, a 192-bit signature master key, and the 192-bit key management master key, a 192-bit symmetric master key (that is, DES), a 192-bit asymmetric master key, and a 256-bit AES master key.

**master key concept.** The idea of using a single cryptographic key, the master key, to encrypt all other keys on the system.

**master key register.** A register in the cryptographic coprocessors that stores the master key that is active on the system.

**master key variant.** A key derived from the master key by use of a control vector. It is used to force separation by type of keys on the system.

**MD4.** Message Digest 4. A hash algorithm.

**MD5.** Message Digest 5. A hash algorithm.

**message authentication code (MAC).** (1) The cryptographic result of block cipher operations on text or data using the cipher block chain (CBC) mode of operation. (D) (2) In ICSF, a MAC is used to authenticate the source of the message, and verify that the message was not altered during transmission or storage.

**modification detection code (MDC).** (1) A 128-bit value that interrelates all bits of a data stream so that the modification of any bit in the data stream results in a new MDC. (2) In ICSF, an MDC is used to verify that a message or stored data has not been altered.

**multiple encipherment.** The method of encrypting a key under a double-length key-encrypting key.

## N

**new master key register.** A register in the key storage unit that stores a master key before you make it active on the system.

**NIST.** U.S. National Institute of Science and Technology.

**NOCV processing.** Process by which the key generator utility program or an application program encrypts a key under a transport key itself rather than a transport key variant.

**noncompatibility mode.** An ICSF method of operation during which CUSP or PCF can run independently and simultaneously on the same z/OS, OS/390 or MVS system. You cannot run a CUSP or PCF application program on ICSF in this mode.

**nonrepudiation.** A method of ensuring that a message was sent by the appropriate individual.

**notarization.** The ANSI X9.17 process involving the coupling of an ANSI key-encrypting key (AKEK) with ASCII character strings containing origin and destination identifiers and then exclusive ORing (or offsetting) the result with a binary counter.

## O

**OAEP.** Optimal asymmetric encryption padding.

**offset.** The process of exclusively ORing a counter to a key.

**old master key register.** A register in the key storage unit that stores a master key that you replaced with a new master key.

**operational form.** The condition of a key when it is encrypted under the master key so that it is active on the system.

**output PIN-encrypting key.** A 128-bit key used to protect a PIN block received from another system or to translate a PIN block from one format to another.

## P

**PAN.** Personal Account Number.

**parameter.** Data passed between programs or procedures. (D)

**parmlib.** A system parameter library, either SYS1.PARMLIB or an installation-supplied library.

**partial notarization.** The ANSI X9.17 standard does not use the term partial notarization. IBM has divided the notarization process into two steps and defined the term partial notarization as a process during which only the first step of the two-step ANSI X9.17 notarization process is performed. This step involves the coupling of an ANSI key-encrypting key (AKEK) with ASCII character strings containing origin and destination identifiers.

**partitioned data set (PDS).** A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data. (D)

**PCI Cryptographic Coprocessor.** The 4758 model 2 standard PCI-bus card supported on the field upgraded IBM S/390 Parallel Enterprise Server - Generation 5, the IBM S/390 Parallel Enterprise Server - Generation 6 and the IBM @server zSeries.

**PCICA.** PCI Cryptographic Accelerator.

**PCICC.** PCI Cryptographic Coprocessor.

**PCI X Cryptographic Coprocessor.** An asynchronous cryptographic coprocessor available on the IBM @server zSeries 990 and IBM @server zSeries 800.

**PCIXCC.** PCI X Cryptographic Coprocessor.

**Personal Account Number (PAN).** A Personal Account Number identifies an individual and relates that individual to an account at a financial institution. It consists of an issuer identification number, customer account number, and one check digit.

**personal identification number (PIN).** The 4- to 12-digit number entered at an automatic teller machine to identify and validate the requester of an automatic teller machine service. Personal identification numbers are always enciphered at the device where they are entered, and are manipulated in a secure fashion.

**Personal Security card.** An ISO-standard "smart card" with a microprocessor that enables it to perform a variety of functions such as identifying and verifying users, and determining which functions each user can perform.

**PIN block.** A 64-bit block of data in a certain PIN block format. A PIN block contains both a PIN and other data.

**PIN generation key.** A 128-bit key used to generate PINs or PIN offsets algorithmically.

**PIN key.** A 128-bit key used in cryptographic functions to generate, transform, and verify the personal identification numbers.

**PIN offset.** For 3624, the difference between a customer-selected PIN and an institution-assigned PIN. For German Bank Pool, the difference between an institution PIN (generated with an institution PIN key) and a pool PIN (generated with a pool PIN key).

**PIN verification key.** A 128-bit key used to verify PINs algorithmically.

**PKA.** Public Key Algorithm.

**PKCS.** Public Key Cryptographic Standards (RSA Data Security, Inc.)

**PKDS.** Public key data set (PKA cryptographic key data set).

**plaintext.** Data in normal, readable form.

**primary space allocation.** An area of direct access storage space initially allocated to a particular data set or file when the data set or file is defined. See also secondary space allocation. (D)

**private key.** In computer security, a key that is known only to the owner and used with a public key algorithm to decrypt data or generate digital signatures.

The data is encrypted and the digital signature is verified using the related public key.

**processor complex.** A configuration that consists of all the machines required for operation.

**Processor Resource/Systems Manager.** Enables logical partitioning of the processor complex, may provide additional byte-multiplexer channel capability, and supports the VM/XA System Product enhancement for Multiple Preferred Guests.

**Programmed Cryptographic Facility (PCF).** (1) An IBM licensed program that provides facilities for enciphering and deciphering data and for creating, maintaining, and managing cryptographic keys. (D) (2) The IBM cryptographic offering, program product 5740-XY5, using software only for encryption and decryption. This product is no longer in service; ICSF is the replacement product.

**PR/SM.** Processor Resource/Systems Manager.

**public key.** In computer security, a key made available to anyone who wants to encrypt information using the public key algorithm or verify a digital signature generated with the related private key. The encrypted data can be decrypted only by use of the related private key.

**public key algorithm (PKA).** In computer security, an asymmetric cryptographic process in which a public key is used for encryption and digital signature verification and a private key is used for decryption and digital signature generation.

**public key cryptography.** In computer security, cryptography in which a public key is used for encryption and a private key is used for decryption. Synonymous with asymmetric cryptography.

## R

**RACE Integrity Primitives Evaluation Message Digest.** A hash algorithm.

**RDO.** Resource definition online.

**record chaining.** When there are multiple cipher requests and the output chaining vector (OCV) from the previous encipher request is used as the input chaining vector (ICV) for the next encipher request.

**Resource Access Control Facility (RACF).** An IBM licensed program that provides for access control by identifying and verifying the users to the system, authorizing access to protected resources, logging the detected unauthorized attempts to enter the system, and logging the detected accesses to protected resources. (D)



**retained key.** A private key that is generated and retained within the secure boundary of the PCI Cryptographic Coprocessor.

**return code.** (1) A code used to influence the execution of succeeding instructions. (A) (2) A value returned to a program to indicate the results of an operation requested by that program. (D)

**Rivest-Shamir-Adleman (RSA) algorithm.** A process for public key cryptography that was developed by R. Rivest, A. Shamir, and L. Adleman.

**RMF.** Resource Manager Interface.

**RMI.** Resource Measurement Facility.

**RSA.** Rivest-Shamir-Adleman.

## S

**SAF.** Security Authorization Facility.

**save area.** Area of main storage in which contents of registers are saved. (A)

**secondary space allocation.** In systems with VSAM, area of direct access storage space allocated after primary space originally allocated is exhausted. See also primary space allocation. (D)

**Secure Electronic Transaction.** A standard created by Visa International and MasterCard for safe-guarding payment card purchases made over open networks.

**secure key.** A key that is encrypted under a master key. When ICSF uses a secure key, it is passed to a cryptographic coprocessor where the coprocessor decrypts the key and performs the function. The secure key never appears in the clear outside of the cryptographic coprocessor.

**Secure Sockets Layer.** A security protocol that provides communications privacy over the Internet by allowing client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

**sequential data set.** A data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape. (D)

**SET.** Secure Electronic Transaction.

**SHA (Secure Hash Algorithm, FIPS 180) .** (Secure Hash Algorithm, FIPS 180) The SHA (Secure Hash Algorithm) family is a set of related cryptographic hash functions designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST). The first member of the family, published in 1993, is officially called SHA. However, today, it is often unofficially called SHA-0 to avoid confusion with its successors. Two years later,

SHA-1, the first successor to SHA, was published. Four more variants, have since been published with increased output ranges and a slightly different design: SHA-224, SHA-256, SHA-384, and SHA-512 (all are sometimes referred to as SHA-2).

**SHA-1 (Secure Hash Algorithm 1, FIPS 180).** A hash algorithm required for use with the Digital Signature Standard.

**SHA-2 (Secure Hash Algorithm 2, FIPS 180).** Four additional variants to the SHA family, with increased output ranges and a slightly different design: SHA-224, SHA-256, SHA-384, and SHA-512 (all are sometimes referred to as SHA-2).

**SHA-224.** One of the SHA-2 algorithms.

**SHA-256 .** One of the SHA-2 algorithms.

**SHA-384.** One of the SHA-2 algorithms.

**SHA-512 .** One of the SHA-2 algorithms.

**single-length key.** A key that is 64 bits long. A key can be single- or double-length. A double-length key is 128 bits long.

**smart card.** A plastic card that has a microchip capable of storing data or process information.

**special secure mode.** An alternative form of security that allows you to enter clear keys with the key generator utility program or generate clear PINs.

**SSL.** Secure Sockets Layer.

**supervisor state.** A state during which a processing unit can execute input/output and other privileged instructions. (D)

**System Authorization Facility (SAF).** An interface to a system security system like the Resource Access Control Facility (RACF).

