

z/OS
Version 2 Release 3

*Cryptographic Services
Integrated Cryptographic Service Facility
System Programmer's Guide*



Note

Before using this information and the product it supports, read the information in [“Notices” on page 399](#).

This edition applies to ICSF FMID HCR77C0 and Version 2 Release 3 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2019-06-25

© **Copyright International Business Machines Corporation 2007, 2019.**

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

Contents

Figures.....	ix
Tables.....	xi
About this information.....	xix
Who should use this information.....	xix
How to use this information.....	xix
Where to find more information.....	xx
IBM Crypto Education.....	xxi
How to send your comments to IBM.....	xxiii
If you have a technical problem.....	xxiii
Summary of changes.....	xxv
Changes made in Cryptographic Support for z/OS V2R1 - z/OS V2R2 (FMID HCR77C0).....	xxv
Changes made in Cryptographic Support for z/OS V1R13 - z/OS V2R2 (FMID HCR77B1).....	xxvi
Changes made in Enhanced Cryptographic Support for z/OS V1R13 - z/OS V2R1 (FMID HCR77B0).....	xxvii
Changes made in Cryptographic Support for z/OS V1R13 - z/OS V2R1 (FMID HCR77A1).....	xxviii
Chapter 1. Introduction to z/OS ICSF.....	1
Features.....	1
Cryptographic hardware features.....	1
Server hardware.....	2
z/OS ICSF FMIDs.....	3
ICSF features.....	4
The Cryptographic Key Data Set (CKDS).....	5
The Public Key Data Set (PKDS).....	6
The Token Data Set (TKDS).....	6
Additional background information.....	6
Running PCF applications on z/OS ICSF.....	7
Using RMF and SMF to monitor z/OS ICSF events.....	7
Controlling access to ICSF.....	8
Steps prior to starting installation.....	8
Chapter 2. Installation, initialization, and customization.....	9
Steps for installation and initialization.....	9
Steps to customize SYS1.PARMLIB.....	10
Creating the CKDS.....	11
Creating the PKDS.....	15
Creating the TKDS.....	17
ICSF system resource planning for random number generation.....	20
Steps to create the installation options data set.....	21
Creating an ICSF CTRACE configuration data set.....	23
Steps to create the ICSF startup procedure.....	25
Steps to provide access to the ICSF panels.....	26
Requiring signature verification for ICSF module CSFINPV2.....	27
Steps to start ICSF for the first time.....	28
Customizing ICSF after the first start.....	30
Parameters in the installation options data set.....	30

Improving CKDS performance.....	51
Dispatching priority of ICSF.....	51
Creating ICSF exits and generic services.....	51
Chapter 3. Migration.....	53
Terminology.....	53
Migrating from earlier software releases.....	54
Actions to perform before installing ICSF FMID HCR77C0.....	54
Actions to perform before the first start of ICSF FMID HCR77C0.....	57
Actions to perform after the first start of ICSF FMID HCR77C0.....	59
Callable services.....	61
Identification of cryptographic features.....	65
Ensure the expected P11 master key support is available.....	66
Key store policy.....	66
ICSF key data sets.....	67
Changing the RSA master key.....	69
Migrating to 24-byte DES master key.....	70
Installation options data set.....	70
Function restrictions.....	71
CICS attachment facility.....	71
Dynamic LPA load.....	72
Special secure mode.....	72
Resource Manager Interface (RMF).....	72
System abend codes.....	72
SMF records.....	76
TKE workstation.....	77
Migrating from the IBM eServer zSeries 900.....	78
Migrating a CKDS and PKDS between a CCF system and a non-CCF system.....	79
Callable services.....	82
Functions not supported.....	83
Setup considerations.....	83
Programming considerations.....	83
Chapter 4. Operating ICSF.....	85
Starting and stopping ICSF.....	85
Starting ICSF during IPL-time.....	86
Modifying ICSF.....	87
Command syntax notation.....	87
How to read syntax diagrams.....	87
ICSF operator commands.....	90
Display ICSF.....	90
SETICSF.....	95
Using different configurations.....	105
Adding and removing cryptographic coprocessors.....	106
Adding cryptographic coprocessors.....	106
Steps for activating/deactivating cryptographic coprocessors.....	106
Steps to configure on/off cryptographic coprocessors.....	107
Steps for enabling/disabling cryptographic coprocessors.....	107
Adding and removing regional cryptographic servers.....	108
Steps to add a regional cryptographic server.....	108
Steps to remove a regional cryptographic server.....	109
Configuring ICSF to use TCP/IP for communications with regional cryptographic servers.....	109
Displaying cryptographic coprocessor status using the DISPLAY ICSF operator command.....	112
Adding a regional cryptographic server using the SETICSF operator command.....	112
Changing regional cryptographic server status using the SETICSF operator command.....	112
Performance considerations for using installation options.....	112
Dispatching priority of ICSF.....	113