**system key.** A key that ICSF creates and uses for internal processing.

**System Management Facility (SMF).** A base component of z/OS that provides the means for gathering and recording information that can be used to evaluate system usage. (D)

## T

**TDEA.** Triple Data Encryption Algorithm.

**TKE.** Trusted key entry.

**Transaction Security System.** An IBM product offering including both hardware and supporting software that provides access control and basic cryptographic key-management functions in a network

environment. In the workstation environment, this includes the 4755 Cryptographic Adapter, the Personal Security Card, the 4754 Security Interface Unit, the Signature Verification feature, the Workstation Security Services Program, and the AIX Security Services Program/6000. In the host environment, this includes the 4753 Network Security Processor and the 4753 Network Security Processor MVS Support Program.

**transport key.** A 128-bit key used to protect keys distributed from one system to another. A transport key can either be an exporter key-encrypting key, an importer key-encrypting key, or an ANSI key-encrypting key.

**transport key variant.** A key derived from a transport key by use of a control vector. It is used to force separation by type for keys sent between systems.

**TRUE.** Task-related User Exit (CICS). The CICS-ICSF Attachment Facility provides a CSFATRUE and CSFATREN routine.

## U

**UAT.** UDX Authority Table.

**UDF.** User-defined function.

**UDK.** User-derived key.

**UDP.** User Developed Program.

**UDX.** User Defined Extension.

## V

**verification pattern.** An 8-byte pattern that ICSF calculates from the key parts you enter when you enter a master key or clear key. You can use the verification pattern to verify that you have entered the key parts correctly and specified a certain type of key.

**Virtual Storage Access Method (VSAM).** An access method for indexed or sequential processing of fixed and variable-length records on direct-access devices. The records in a VSAM data set or file can be organized in logical sequence by means of a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by means of relative-record number.

**Virtual Telecommunications Access Method (VTAM).** An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability. (D)

**VISA.** A financial institution consortium that has defined four PIN block formats and a method for PIN verification.

**VISA PIN Verification Value (VISA PVV).** An input to the VISA PIN verification process that, in practice, works similarly to a PIN offset.

## Numerics

**3621.** A model of an IBM Automatic Teller Machine that has a defined PIN block format.

**3624.** A model of an IBM Automatic Teller Machine that has a defined PIN block format and methods of PIN calculation.

**4753.** The Network Security processor. The IBM 4753 is a processor that uses the Data Encryption Algorithm and the RSA algorithm to provide cryptographic support for systems requiring secure transaction processing (and other cryptographic services) at the host computer. The NSP includes a 4755 cryptographic adapter in a workstation which is channel attached to a S/390 host computer.

**4758.** The IBM PCI Cryptographic processor provides a secure programming and hardware environment where DES and RSA processes are performed.



---

# Index

## Numerics

3621 PIN block format 471, 930  
3624 PIN block format 471, 930  
4700-PAD processing rule 374, 384  
4704-EPP PIN block format 471

## A

accessibility 977  
    contact IBM 977  
    features 977  
    screen readers 979  
accessing  
    callable service 9  
    invocation requirements 9  
AES algorithm 15  
affinity (IEAAFFN callable service) 10  
AKEK key type 25  
ALET (alternate entry point)  
    format 5  
algorithm  
    3624 PIN generation 932  
    3624 PIN verification 935  
    AES 15  
    CDMF 15  
    DES 15  
    GBP PIN generation 933  
    GBP PIN verification 937  
    GBP-PIN 511  
    GBP-PINO 511  
    IBM-PIN 511  
    IBM-PINO 511  
    PIN offset generation 934  
    PIN, detailed 932  
    PIN, general 60  
    PVV generation 938  
    PVV verification 939  
    VISA PIN 938  
    VISA-PVV 488, 511  
    VISAPVV4 511  
ANSI 9.9-1 algorithm 423  
ANSI key-encrypting key (AKEK) 25  
ANSI X3.106 processing rule 940  
ANSI X9.17 EDC generate callable service (CSNAEGN)  
    format 685  
    overview 54  
    parameters 685  
    syntax 685  
ANSI X9.17 key export callable service (CSNAKEX)  
    format 687  
    overview 54  
    parameters 687  
    syntax 687  
ANSI X9.17 key import callable service (CSNAKIM)  
    format 692  
    overview 54  
    parameters 692

ANSI X9.17 key import callable service (CSNAKIM) *(continued)*  
    syntax 692  
ANSI X9.17 key management 685  
    overview 53  
ANSI X9.17 key translate callable service (CSNAKTR)  
    format 697  
    overview 55  
    parameters 697  
    syntax 697  
ANSI X9.17 key-encrypting key 22  
ANSI X9.17 transport key partial notarize callable service (CSNATKN)  
    overview 55  
ANSI X9.17 transport key partial notarize (CSNATKN)  
    format 702  
    parameters 702  
    syntax 702  
ANSI X9.19 optional double MAC  
    procedure 423  
ANSI X9.23 processing rule 374, 384, 941  
ANSI X9.8 507  
ANSI X9.8 PIN block format 929  
ASCII to EBCDIC conversion  
    table 959  
assistive technologies 977  
asym\_encrypted\_key parameter  
    remote key export callable service 252  
asym\_encrypted\_key\_length parameter  
    remote key export callable service 252  
authenticating messages 423

## C

c-variable encrypting key identifier parameter  
    cryptographic variable encipher callable service 115  
call  
    successful 11  
    unsuccessful 12  
callable service  
    ANSI X9.17 EDC generate (CSNAEGN) 54, 685  
    ANSI X9.17 key export (CSNAKEX) 54, 687  
    ANSI X9.17 key import (CSNAKIM) 54, 692  
    ANSI X9.17 key translate (CSNAKTR) 55, 697  
    ANSI X9.17 transport key partial notarize (CSNATKN) 55  
    ANSI X9.17 transport key partial notarize (CSNATKN) 702  
    character/nibble conversion (CSNBXBC and CSNBXCB) 643  
    callable service *(continued)*  
        ciphertext 70  
        ciphertext translate (CSNBCTT or CSNBCTT1) 354  
        Ciphertext Translate2 (CSNBCTT2, CSNBCTT3, CSNECTT2, CSNECTT3) 357  
        CKDS key record create (CSNBKRC) 50, 613  
        CKDS key record create2 (CSNBKRC2 and CSNEKRC2) 50  
        CKDS Key Record Create2 (CSNBKRC2 and CSNEKRC2) 615  
        CKDS key record delete (CSNBKRD) 50, 617  
        CKDS key record read (CSNBKRR) 50, 619  
        CKDS key record read2 (CSNBKRR2 and CSNEKRR2) 51  
        CKDS Key Record Read2 (CSNBKRR2 and CSNEKRR2) 621  
        CKDS key record write (CSNBKRW) 51, 623  
        CKDS key record write2 (CSNBKRW2 and CSNEKRW2) 51  
        CKDS Key Record Write2 (CSNBKRW2 and CSNEKRW2) 626  
        clear key import (CSNBCKI) 28, 104  
        clear PIN encrypt (CSNBCPE) 60, 476  
        clear PIN generate (CSNBPGN) 61, 480  
        clear PIN generate alternate (CSNBCPA) 61, 485  
        code conversion (CSNBXAE) 65  
        code conversion (CSNBXBC) 65  
        code conversion (CSNBXCB) 65  
        code conversion (CSNBXEA and CSNBXAE) 645  
        code conversion (CSNBXEA) 65  
        coding examples 909  
            Assembler H 914  
            C 909  
            COBOL 912  
            PL/1 916  
        control vector generate (CSNBCVG) 28, 106  
        control vector translate callable service (CSNBCVT) 28, 110  
        coordinated KDS administration (CSFCRC and CSFCRC6) 628  
        coordinated KDS administration callable services (CSFCRC and CSFCRC6) 51  
        cryptographic variable encipher (CSNBCVE) 28, 114  
        CSFxxxx format 4  
        CSNBxxxx format 4  
        CVV Key Combine (CSNBCKC and CSNECKC) 491  
        data key export (CSNBKX) 28, 116

callable service (*continued*)

data key import (CSNBDKM) 28, 119  
decipher (CSNBDEC or CSNBDEC1) 370  
decode (CSNBDCO) 377  
definition 3, 15  
digital signature generate (CSNDDSG) 87, 557  
digital signature verify (CSNDDSV) 88, 566  
diversified key generate (CSNBDBG) 28, 122  
ECC Diffie-Hellman (CSNDEDH and CSNFEDH) 129  
encipher (CSNBENC or CSNBENC1) 379  
encode (CSNBECO) 387  
encrypted PIN generate (CSNBEPG) 61, 497  
encrypted PIN translate (CSNBPTR) 61, 502  
encrypted PIN verification (CSNBPVR) 62  
encrypted PIN verify (CSNBPVR) 509  
format 677, 682  
get attribute value (CSFPGAV) 722  
HMAC Generate (CSNBHMG, CSNEHMG, CSNBHMG1 and CSNEHMG1) 425  
HMAC generation (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1) 57  
HMAC verification (CSNBHMG or CSNBHMG1 and CSNEHMG or CSNEHMG1) 57  
HMAC Verify (CSNBHMG, CSNEHMG, CSNBHMG1 and CSNEHMG1) 429  
ICSF Query Algorithm (CSFIQA) 65, 647  
ICSF Query Service (CSFIQF) 65, 652  
IEAAFFN (affinity) 10  
installation-defined 15  
invoking a 3  
key export (CSNBKEX) 28, 137  
key generate (CSNBKGN) 28, 67, 142  
Key Generate (CSNBKGN) 32  
key generate2 (CSNBKGN2 and CSNEKGN2) 32, 33  
Key Generate2 (CSNBKGN2 and CSNEKGN2) 156  
key import (CSNBKIM) 29, 165  
key part import (CSNBKPI) 29, 171  
key part import2 (CSNBKPI2 and CSNEKPI2) 32, 33, 34  
Key Part Import2 (CSNBKPI2 and CSNEKPI2) 176  
key test (CSNBKYT) 179  
key test (CSNBKYT and CSNBKYTX) 29  
key test extended (CSNBKYTX) 188  
Key Test2 (CSNBKYT2 and CSNEKYT2) 184  
key token build (CSNBKTB) 29, 192  
Key Token Build2 (CSNBKTB2 and CSNEKTB2) 202

callable service (*continued*)

key translate (CSNBKTR) 29, 209  
Key Translate2 (CSNBKTR2 and CSNEKTR2) 211  
link edit step 13  
MAC generate (CSNBMG or CSNBMG1) 434  
MAC generation (CSNBMG or CSNBMG1) 57  
MAC verification (CSNBMR or CSNBMR1) 58  
MAC verify (CSNBMR or CSNBMR1) 440  
MDC generate (CSNBMDG or CSNBMDG1) 445  
MDC generation (CSNBMDG or CSNBMDG1) 59  
multiple clear key import (CSNBCKM) 30, 218  
multiple secure key import (CSNBCKM) 30, 222  
one-way hash generate (CSNBOWH and CSNBOWH1) 59  
one-way hash generate (CSNBOWH, CSNEOWH and CSNBOWH1) 450  
overview 3  
PCI interface (CSFPCI) 677  
PIN change/unblock (CSNBPCU) 62  
PIN Change/Unblock (CSNBPCU) 516  
PKA decrypt (CSNDPKD) 52  
PKA encrypt (CSNDPKE) 52  
PKA key generate (CSNDPKG) 88, 571  
PKA key import (CSNDPKI) 88, 578  
PKA key token build (CSNDPKB) 88, 583  
PKA key token change (CSNDKTC and CSNFKTC) 89  
PKA key token change (CSNDKTC) 596  
PKA keyTranslate (CSNDPKT) 599  
PKA public key extract (CSNDPKX) 89, 604  
PKCS #11 Derive key (CSFPDVK) 717  
PKCS #11 Derive multiple keys (CSFPDMK) 709  
PKCS #11 Generate HMAC (CSFPHMG) 730  
PKCS #11 Generate secret key (CSFPGSK) 728  
PKCS #11 One-way hash, sign, or verify (CSFPOWH) 737  
PKCS #11 Private key sign (CSFPKPS) 742  
PKCS #11 Pseudo-random function (CSFPPRF) 747  
PKCS #11 Public key verify (CSFPPKV) 744  
PKCS #11 Secret key decrypt (CSFPSKD) 752  
PKCS #11 Secret key encrypt (CSFPSKE) 756  
PKCS #11 Unwrap key (CSFPUWK) 772

callable service (*continued*)

PKCS #11 Verify HMAC (CSFPHMV) 734  
PKCS #11 wrap key (CSFPWPK) 775  
PKDS key record create (CSNDKRC) 632  
PKDS key record delete (CSNDKRD) 634  
PKDS key record read (CSNDKRR) 636  
PKDS key record write (CSNDKRW) 638  
PKSC interface (CSFPKSC) 682  
PPKCS #11 Generate key pair (CSFPGKP) 725  
prohibit export (CSNBPEX) 30, 239  
prohibit export extended (CSNBPEXX) 30, 241  
random number generate (CSNBRNG) 30, 243  
random number generate (CSNBRNGL) 243  
remote key export (CSNDRKX) 30, 247  
restrict key attribute (CSNBRKA and CSNERKA) 30, 32, 34  
Restrict Key Attribute (CSNBRKA and CSNERKA) 255  
retained key delete (CSNDRKD) 606  
retained key list (CSNDRKL) 609  
secure key import (CSNBSKI) 30, 260  
Secure Key Import2 (CSNBSKI2 and CSNESKI2) 264  
secure messaging for keys (CSNBKEY) 523  
secure messaging for PINs (CSNBSPN) 526  
security considerations 9  
sequences 66  
set attribute value (CSFPSAV) 749  
SET block compose (CSNDSBC) 91, 531  
SET block decompose (CSNDSBD) 91, 536  
symmetric algorithm decipher (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1) 389  
symmetric key decipher (CSNBSYD and CSNBSYD1) 402  
symmetric key encipher (CSNBSE, CSNBSE1, CSNESAE, and CSNESAE1) 396  
symmetric key encipher (CSNBSE, CSNBSE1, CSNESYE and CSNESYE1) 412  
symmetric key export (CSNDSYX) 31, 269  
symmetric key generate (CSNDSYG) 31, 275  
symmetric key import (CSNDSYI) 31, 283  
Symmetric Key Import2 (CSNDSYI2 and CSNFSYI2) 291  
Symmetric MAC generate (CSNBMSG, CSNBMSG1, CSNESMG, and CSNESMG1) 454



- callable service (*continued*)
  - Symmetric MAC Generate Callable Service (CSNBMSG, CSNBMSG1, CSNESMG and CSNESMG1) 58
  - Symmetric MAC verify (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1) 459
  - Symmetric MAC Verify Callable Service (CSNBSMV, CSNBSMV1, CSNESMV and CSNESMV1) 58
  - syntax 4
  - token record create (CSFPTRC) 762
  - token record delete (CSFPTRD) 766
  - token record list (CSFPTRL) 768
  - TR-31 export (CSNBT31X and CSNET31X) 301
  - TR-31 import (CSNBT31I and CSNET31I) 315
  - TR-31 Optional Data Build (CSNBT31O and CSNET31O) 329
  - TR-31 Optional Data Read (CSNBT31R and CSNET31R) 332
  - TR-31 Parse (CSNBT31P and CSNET31P) 336
  - transaction validation 62
  - Transaction Validation (CSNBTRV) 542
  - transform CDMF key (CSNBTK) 31, 295
  - translating ciphertext 56
  - trusted block create (CSNDTBC) 31, 298
  - Unique Key Derive (CSNBUKD and CSNEUKD) 339
  - User derived key (CSFUDK) 347
  - using key types and key forms 11
  - VISA CVV service generate (CSNBCSG) 546
  - VISA CVV service verify (CSNBCSV) 551
  - with ALETs (alternate entry point) 5
  - X9.9 data editing (CSNB9ED) 65, 673
- CBC processing rule 374, 384
- CDMF
  - overview 52
- CDMF algorithm 15
- CDMF key, transforming
  - algorithm 951
  - callable service 295
- certificate length parameter
  - remote key export callable service 249
- certificate parameter
  - remote key export callable service 249
- certificate\_parms parameter
  - remote key export callable service 250
- certificate\_parms\_length parameter
  - remote key export callable service 250
- chaining vector length parameter
  - one-way hash generate callable service 452
  - Symmetric MAC generate callable service 457
- chaining vector length parameter (*continued*)
  - Symmetric MAC verify callable service 462
- chaining vector parameter
  - decipher callable service 375
  - encipher callable service 385
  - MAC generate callable service 437
  - MAC verify callable service 443
  - MDC generate callable service 448
  - one-way hash generate callable service 452
  - symmetric MAC generate callable service 457
  - Symmetric MAC verify callable service 462
- changing control vectors 904
- character/nibble conversion callable service (CSNBXBC and CSNBXCB)
  - format 643
  - parameters 643
  - syntax 643
- character/nibble conversion callable services (CSNBXBC and CSNBXCB)
  - overview 65
- choosing between
  - CSNBCTT and CSNBCTT1 354
  - CSNBCTT2 and CSNBCTT3 358
  - CSNBDEC and CSNBDEC1 371
  - CSNBENC and CSNBENC1 381
  - CSNBMDG and CSNBMDG1 445
  - CSNBMG and CSNBMG1 434
  - CSNBMR and CSNBMR1 440
  - CSNBSYD and CSNBSYD1 404, 454
  - CSNBSYE and CSNBSYE1 413
  - CSNESAE and CSNESAE1 396
- CIPHER
  - keys 21
- cipher block chaining (CBC) 351
  - mode 352
- cipher feedback (CFB)
  - mode 352
- cipher text id parameter
  - decipher callable service 395
  - encipher callable service 401
- ciphertext
  - cryptographic variable encipher callable service 115
  - deciphering 56, 351
  - encoding 387
  - field 387, 411, 421
  - translating 56, 354, 357
- ciphertext id parameter
  - decipher callable service 376, 410
  - encipher callable service 386, 420
- ciphertext parameter
  - decipher callable service 373
  - decode callable service 378
  - encipher callable service 385
  - encode callable service 389
- ciphertext translate callable service (CSNBCTT or CSNBCTT1)
  - format 355
  - parameters 355
  - syntax 355
  - using 70
- CKDS (cryptographic key data set)
  - record format 841, 846, 847, 850
- CKDS key record create callable service (CSNBKRC)
  - format 613
  - overview 50
  - parameters 613
  - syntax 613
- CKDS key record delete callable service (CSNBKRD)
  - format 617
  - parameters 617
  - syntax 617
- CKDS key record read callable service (CSNBKRR)
  - format 619
  - overview 50
  - parameters 619
  - syntax 619
- CKDS key record write callable service (CSNBKRW)
  - format 623
  - overview 51
  - parameters 623
  - syntax 623
- clear key
  - deciphering data with 377
  - definition 25
  - enciphering 260
  - enciphering data with 387
  - encoding and decoding data with 56
  - protecting 351
- clear key import callable service (CSNBCKI)
  - format 104
  - overview 28
  - parameters 104
  - syntax 104
- clear key length parameter
  - multiple clear key import callable service 219, 224
- clear key parameter
  - clear key import callable service 105
  - decode callable service 378
  - encode callable service 388
  - multiple clear key import callable service 219, 224
  - secure key import callable service 261
- clear PIN encrypt callable service (CSNBCPE)
  - format 477
  - syntax 477
- clear PIN encrypt service (CSNBCPE)
  - parameters 477
- clear PIN generate alternate callable service (CSNBCPA)
  - format 485
  - overview 61
  - parameters 485
  - syntax 485
- clear PIN generate callable service (CSNBPGN)
  - format 481
  - parameters 481
  - syntax 481