VTAM session-level encryption.....	113
System SSL encryption.....	113
Access method services cryptographic option.....	113
Remote key loading.....	113
Event recording.....	114
System Management Facilities (SMF) recording.....	114
Message recording.....	122
Security considerations.....	122
Controlling the program environment.....	123
Controlling access to KGUP.....	123
Controlling access to CSFDUTIL.....	123
Controlling access to the callable services.....	123
Controlling access to cryptographic keys.....	124
Controlling access to secure key tokens.....	124
Scheduling changes for cryptographic keys.....	124
Controlling access to administrative panel functions.....	124
Obtaining RACF SMF log records.....	124
Debugging aids.....	125
Component trace.....	125
Abnormal endings.....	126
IPCS formatting routine.....	126
Detecting ICSF serialization contention conditions.....	127
ENF signals.....	129

Chapter 5. Installation exits..... 131

Types of exits.....	131
Mainline exits.....	131
Exits for the services.....	132
The PCF CKDS conversion program exit.....	132
The single-record, read-write exit.....	132
The cryptographic key data set entry retrieval exit.....	132
Security exits.....	132
The KGUP exit.....	133
Entry and return specifications.....	133
Registers at entry.....	133
Registers at return.....	134
Exits environment.....	135
Mainline exits.....	135
Service exits.....	135
CKDS entry retrieval exit.....	135
KGUP, Conversion Programs, and Single-record, Read-write exits.....	135
Security exits.....	135
Exit recovery.....	135
Mainline installation exits.....	136
Purpose and use of the exits.....	136
Environment of the exits.....	136
Installing the exits.....	137
Input.....	137
Return Codes.....	143
Services installation exits.....	144
Purpose and use of the exits.....	144
Environment of the exits.....	144
Installing the exits.....	145
Input.....	150
Return Codes.....	156
CSF_SERVICE_EXIT - ICSF callable services exit.....	157
Cryptographic key data set entry retrieval installation exit.....	159

Purpose and use of the exit.....	159
Environment of the exit.....	159
Installing the exit.....	160
Input.....	160
Return codes.....	161
PCF conversion program installation exit.....	161
Purpose and use of the exit.....	162
Environment of the exit.....	162
Installing the exit.....	162
Input.....	163
Return codes.....	164
Single-record, Read-write installation exit.....	164
Purpose and use of the exit.....	165
Environment of the exit.....	165
Installing the exit.....	165
Input.....	166
Return codes.....	167
Exit points for security installation exits.....	168
Security installation exits.....	168
Purpose and use of the exits.....	168
Environment of the exits.....	169
Installing the exits.....	169
Input.....	170
Return codes.....	171
Key generator utility program installation exit.....	172
Purpose and use of the exit.....	172
Environment of the exit.....	173
Installing the exit.....	173
Input.....	174
The SET statement.....	182
Return codes.....	182
Chapter 6. Installation-defined Callable Services.....	185
Writing a callable service.....	185
Contents of registers.....	186
Security access control checking.....	187
Checking the parameters.....	187
Link-editing the callable service.....	187
Defining a callable service.....	188
Writing a service stub.....	188
Example of a service stub.....	189
Chapter 7. Converting a CKDS from fixed length to variable length record format	195
Chapter 8. Migration from PCF to z/OS ICSF.....	199
Running PCF and z/OS ICSF on the same system.....	199
Running in compatibility mode.....	199
Running in coexistence mode.....	200
Changing the DES master key in compatibility or coexistence mode.....	201
Running in noncompatibility mode.....	201
Specifying compatibility modes during migration.....	201
Converting a PCF CKDS to ICSF format.....	202
How the PCF conversion program runs.....	202
Using the conversion program override file.....	204
Running the conversion program.....	209
Appendix A. Diagnosis reference information.....	215