- clear PIN generate key identifier
  - parameter 487
  - clear PIN generate callable
    - service 481
- clear text id parameter
  - decipher callable service 376, 395, 410
  - encipher callable service 385, 401, 419
- clear text parameter
  - decipher callable service 375
  - decode callable service 378
  - encipher callable service 383
  - encode callable service 388
- code conversion callable service (CSNBXEA and CSNBXAE)
  - format 645
  - parameters 645
  - syntax 645
- code conversion callable services (CSNBXEA and CSNBXAE)
  - overview 65
- code table parameter
  - character/nibble conversion callable
    - service 644
  - code conversion callable service 646
- coding examples 909
  - Assembler H 914
  - C 909
  - COBOL 912
  - PL/1 916
- Commercial Data Masking Facility (CDMF) 351
- control information
  - for digital signature generate 559
  - for digital signature verify 567
  - for diversified key generate 124
  - for key test 181
  - for key test extended 190
  - for MAC generate 436
  - for MAC verify 442, 461
  - for MDC generate 448
  - for multiple clear key import 219
  - for multiple secure key import 223, 303, 317, 333, 493
  - for one-way hash generate 451
  - for PKA key token build 585
  - for symmetric algorithm encipher 3, 392, 398
  - for symmetric key encipher 406, 415
  - for symmetric key generate 276
  - for symmetric key import 284, 292
  - for symmetric MAC generate 457
  - for user derived key 348
  - random number generate callable
    - service 245
- control vector
  - description 893
  - value 893
- control vector generate (CSNBCVG)
  - parameters 106
- control vector generate callable service (CSNBCVG)
  - format 106
  - overview 28
  - syntax 106
- control vector parameter
  - control vector generate callable
    - service 110
- control vector translate callable service (CSNBCVT)
  - format 111
  - overview 28
  - parameters 111, 426, 430, 615, 621, 626
  - syntax 111
- control vector, description of 16, 19
- control vectors, changing 904
- coordinated KDS administration callable
  - service (CSFCRC and CSFCRC6) 628
- coordinated KDS administration callable
  - services (CSFCRC and CSFCRC6)
    - overview 51
- cryptographic feature
  - description xxiii
- cryptographic key data set (CKDS)
  - held keys 22
  - storing keys 27, 49, 103
- cryptographic variable encipher (CSNBCVE)
  - parameters 114
- cryptographic variable encipher callable
  - service (CSNBCVE)
    - format 114
    - overview 28
    - syntax 114
- CSFCRC callable service 628
- CSFCRC6 callable service 628
- CSFIQA callable service 647
- CSFIQF callable service 652
- CSFIQF6 652
- CSFPCI callable service 677
- CSFPDMK callable service 709
- CSFPDVK callable service 717
- CSFPGAV callable service 722
- CSFPGKP callable service 725
- CSFPGSK callable service 728
- CSFPHMG callable service 730
- CSFPHMV callable service 734
- CSFPKSC callable service 682
- CSFPOWH callable service 737
- CSFPPKS callable service 742
- CSFPPKV callable service 744
- CSFPPRF callable service 747
- CSFPSAV callable service 749
- CSFPSKD callable service 752
- CSFPSKE callable service 756
- CSFPTRC callable service 762
- CSFPTRD callable service 766
- CSFPTRL callable service 768
- CSFPUWK callable service 772
- CSFPWPK callable service 775
- CSFUDK callable service 347
- CSFxxxx format 4
- CSNAEGN callable service 685
- CSNAKEX callable service 687, 692
- CSNAKTR callable service 697
- CSNATKN callable service 702
- CSNB9ED callable service 673
- CSNBCKC and CSNECKC callable
  - services 491
- CSNBCKI callable service 104
- CSNBCKM callable service 218
- CSNBCPA callable service 485
- CSNBCPE callable service 476
- CSNBCSG callable service 546
- CSNBCSV callable service 551
- CSNBCTT or CSNBCTT1 callable
  - service 354
- CSNBCTT2, CSNBCTT3, CSNECTT2, or CSNECTT3 callable service 357
- CSNBCVE callable service 114
- CSNBCVG callable service 106
- CSNBCVT callable service 110
- CSNBDCO callable service 377
- CSNBDEC or CSNBDEC1 callable
  - service 370
- CSNBDBG callable service 122
- CSNBDKM callable service 119
- CSNBDKX callable service 116
- CSNBECO callable service 387
- CSNBENC or CSNBENC1 callable
  - service 379
- CSNBEPG callable service 497
- CSNBHMG, CSNEHMG, CSNBHMG1
  - and CSNEHMG1 callable services 425
- CSNBHMGV, CSNEHMGV, CSNBHMGV1 and CSNEHMGV1 callable services 429
- CSNBKEX callable service 137
- CSNBKGN callable service 142
- CSNBKGN2 and CSNEKGN2 callable
  - services 156
- CSNBKIM callable service 165
- CSNBKPI callable service 171
- CSNBKPI2 and CSNEKPI2 callable
  - services 176
- CSNBKRC callable service 613
- CSNBKRC2 and CSNEKRC2 callable
  - services 615
- CSNBKRD callable service 617
- CSNBKRR callable service 619
- CSNBKRR2 and CSNEKRR2 callable
  - services 621
- CSNBKRW callable service 623
- CSNBKRW2 and CSNEKRW2 callable
  - services 626
- CSNBKTB callable service 192
- CSNBKTB2 and CSNEKTB2 callable
  - services 202
- CSNBKTR callable service 209
- CSNBKTR2 and CSNEKTR2 callable
  - services 211
- CSNBKYT callable service 179
- CSNBKYT2 and CSNEKYT2 callable
  - services 184
- CSNBKYTX callable service 188
- CSNBMDG or CSNBMDG1 callable
  - service 445
- CSNBMGN or CSNBMGN1 callable
  - service 434
- CSNBMRV or CSNBMRV1 callable
  - service 440
- CSNBOWH, CSNEOWH and CSNBOWH1 callable services 450
- CSNBPCU callable service 516
- CSNBPEX callable service 239
- CSNBPEXX callable service 241
- CSNBPGN callable service 480
- CSNBPTR callable service 502
- CSNBPVR callable service 509

- CSNBRKA and CSNERKA callable services 255
- CSNBRNG callable service 243
- CSNBRNGL callable service 243
- CSNBSAD or CSNBSAD1 and CSNESAD or CSNESAD1 389
- CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1 callable service 396
- CSNBSKI callable service 260
- CSNBSKI2 and CSNESKI2 callable services 264
- CSNBSKM callable service 222
- CSNBSKY callable service 523
- CSNBMSG, CSNBMSG1, CSNESMG, and CSNESMG1 callable service 454
- CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1 callable service 459
- CSNBSPN callable service 526
- CSNBSYD and CSNBSYD1 callable service 402
- CSNBSYE and CSNBSYE1 callable service 412
- CSNBT31I and CSNET31I callable services 315
- CSNBT31O and CSNET31O callable services 329
- CSNBT31P and CSNET31P callable services 336
- CSNBT31R and CSNET31R callable services 332
- CSNBT31X and CSNET31X callable services 301
- CSNBTCK callable service 295
- CSNBTRV callable service 542
- CSNBUKD and CSNEUKD callable service 339
- CSNBXAE callable service 645
- CSNBXBC callable service 643
- CSNBXCB callable service 643
- CSNBXEA callable service 645
- CSNBxxxx format 4
- CSNDDSG callable service 557
- CSNDDSV callable service 566
- CSNDEDH and CSNFEDH callable services 129
- CSNDKRC callable service 632
- CSNDKRD callable service 634
- CSNDKRR callable service 636
- CSNDKRW callable service 638
- CSNDKTC callable service 596
- CSNDPKB callable service 583
- CSNDPKD callable service 228
- CSNDPKE callable service 234
- CSNDPKG callable service 571
- CSNDPKI callable service 578
- CSNDPKT callable service 599
- CSNDPKX callable service 604
- CSNDRKD callable service 606
- CSNDRKL callable service 609
- CSNDSBC callable service 531
- CSNDSBD callable service 536
- CSNDSYG callable service 275
- CSNDSYI callable service 283
- CSNDSYI2 callable service 291
- CSNDSYX callable service 269
- CSNDTBC callable service 298
- CSNECKI 104

- CSNECKM 218
- CSNEKGN 142
- CSNEOWH 450
- CSNERNG 243
- CSNFKRC 632
- CSNFKRD 634
- CSNFPKB 584
- CSNFPKD 229
- CSNFPKE 234
- CSNFPKG 572
- CSNFPKI 578
- CSNFPKX 604
- CSNFRKD 606
- CSNFRKL 609
- CSNFSYI2 callable service 291
- CUSP processing rule 374, 384, 942
- CVV Key Combine callable service (CSNBCKC and CSNECKC) 491