Cryptographic Key Data Set (CKDS) formats.....	215
Public Key Data Set (PKDS) format.....	219
Format of the PKDS header record.....	219
Format of the PKDS record.....	220
Token data set (TKDS) format.....	221
Format of the header record of the token data set.....	221
Format of the token and object records.....	222
Common record format (KDSR).....	250
AES key token format.....	253
AES internal fixed-length key token.....	253
Token validation value.....	254
DES key token formats.....	255
DES fixed-length key token.....	255
DES external key token.....	258
External RKX DES key token.....	259
DES null key token.....	260
Variable-length symmetric key token formats.....	261
Variable-length symmetric key token.....	261
Variable-length symmetric null key token.....	286
PKA key token formats.....	287
Internal PKA tokens.....	287
PKA null key token.....	287
RSA key token formats.....	287
ECC key token format.....	310
Trusted blocks.....	315
Data areas.....	329
The Cryptographic Communication Vector Table (CCVT).....	329
The Cryptographic Communication Vector Table Extension (CCVE).....	331
Generic Service Table (CSFMGST).....	331
RMF measurements table.....	332

Appendix B. ICSF SMF records..... 339

Record type 82 (52) — ICSF record.....	339
Record environment.....	340
Record mapping.....	340
Subtype 1.....	343
Subtype 7.....	344
Subtype 8.....	345
Subtype 9.....	345
Subtype 13.....	346
Subtype 14.....	346
Subtype 15.....	348
Subtype 16.....	349
Subtype 18.....	350
Subtype 19.....	352
Subtype 20.....	352
Subtype 21.....	354
Subtype 22.....	355
Subtype 23.....	355
Subtype 24.....	355
Subtype 25.....	356
Subtype 26.....	356
Subtype 27.....	357
Subtype 28.....	358
Subtype 29.....	358
Subtype 30.....	359
Subtype 40.....	359

Subtype 41.....	362
Subtype 42.....	365
Subtype 43.....	368
Subtype 44.....	369
Subtype 45.....	371
Subtype 46.....	373
Subtype 47.....	375
Appendix C. CICS-ICSF Attachment Facility.....	377
Installing the CICS-ICSF Attachment Facility.....	377
Steps for installing the CICS-ICSF attachment facility.....	377
Appendix D. Helpful hints for ICSF first time startup.....	381
Checklist for first-time startup of ICSF.....	381
Step 1. Hardware setup.....	381
Step 2. LPAR activation profiles.....	381
Step 3. ICSF setup.....	382
Step 4. TKE setup.....	382
Step 5. ICSF startup.....	383
Step 6. Loading master keys and initializing the CKDS through ICSF panels.....	383
Step 7. Customizing TKE and loading master keys.....	385
Step 8. CICS-ICSF Attachment Facility setup.....	386
Step 9. Complete ICSF initialization.....	386
Commonly encountered ICSF first time setup/initialization messages.....	387
Appendix E. Using AMS REPRO encryption.....	389
Steps for setting up ICSF	389
Appendix F. Systems without Cryptographic features.....	391
Applications and programs.....	391
Callable services.....	391
ICSF setup and initialization.....	392
Secure Sockets Layer (SSL).....	393
TKE workstation.....	393
Appendix G. Accessibility.....	395
Accessibility features.....	395
Consult assistive technologies.....	395
Keyboard navigation of the user interface.....	395
Dotted decimal syntax diagrams.....	395
Notices.....	399
Terms and conditions for product documentation.....	400
IBM Online Privacy Statement.....	401
Policy for unsupported hardware.....	401
Minimum supported hardware.....	402
Trademarks.....	402
Index.....	403

Figures

1. Multiple Crypto coprocessors on a complex.....	106
2. ICSF coprocessor managementpanelsICSF Coprocessor Management.....	107
3. EXPB control block for mainline exits.....	138
4. EXPB control block in the service exits.....	151
5. Example of a service entry and exit.....	187
6. Example of a service stub (1 of 5)service stubexample.....	190
7. Example of a service stub (2 of 5).....	191
8. Example of a service stub (3 of 5).....	192
9. Example of a service stub (4 of 5).....	193
10. Example of a service stub (5 of 5).....	194
11. Example of a Conversion Initial Activity Report.....	211
12. Example of a Conversion Update Activity Report.....	212