## D

- data
  - deciphering 370
  - enciphering 379
  - enciphering and deciphering 55
  - encoding and decoding 56
  - protecting 351
- data array parameter
  - clear PIN generate alternate callable service 489
  - clear PIN generate callable service 482
  - encrypted PIN generate callable service 499
  - encrypted PIN verify callable service 512
- data integrity
  - ensuring 56
  - verifying 423
- data key
  - exporting 116
  - importing 104
  - reenciphering 116
- data key export callable service (CSNBKX)
  - format 116
  - overview 28
  - parameters 116
  - syntax 116
- data key import callable service (CSNBKIM)
  - format 119
  - overview 28
  - parameters 119
  - syntax 119
- DATA key type 25
- data length parameter
  - diversified key generate callable service 126
- data space
  - callable services that use data in data spaces 5
- data-encrypting key 20
- data-translation key 20, 354
- DATAM key type 25
- DATAMV key type 25
- DATAXLAT key type 25

- decipher callable service (CSNBDEC or CSNBDEC1)
  - format 372
  - syntax 372
- deciphering
  - data 351, 370
  - data with clear key 377
- decode callable service (CSNBDCO)
  - format 377
  - parameters 377
  - syntax 377
- DES algorithm 15, 351
- DES external key token format 839
- DES internal key token format 836
- destination identifier 54
- digital signature generate callable service (CSNDDSG)
  - format 557
  - overview 87
  - parameters 557
  - syntax 557
- digital signature verify callable service (CSNDDSV)
  - format 566
  - overview 88
  - parameters 566
  - syntax 566
- diversified key generate callable service (CSNBKDG)
  - format 122
  - overview 28
  - parameters 122
  - syntax 122
- double-length key
  - using 22
- DSS private external key token 870
- DSS private internal key token 871
- DSS public token 869
- dynamic CKDS update callable services
  - description 49

## E

- EBCDIC to ASCII conversion
  - table 959
- ECDSA algorithm 83
- ECI-1 507
- ECI-2 PIN block format 471, 930
- ECI-3 PIN block format 471, 931
- ECI-4 507
- EDC
  - generating 685
- electronic code book (ECB) 351
  - mode 352
- Elliptic Curve Digital Signature Algorithm (ECDSA) 83
- encipher callable service (CSNBENC or CSNBENC1)
  - format 381
  - parameters 382
  - syntax 381
- enciphered
  - key 142, 262, 351
  - under master key 165
- enciphering
  - data 351, 379
  - string with clear key 387



- encode callable service (CSNBECO)
  - format 387
  - parameters 387
  - syntax 387
- encrypted PIN block parameter
  - clear PIN generate alternate callable service 487
  - encrypted PIN verify callable service 511
- encrypted PIN generate callable service (CSNBEPG)
  - format 497
  - syntax 497
- encrypted PIN generate service (CSNBEPG)
  - parameters 497
- encrypted PIN translate callable service (CSNBPTR) 502
  - extraction rules 931
  - format 502
  - parameters 503
  - syntax 502
- encrypted PIN verification callable service (CSNBPVVR)
  - extraction rules 931
- encrypted PIN verify callable service (CSNBPVVR)
  - format 509
  - parameters 510
  - syntax 509
- ensuring data integrity and authenticity 56
- error detection code (EDC)
  - generating 685
- EX key form 67
- examples of callable services 909
- EXEX key form 69
- exit data 7
- exit data length 7
- exit, installation 8
- exportable key form 17
  - definition 17
  - generating 67
  - value 143
- exporter key identifier parameter
  - data key export callable service 118
  - key export callable service 138
- EXPORTER key type 25
- exporter key-encrypting key 21
  - any DES key 137
  - enciphering data key 116
- exporting keys
  - trusted blocks 42
- external key token 8, 18, 93
  - DES 839
  - PKA 95
    - DSS private 870
    - RSA private 852
- extra\_data parameter
  - remote key export callable service 253
- extra\_data\_length parameter
  - remote key export callable service 253
- extraction rules, PIN 931

## F

- FEATURE=CRYPTO keyword
  - SCHEDULE macro 10
- form parameter
  - random number generate callable service 244
- format control 474
- formats, PIN 60
- functions of
  - cryptographic keys 15
  - ICSF 15

## G

- GBP-PIN algorithm 511
- GBP-PINO algorithm 511
- generated key identifier 1 parameter
  - key generate callable service 149
- generated key identifier 2 parameter
  - key generate callable service 150
- generated key identifier parameter
  - diversified key generate callable service 126
- generating an error detection code (EDC) 685
- generating encrypted keys 142
- generating key identifier parameter
  - diversified key generate callable service 126
- generating keys
  - remote key export 45
- German Banking Pool PIN algorithm 933
- get attribute value callable service (CSFPGAV)
  - format 722
  - parameters 722
  - syntax 722

## H

- hash length parameter
  - digital signature generate callable service 560
  - digital signature verify callable service 568
  - one-way hash generate callable service 453
- hash parameter
  - digital signature generate callable service 560
  - digital signature verify callable service 568
  - one-way hash generate callable service 453
- HEXDIGIT PIN extraction method
  - keyword 472
- high-level languages 4
- HMAC
  - keys 21

## I

- IBM 3624 480, 509
- IBM 4700 processing rule 941

- IBM GBP 480, 509
- IBM-4700 PIN block format 930
- IBM-PIN algorithm 511
- IBM-PINO algorithm 511
- ICSF
  - functions 15
  - overview 15
- ICSF Query Algorithm (CSFIQA)
  - parameters 647
  - syntax 647
- ICSF Query Algorithm (CSFIQA))
  - format 647
- ICSF Query Algorithm Service (CSFIQA)
  - overview 65
- ICSF Query Facility (CSFIQF)
  - parameters 652
  - syntax 652
- ICSF Query Facility (CSFIQF))
  - format 652
- ICSF Query Facility Service (CSFIQF)
  - overview 65
- IEAAFFN callable service (affinity) 10
- IM key form 67
- IMEX key form 69
- IMIM key form 68
- importable key form 17
  - definition 17
  - generating 67
  - value 143
- imported key identifier length parameter
  - multiple secure key import callable service 225
- imported key identifier parameter
  - multiple secure key import callable service 225
- importer key identifier parameter
  - key import callable service 167
  - secure key import callable service 262
- IMPORTER key type 25
- importer key-encrypting key 21
  - enciphering clear key 260, 262
- importer\_key\_identifier parameter
  - remote key export callable service 252
- importer\_key\_length parameter
  - remote key export callable service 251, 252, 253
- importing a non-exportable key 241
- improving performance using partial notarization 949
- INBK PIN 468, 480
- INBK-PIN 509
- Information Protection System (IPS) 942
- initial chaining vector (ICV)
  - description 352, 940
- initialization vector in parameter
  - ciphertext translate callable service 356
- initialization vector out parameter
  - ciphertext translate callable service 356
- initialization vector parameter
  - cryptographic variable encipher callable service 115
  - decipher callable service 374
  - encipher callable service 383

- initialization vector parameter *(continued)*
  - key token build callable service 198
- input data transport key 354
- input KEK key identifier parameter
  - key translate callable service 210
- input PIN profile parameter
  - clear PIN generate alternate callable service 487
  - encrypted PIN translate callable service 503
  - encrypted PIN verify callable service 510
- input PIN-encrypting key identifier parameter
  - encrypted PIN translate callable service 503
  - encrypted PIN verify callable service 510
- input\_block parameter
  - trusted block create callable service 299
- input\_block\_identifier parameter
  - trusted block create callable service 299
- installation exit
  - post-processing 8
  - preprocessing 8
- installation-defined callable service 15
- Integrated Cryptographic Service Facility (ICSF)
  - description xxiii
- Integrity 879
- Interbank PIN 79, 468, 480, 509
- internal key token 8, 18, 93, 94
  - aes; 836
  - DES 836, 837
  - PKA
    - DSS private 871
    - RSA private 860, 861, 862, 873, 876, 877
- invocation requirements 9
- IPINENC key type 25, 503
- IPS processing rule 374, 384, 942
- ISO-0 PIN block format 471
- ISO-1 PIN block format 471, 930
- ISO-2 PIN block format 471, 930
- ISO-3 PIN block format 471

## J

- JCL statements, sample 13

## K

- KEK key identifier parameter
  - control vector translate callable service 111
- KEK key identifier 1 parameter
  - key generate callable service 148
- KEK key identifier 2 parameter
  - key generate callable service 149
- KEK key identifier parameter
  - key test extended callable service 191
  - prohibit export extended callable service 242

- KEK key identifier parameter *(continued)*
  - transform CDMF key callable service 297
- key array parameter
  - control vector translate callable service 112
- key array right parameter
  - control vector translate callable service 112
- Key Data Set management 613
  - callable services 613
- key encrypting key identifier parameter 278
- key export callable service (CSNBKEX)
  - format 137
  - overview 28
  - parameters 137
  - syntax 137
- key flow 17
- key form
  - combinations for a key pair 151
  - combinations with key type 151
  - definition 17
  - exportable 17
  - importable 17
  - operational 17
  - value 143
- key form parameter
  - key generate callable service 143
  - secure key import callable service 262
- key generate callable service (CSNBKGN)
  - format 142
  - overview 27
  - parameters 142
  - syntax 142
  - using 67
- key generator utility program (KGUP)
  - description 27
- key identifier 8
  - PKA keys 93
- key identifier in parameter
  - ciphertext translate callable service 356
- key identifier length parameter
  - multiple clear key import callable service 220
  - Symmetric MAC generate callable service 456
  - symmetric MAC verify callable service 460
- key identifier out parameter
  - ciphertext translate callable service 356
- key identifier parameter
  - clear key import callable service 105
  - decipher callable service 373
  - diversified key generate callable service 126
  - encipher callable service 382
  - key test callable service 182
  - key test extended callable service 190
  - MAC generate callable service 436
  - MAC generation callable service 441
  - multiple clear key import callable service 220

- key identifier parameter *(continued)*
  - secure key import callable service 262
  - Symmetric MAC generate callable service 456
  - symmetric MAC verify callable service 461
- key import callable service (CSNBKIM)
  - format 165
  - overview 29
  - parameters 165
  - syntax 165
- key label 8, 94
  - security considerations 9
- key length parameter
  - key generate callable service 144
- key management
  - ANSI X9.17 standard 685
- key pair 151
- key part import callable service (CSNBKPI)
  - format 171
  - overview 29
  - parameters 171
  - syntax 171
- key record delete callable service (CSNBKRD)
  - overview 50
- key test callable service (CSNBKYT and CSNBKYTX)
  - overview 29
- key test callable service (CSNBKYT)
  - parameters 179
- key test callable services (CSNBKYT)
  - format 179
  - syntax 179
- key test extended callable service (CSNBKYTX)
  - parameters 188
- key test extended callable services (CSNBKYTX)
  - syntax 188
- key test extended callable services (CSNBKYTX)
  - format 188
- key token 17, 94
  - aes; internal 836
  - DES
    - external 839
    - internal 836
    - null 841
  - DES internal 837
  - external 18
  - internal 18, 94
  - null 19
  - PKA 91
    - DSS private external 870
    - DSS private internal 871
    - DSS public 869
    - null 851
    - RSA 1024-bit modulus-exponent
      - private external 853
    - RSA 1024-bit private internal 861, 862
    - RSA 2048-bit Chinese remainder theorem private internal 867

- key token *(continued)*
  - PKA *(continued)*
    - RSA 4096-bit Chinese remainder theorem private external 858
    - RSA 4096-bit modulus-exponent private external 854
    - RSA private external 852
    - RSA private internal 860, 873, 876, 877
    - RSA public 851
    - PKA external 95
  - key token build callable service (CSNBKTB and CSNEKTB) overview 32
  - key token build callable service (CSNBKTB)
    - format 192
    - overview 29
    - parameters 192
    - syntax 192
  - key translate (CSNBKTR) parameters 209
  - key translate callable service (CSNBKTR)
    - format 209
    - overview 29
    - syntax 209
  - Key Translate2 callable service (CSNBKTR2 and CSNEKTR2)
    - format 212
    - parameters 212
    - syntax 212
  - key type 1 68, 69
  - key type 1 parameter
    - key generate callable service 147
  - key type 2 68, 69
  - key type 2 parameter
    - key generate callable service 148
  - key type parameter
    - key export callable service 138
    - key import callable service 166
    - key token build callable service 194
    - secure key import callable service 262
    - user derived key callable service 348
  - key value structure length
    - parameter 586
  - key value structure parameter 586
  - key\_check\_length parameter
    - remote key export callable service 254
  - key\_check\_parameters parameter
    - remote key export callable service 254
  - key\_check\_parameters\_length parameter
    - remote key export callable service 253
  - key\_check\_value parameter
    - remote key export callable service 254
  - key-encrypting key 21
    - definition 16
    - description 21
    - exporter 116, 137
    - importer 260
  - keyboard
    - navigation 977
    - PF keys 977

- keyboard *(continued)*
  - shortcut keys 977
- keys
  - ANSI X9.17 key-encrypting 22
  - changing CDMF DATA key to transformed shortened DES 295
  - CIPHER 21
  - clear 25, 260
  - control vector 16, 19
  - create
    - values for keys 31
  - creating 11
  - cryptographic, functions of 15
  - data key
    - exporting 116
    - importing 104
    - reenciphering 116
  - data-encrypting 20
  - data-translation 20
  - double-length 68, 69
  - enciphered 262
  - export
    - values for keys 30
  - exporter key-encrypting 21
  - forms 17
  - generating
    - encrypted 142
    - values for keys 30
  - held in applications 22
  - held in CKDS 22
  - HMAC 21
  - importer key-encrypting 21
  - key-encrypting 21
  - list of types 25
  - MAC 21
  - managing 103
  - master key variant 16
  - master, DES 19
  - master, AES 20
  - pair 68, 69
  - parity 104
  - PIN 21
  - PIN-encrypting key 502
  - PKA master 83
    - Key Management Master Key (KMMK) 83
    - Signature Master Key (SMK) 83
  - possible forms 29
  - protecting 351
  - reenciphered 165
  - reenciphering 137
  - separation 16
  - single-length 67, 68
  - transport 21
  - transport key variant 16
  - types of 19
  - using 11
  - VISA PVV
    - generating 485

## L

- languages, high-level 4
- large data object 941
- linking callable services 13
- local enciphered key token
  - parameter 278

## M

- MAC
  - generation callable service 57
  - keys 21
  - length keywords 436, 442, 457, 461
  - managing 57
  - verification callable service 58
- MAC generate callable service (CSNBGMN or CSNBGMN1)
  - format 435
  - parameters 435
  - syntax 435
- MAC key type 25
- mac length parameter
  - Symmetric MAC generate callable service 458
  - symmetric MAC verify callable service 462
- mac parameter
  - MAC generate callable service 437
  - MAC verify callable service 443
  - Symmetric MAC generate callable service 458
  - symmetric MAC verify callable service 462
- MAC verify callable service (CSNBMVR or CSNBMVR1)
  - format 440
  - parameters 441
  - syntax 440
- MACVER key type 25
- managing keys 103
- mask array left parameter
  - control vector translate callable service 112
- mask array preparation 904
- mask array right parameter
  - control vector translate callable service 112
- master key
  - AES 20
  - changing
    - possible effect on internal key tokens 18
  - enciphered key 165
- master key variant 16
- master key, DES 19
- MDC
  - generate callable service 59
  - length keywords 448
  - managing 59
- mdc parameter
  - MDC generate callable service 448
- message authentication
  - definition 57
- message authentication code (MAC)
  - description 423
  - generating 423, 434, 454
  - verifying 423, 440, 459
- messages
  - authenticating 423
- migration consideration
  - return codes from PCF macros 7
- mode, special secure 10
- modes of operation 351
- modification detection
  - definition 59

- modification detection code (MDC)
  - generating 424, 445
  - verifying 424
- multiple clear key import callable service (CSNBCKM) 218
  - format 218
  - overview 30, 32
  - parameters 218
  - syntax 218
- multiple node network 354
- multiple secure key import callable service (CSNBSKM and CSNESKM)
  - overview 32
- multiple secure key import callable service (CSNBSKM) 222
  - format 222
  - overview 30
  - parameters 222
  - syntax 222

## N

- navigation
  - keyboard 977
- notarization 54
- Notices 979
- null key token 19
  - format 841, 851
- number, generated 243

## O

- object ion key (OPK) 888
- offsetting 54, 949
- one-way hash generate callable service (CSNBOWH and CSNBOWH1)
  - overview 59
- one-way hash generate callable service (CSNBOWH, CSNEOWH and CSNBOWH1)
  - format 450
  - parameters 450
  - syntax 450
- OP key form 67
- operational key form 17
  - definition 17
  - generating 67
  - value 143
- OPEX key form 68
- OPIM key form 68
- OPINENC key type 25, 503
- OPK, object protection key 888
- OPOP key form 68
- origin identifier 54
- output chaining vector (OCV)
  - description 940
- output data transport key 354
- output KEK key identifier parameter
  - key translate callable service 210
- output PIN profile parameter
  - encrypted PIN translate callable service 505
- output PIN-encrypt translation key identifier parameter
  - encrypted PIN translate callable service 503
- overview of callable services 3

## P

- pad character parameter
  - encipher callable service 385
  - key token build callable service 198
- pad digit 476
  - format 475
- PADDIGIT PIN extraction method
  - keyword 472
- padding schemes 371, 380
- PADEXIST PIN extraction method
  - keyword 472
- pair of keys 68, 69
- PAN data in parameter
  - encrypted PIN translate callable service 504
- PAN data out parameter
  - encrypted PIN translate callable service 506
- PAN data parameter
  - clear PIN encrypt callable service 479
  - clear PIN generate alternate callable service 487
  - encrypted PIN generate callable service 500
  - encrypted PIN verify callable service 511
- parameter
  - attribute definitions 5
  - definitions 6
  - direction 6
  - exit data 7
  - exit data length 7
  - reason code 7
  - return code 7
  - type 6
- parity of key 104, 260
  - adjusting 181, 190
  - EVEN 245
  - ODD 245
- partial notarization 54, 949
  - calculation for a double-length
    - AKEK 950
  - calculation for a single-length
    - AKEK 950
- PCF
  - key separation 16
  - keys 22
  - macros 7
  - migration consideration 7
- PCI interface callable service (CSFPCI)
  - parameters 677
  - syntax 677
- performance considerations 10
- personal account number (PAN)
  - for encrypted PIN translate 504
  - for encrypted PIN verify 511
- personal authentication
  - definition 59
- personal identification number (PIN)
  - 3624 PIN generation algorithm 932
  - 3624 PIN verification algorithm 935
  - algorithm value 488, 511
  - algorithms 60, 467, 480
- personal identification number (PIN)
  - (continued)
  - block format 468, 502
  - clear PIN encrypt callable service 60
  - clear PIN generate alternate callable service 61, 485
  - definition 59
  - description 465
  - detailed algorithms 932
  - encrypted generation callable service 61
  - encrypting key 468, 502
  - extraction rules 931
  - formats 60
  - GBP PIN verification algorithm 937
  - generating 467, 480
    - from encrypted PIN block 467
  - generation callable service 61, 480
  - German Banking Pool PIN
    - algorithm 933
  - keys 21
  - managing 59
  - PIN offset generation algorithm 934
  - PVV generation algorithm 938
  - PVV verification algorithm 939
  - translating 467
  - translation callable service 61, 502
  - translation of, in networks 466
  - using 465
  - verification callable service 62, 509
  - verifying 467, 509
  - VISA PIN algorithm 938
- PIN block format
  - 3621 930
  - 3624 930
  - additional names 507
  - ANSI X9.8 929
  - detail 929
  - ECI-2 930
  - ECI-3 931
  - format values 471
  - IBM-4700 930
  - ISO-1 930
  - ISO-2 930
  - PIN extraction method keywords 472
  - VISA-2 930
  - VISA-3 930
- PIN block in parameter
  - encrypted PIN translate callable service 504
- PIN block out parameter
  - encrypted PIN translate callable service 506
- PIN block variant constant (PBVC)
  - description 474, 490
  - for clear PIN generate alternate 490
  - for encrypted PIN translate 507
  - for PIN verification 514
- PIN Change/Unblock
  - format 517
  - syntax 517
- PIN Change/Unblock (CSNBPCU) 516
  - parameters 517
- PIN check length parameter 488
  - clear PIN encrypt callable service 479

- PIN check length parameter *(continued)*
  - clear PIN generate callable service 482
  - PIN verify callable service 512
- PIN encryption key identifier parameter 486
- PIN encrypting key identifier parameter
  - clear PIN encrypt callable service 478
- PIN generating key identifier parameter
  - encrypted PIN generate callable service 498
- PIN keys 21
- PIN length parameter
  - clear PIN generate callable service 478, 482
  - encrypted PIN generate callable service 499
- PIN notation 929
- PIN profile 471
  - description 503, 510
- PIN profile parameter 487
  - encrypted PIN generate callable service 500
- PIN validation value (PVV) 480
- PIN verifying key identifier parameter
  - encrypted PIN verify callable service 510
- PINBLOCK PIN extraction method
  - keyword 472
- PINGEN key type 25
- PINLEN04 PIN extraction method
  - keyword 472
- PINLEN12 PIN extraction method
  - keyword 472
- PINVER key type 25
- PKA decrypt callable service (CSNDPKD)
  - overview 52
- PKA decrypt callable servicec 228
- PKA encrypt callable service (CSNDPKE)
  - overview 52
- PKA encrypt callable servicec 234
- PKA external key token 95
- PKA key generate callable service (CSNDPKG)
  - format 571
  - parameters 571
  - syntax 571
- PKA key import callable service (CSNDPKI)
  - format 578
  - overview 88
  - parameters 578
  - syntax 578
- PKA key token 91
  - external 95
  - record format
    - DSS private external 870
    - DSS private internal 871
    - DSS public 869
    - RSA 1024-bit modulus-exponent
      - private external 853
    - RSA 1024-bit private internal 861, 862
    - RSA 2048-bit Chinese remainder theorem private internal 867
- PKA key token *(continued)*
  - record format *(continued)*
    - RSA 4096-bit Chinese remainder theorem private external 858
    - RSA 4096-bit modulus-exponent private external 854
    - RSA private external 852
    - RSA private internal 860, 873, 876, 877
    - RSA public 851
- PKA key token build callable service (CSNDPKB)
  - format 583
  - overview 88
  - parameters 583
  - syntax 583
- PKA key token change (CSNDKTC)
  - parameters 597
- PKA key token change callable service (CSNDKTC and CSNFKTC)
  - overview 89
- PKA key token change callable service (CSNDKTC) 596
- PKA key translate callable service (CSNDPKT)
  - format 599
  - parameters 599
  - syntax 599
- PKA master key 85
- PKA private key identifier length
  - parameter 559
- PKA private key identifier
  - parameter 559
- PKA public key extract callable service (CSNDPKX)
  - format 604
  - overview 89
  - parameters 604
  - syntax 604
- PKA public key identifier length
  - parameter 568
- PKA public key identifier parameter 568
- PKA92 key format and encryption
  - process 947
- pkcs #11
  - using 97
- PKCS #11
  - callable services 97, 709
  - objects 709
  - tokens 709
  - using 709
- PKDS key record create callable service (CSNDKRC)
  - format 632
  - parameters 632
  - syntax 632
- PKDS key record delete callable service (CSNDKRD) 634
  - format 634
  - parameters 634
  - syntax 634
- PKDS key record read callable service (CSNDKRR) 636
  - format 636
  - parameters 637
  - syntax 636
- PKDS key record write callable service (CSNDKRW) 638
  - format 639
  - parameters 639
  - syntax 639
- PKSC interface 682
- PKSC interface callable service (CSFPKSC)
  - parameters 682
  - syntax 682
- plaintext
  - enciphering 351
  - encoding 387
  - field 387, 411, 421
- plaintext parameter
  - cryptographic variable encipher callable service 115
- post-processing exit 8
- preprocessing exit 8
- privacy 55
- private external key token
  - DSS 870
  - RSA 852
- private internal key token
  - DSS 871
  - RSA 860, 861, 862, 873, 876, 877
- private key name length parameter 593
- private key name parameter 594
- processing rule
  - 4700-PAD 374, 384
  - ANSI X3.106 940
  - ANSI X9.23 374, 384, 941
  - CBC 374, 384
  - cipher 940
  - cipher last block 941
  - CUSP 942
  - CUSP/IPS 374, 384
  - decipher 374
  - encipher 384
  - GBP-PIN 482
  - GBP-PINO 482
  - IBM 4700 941
  - IBM-PIN 482
  - IBM-PINO 482
  - INBK-PIN 482
  - IPS 942
  - recommendations for encipher 383
  - segmenting 941
  - VISA-PVV 482
- prohibit export (CSNBPEX) 239
- prohibit export callable service (CSNBPEX)
  - format 239
  - overview 30
  - syntax 239
- prohibit export extended callable service (CSNBPEXX)
  - format 241
  - overview 30
  - parameters 241
  - syntax 241
- protecting data and keys 351
- public key token
  - DSS 869
  - RSA 851



# R

- RACF authorization 9
- random number generate callable service (CSNBRNG)
  - format 243
  - overview 30
  - parameters 243
  - syntax 243
- random number generate callable service (CSNBRNGL)
  - format 243
  - parameters 243
  - syntax 243
- random number parameter
  - key test callable service 182
  - key test extended callable service 190
  - random number generate callable service 246
- random\_number\_length
  - random number generate callable service 245
- reason codes 7, 12
- reason codes for ICSF
  - for return code 0 (0) 782
  - for return code 10 (16) 833
  - for return code 4 (4) 783
  - for return code 8 (8) 786
  - for return code C (12) 821
- recommendations for encipher processing
  - rules 383
- record chaining 942
- reenciphered
  - key 165
- reenciphering
  - data-encrypting key 116
  - PIN block 502
- remote key distribution 35
  - benefits 48
  - scenario 47
- remote key export
  - exporting keys 42
  - generating keys 45
- remote key export callable service (CSNDRKX)
  - format 247
  - overview 30
  - parameters 247
  - syntax 247
- remote key loading 35
  - example 35
  - new method 35
- remote key-loading
  - CCA API changes 40
- reserved
  - random number generate callable service 245
- reserved data length parameter
  - symmetric MAC generate callable service 458
  - symmetric MAC verify callable service 462
- reserved data parameter
  - Symmetric MAC generate callable service 458
  - symmetric MAC verify callable service 462
- reserved parameter
  - control vector generate callable service 109, 210
- reserved\_length
  - random number generate callable service 245
- retained key delete callable service (CSNDRKD)
  - format 606
  - overview 90
  - parameters 606
  - syntax 606
- retained key list callable service (CSNDRKL)
  - format 609
  - overview 90
  - parameters 609
  - syntax 609
- retained private keys
  - overview 90
- return codes 7, 12
  - from PCF macros
    - migration consideration 7
- returned PVV parameter 489
- returned result parameter
  - clear PIN generate callable service 483
- Rivest-Shamir-Adleman (RSA)
  - algorithm 83
- RKX key token 40
- RKX key-token 839
- RSA 1024-bit private internal key
  - token 861, 862
- RSA algorithm 83
- RSA enciphered key length parameter
  - symmetric key generate callable service 279
  - symmetric key import callable service 285
- RSA enciphered key parameter
  - symmetric key generate callable service 279
  - symmetric key import callable service 285
- RSA private external Chinese remainder theorem key token 858
- RSA private external key token 852
- RSA private external modulus-exponent key token 853, 854
- RSA private internal Chinese remainder theorem key token 867
- RSA private internal key token 860, 873, 876, 877
- RSA private key identifier 285
- RSA private key identifier length 285
- RSA public key identifier length
  - parameter
    - for symmetric key generate 278
- RSA public key identifier parameter 278
- RSA public token 851
- rule array count parameter
  - clear PIN encrypt callable service 478
  - Clear PIN encrypt callable service 112, 498
  - clear PIN generate alternate callable service 487
- rule array count parameter *(continued)*
  - clear PIN generate callable service 481
  - control vector translate callable service 112
  - decipher callable service 374
  - digital signature generate callable service 558
  - digital signature verify callable service 567
  - diversified key generate callable service 124
  - encipher callable service 383
  - encrypted PIN translate callable service 504
  - encrypted PIN verify callable service 511
  - key test callable service 181
  - key test extended callable service 189
  - key token build callable service 195
  - MAC generate callable service 436
  - MAC generation callable service 442
  - MDC generate callable service 447
  - one-way hash generate callable service 451
  - PKA key generate callable service 573, 600
  - PKA key import callable service 579
  - PKA key token build callable service 585
  - PKA public key extract callable service 604
  - symmetric key export callable service 270
  - symmetric key generate callable service 276
  - symmetric key import callable service 284
  - Symmetric MAC generate callable service 456
  - Symmetric MAC verify callable service 461
  - transform CDMF key callable service 249, 296
  - trusted block create callable service 299
  - user derived key callable service 348

- rule array parameter *(continued)*
  - encrypted PIN generate callable service 498
  - encrypted PIN translate callable service 504
  - encrypted PIN verify callable service 511
  - key test callable service 181
  - key test extended callable service 190
  - key token build callable service 195
  - MAC generate callable service 436
  - MAC generation callable service 442
  - MDC generate callable service 448
  - one-way hash generate callable service 451
  - PKA key generate callable service 573, 601
  - PKA key token build callable service 585
  - PKA public key extract callable service 605
  - random number generate callable service 245
  - symmetric key export callable service 270
  - symmetric key generate callable service 276
  - symmetric key import callable service 284
  - Symmetric MAC generate callable service 457
  - symmetric MAC verify callable service 461
  - transform CDMF key callable service 249, 296
  - trusted block create callable service 299
  - user derived key callable service 348
- rule\_array\_count
  - ICSF query service callable service 648, 653
  - random number generate callable service 245
- rule\_id parameter
  - remote key export callable service 251
- rule\_id\_length parameter
  - remote key export callable service 251

## S

- sample JCL statements 13
- SCHEDULE macro
  - FEATURE=CRYPTO keyword 10
- screen readers
  - accessibility 979
- SCSFMOD0 module 13
- section sequence, trusted block 878
- secure key import callable service (CSNBSKI)
  - format 260
  - overview 30
  - parameters 260
  - syntax 260
- secure messaging
  - overview 64

- secure messaging for keys callable service (CSNBSKY)
  - format 523, 596
  - parameters 523, 543
  - syntax 523, 596
- Secure messaging for keys callable service (CSNBSKY) 523
- secure messaging for PINs callable service (CSNBSPN)
  - format 527
  - parameters 527
  - syntax 527
- Secure messaging for PINs callable service (CSNBSPN) 526
- Secure Sockets Layer (SSL) 51
- security considerations 9
- segmenting
  - control keywords 436, 442, 448, 457, 461
  - definition 941
  - rule, large data object 941
- sequence number parameter
  - encrypted PIN translate callable service 506
- sequences of callable service 66
- set attribute value callable service (CSFPSAV)
  - format 749
  - parameters 749
  - syntax 749
- SET block compose callable service (CSNDSBC) 531
  - format 531
  - overview 91
  - parameters 531
  - syntax 531
- SET block decompose callable service (CSNDSBD) 536
  - format 537
  - overview 91
  - parameters 537
  - syntax 537
- SET protocol 91
- SET Secure Electronic Transaction 91
- short blocks 380
- shortcut keys 977
- signature bit length parameter 560
- signature field length parameter
  - digital signature generate callable service 560
  - digital signature verify callable service 568
- signature field parameter
  - digital signature generate callable service 561
  - digital signature verify callable service 569
- single-length key
  - purpose 67, 68
  - using 22
- source key identifier length parameter
  - PKA key import callable service 579
  - PKA public key extract callable service 605
- source key identifier parameter
  - data key export callable service 117
  - key export callable service 138

- source key identifier parameter *(continued)*
  - key import callable service 167
  - PKA key import callable service 580
  - PKA public key extract callable service 605
  - transform CDMF key callable service 297
- source key token length parameter
  - prohibit export extended callable service 242
- source text parameter
  - character/nibble conversion callable service 644
  - code conversion callable service 646
  - X9.9 data editing callable service 674
- source\_key\_length parameter
  - remote key export callable service 252
- special secure mode 10
- SRB, scheduling 10
- SSL support 51
- sym\_encrypted\_key\_length parameter
  - remote key export callable service 253
- symmetric algorithm decipher callable service (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1)
  - format 389
  - parameters 389
  - syntax 389
- symmetric algorithm encipher callable service (CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1)
  - format 396
  - syntax 396
- symmetric algorithm encipher callable service CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1)
  - parameters 396
- symmetric key decipher callable service (CSNBSYD and CSNBSYD1)
  - format 402
  - parameters 402
  - syntax 402
- symmetric key encipher callable service (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1)
  - format 412
  - parameters 412
  - syntax 412
- symmetric key export callable service (CSNDSYX and CSNFSYX)
  - overview 33
- symmetric key export callable service (CSNDSYX)
  - format 269
  - overview 31
  - parameters 269
  - syntax 269
- symmetric key generate callable service (CSNDSYG and CSNFSYG)
  - overview 33
- symmetric key generate callable service (CSNDSYG)
  - format 275
  - overview 31

- symmetric key generate callable service (CSNDSYG) (*continued*)
  - parameters 275
  - syntax 275
- symmetric key import callable service (CSNDSYI and CSNFSYI)
  - overview 33
- symmetric key import callable service (CSNDSYI)
  - format 283
  - overview 31
  - parameters 283
  - syntax 283
- Symmetric MAC
  - generation callable service 58
  - verify callable service 58
- Symmetric MAC generate callable service (CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1)
  - format 455
  - parameters 455
  - syntax 455
- Symmetric MAC verify callable service (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1)
  - format 459
  - parameters 460
  - syntax 459
- syntax for callable service 4

## T

- target key identifier length
  - parameter 580
- target key identifier parameter 580
  - data key export callable service 118
  - key export callable service 139
  - key import callable service 167
  - symmetric key import callable service 286
  - transform CDMF key callable service 297
- target key token parameter
  - encrypted PIN generate callable service 113
- target public key token length
  - parameter 605
- target public key token parameter 605
- target text parameter
  - character/nibble conversion callable service 644, 649, 651, 672
  - code conversion callable service 646
  - ICSF query facility callable service 654
  - X9.9 data editing callable service 674
- text id in parameter
  - ciphertext translate callable service 356
  - MAC generate callable service 438
  - MAC verify callable service 443
  - MDC generate callable service 449
  - one-way hash generate callable service 453
  - symmetric MAC verify callable service 463
- text id out parameter
  - ciphertext translate callable service 357
- text in parameter
  - ciphertext translate callable service 356
- text length parameter
  - character/nibble conversion callable service 644
  - ciphertext translate callable service 356
  - code conversion callable service 646
  - cryptographic variable encipher callable service 115
  - decipher callable service 373
  - encipher callable service 383
  - MAC generate callable service 436
  - MAC generation callable service 441
  - MDC generate callable service 447
  - one-way hash generate callable service 452
  - Symmetric MAC generate callable service 456
  - Symmetric MAC verify callable service 461
  - X9.9 data editing callable service 674
- text out parameter
  - ciphertext translate callable service 356
- text parameter
  - MAC generate callable service 436
  - MAC generation callable service 442
  - MDC generate callable service 447
  - one-way hash generate callable service 452
  - Symmetric MAC generate callable service 456
  - symmetric MAC verify callable service 461
- text, translating 354, 357
- TKE
  - overview 64
- token record create callable service (CSFPTRC)
  - format 762
  - parameters 762
  - syntax 762
- token record delete callable service (CSFPTRD)
  - format 766
  - parameters 766
  - syntax 766
- token record list callable service (CSFPTRL)
  - format 768
  - parameters 768
  - syntax 768
- token validation value (TVV) 836
- TR-31 export callable service (CSNBT31X and CSNET31X) 301
- TR-31 import callable service (CSNBT31I and CSNET31I) 315
- TR-31 Optional Data Build callable services (CSNBT31O and CSNET31O) 329

- TR-31 Optional Data Read callable services (CSNBT31R and CSNET31R) 332
- TR-31 Parse callable service (CSNBT31P and CSNET31P) 336
- trailing short blocks 380
- transaction validation callable service (CSNBSKY)
  - format 543
  - syntax 543
- transaction validation callable service (CSNBTRV) 542
- transform CDMF key algorithm 951
- transform CDMF key callable service (CSNBTCK)
  - format 295
  - overview 31
  - parameters 295
  - syntax 295
- transformed shortened DES key 295
- transport key 21
- transport key variant 16
- transport\_key\_identifier parameter
  - remote key export callable service 251
  - trusted block create callable service 300
- transport\_key\_length parameter
  - remote key export callable service 251
- trusted block 36
- trusted block create
  - callable service 298
- trusted block create callable service (CSNDTBC)
  - format 298
  - overview 31
  - parameters 298
  - syntax 298
- trusted block key token
  - trusted block key token
    - trusted block key token 877
- trusted blocks
  - CCA API changes 40
  - creating 41
  - exporting keys 42
  - using 41
- Trusted Key Entry
  - overview 64
- trusted\_block\_identifier parameter
  - trusted block create callable service 249, 300
- trusted\_block\_length parameter
  - remote key export callable service 249
  - trusted block create callable service 300
- types of keys 19

## U

- UKPT
  - format 476
- user derived key
  - generating 347
  - processing rules 348



- user interface
  - ISPF 977
  - TSO/E 977
- utilities
  - character/nibble conversion 643
  - code conversion 645
  - ICSF Query Algorithm 647
  - ICSF Query Facility 652
  - key token build 192
  - PKA key token build 583
  - X9.9 data editing 673

## V

- V1R13 new information xxxi
- verification pattern parameter 182, 190
- verification pattern, generating and
  - verifying 179, 188
- verifying data integrity and
  - authenticity 423
- VISA CVV service generate callable
  - service (CSNBCSG) 546
    - format 547
    - parameters 547
    - syntax 547
- VISA CVV service verify callable service
  - (CSNBCSV) 551
    - format 551
    - parameters 551
    - syntax 551
- VISA PVV 480
  - generating 485
- VISA-1 507
- VISA-2 PIN block format 471, 930
- VISA-3 PIN block format 471, 930
- VISA-4 PIN block format 471
- VISA-PVV algorithm 488, 511
- VISAPVV4 algorithm 511

## X

- X9.9 data editing callable service
  - (CSNB9ED)
    - format 673
    - overview 65
    - parameters 673
    - syntax 673
- X9.9-1 keyword 436, 442





Product Number: 5694-A01

Printed in USA

SA22-7522-16