Tables

1. z/OS ICSF FMIDs.....	4
2. FMID and Hardware.....	4
3. Exit identifiers and exit invocations.....	38
4. Information about this migration action	55
5. Information about this migration action	56
6. Information about this migration action	57
7. Information about this migration action	58
8. Information about this migration action	59
9. Summary of new and changed ICSF callable services.....	61
10. Cryptographic adapter identification.....	66
11. Mapping of Enterprise PKCS #11 ACPs to firmware levels.....	78
12. Syntax examples.....	88
13. IPCS symbols and format references for the ICSF Control Blocks.....	127
14. DISPLAY GRS command syntax ICSF key data set ENQ resources.....	128
15. ICSF ENF codes.....	130
16. EXPB Control Block format for Mainline Exits.....	138
17. CSFEXIT1 parameters.....	139
18. CSFEXIT2 and CSFEXIT3 parameters.....	140
19. CSFEXIT4 and CSFEXIT5 parameters.....	140
20. Format of the Exit Name table.....	141
21. Services and their ICSF names.....	146
22. Compatibility services and their ICSF names.....	150
23. EXPB Control Block Format for Services.....	151

24. SPB Control Block Format.....	154
25. IXIB control block format.....	158
26. The CKDS Entry Retrieval Exit Parameters.....	161
27. CVXP Control Block Format.....	163
28. RWXP Control Block Format.....	166
29. Parameters received by the Security Service Exit.....	171
30. Parameters received by the Security Key Exit.....	171
31. KGXP Control Block Format.....	174
32. Format of Records in the Override File.....	204
33. Cryptographic Key Data Set Header Record Format.....	215
34. Cryptographic Key Data Set Record Format.....	217
35. Variable-Length Cryptographic Key Data Set Record Format.....	218
36. Public Key Data Set Header Record Format.....	219
37. Public Key Data Set Record Format.....	220
38. Format of the header record of the token data set.....	221
39. Format of the common section of the token and object records.....	222
40. Format of the unique section of the token record.....	223
41. Format of the token object flags.....	224
42. Format of the token certificate object.....	225
43. Format of the token public key object (Version 0).....	226
44. Format of the token public key object (Version 1).....	227
45. Format of the token public key object (Version 2).....	230
46. Format of the token public key object (Version 3).....	232
47. Format of the token private key object (Version 0).....	234
48. Format of the token private key object (Version 1).....	235

49. Format of the token private key object (Version 2).....	238
50. Format of the token private key object (Version 3).....	241
51. Format of the token secret key object (Version 0).....	244
52. Format of the token secret key object (Version 1).....	245
53. Format of the token secret key object (Version 3).....	246
54. Format of the token domain parameters object (Version 1).....	247
55. Format of the token domain parameters object (Version 2).....	248
56. Format of the token data object.....	249
57. Format of the KDSR record fixed data area.....	251
58. Format of KDSR metadata area.....	252
59. Format of KDSR variable-length metadata block.....	252
60. AES internal fixed-length key token format.....	253
61. DES internal fixed-length key token format.....	255
62. Format of DES external fixed-length key tokens.....	257
63. Format of External Key Tokens.....	258
64. External RKX DES key-token format, version X'10'.....	259
65. Format of Null Key Tokens.....	260
66. Variable-length symmetric key token.....	261
67. DESUSECV key-usage fields.....	266
68. HMAC algorithm key-usage fields.....	266
69. AES algorithm MAC key associated data.....	268
70. AES algorithm PINCALC key associated data.....	269
71. AES algorithm PINPROT key associated data.....	270
72. AES algorithm PINPRW key associated data.....	272
73. AES algorithm DKYGENKY key associated data.....	273

74. AES algorithm SECMSG key associated data.....	277
75. AES algorithm KEK key-usage fields.....	277
76. AES algorithm CIPHER key associated data.....	280
77. AES and HMAC algorithm key-management fields.....	282
78. DESUSECV key-management fields.....	286
79. Variable-length symmetric null token.....	286
80. Format of PKA Null Key Tokens.....	287
81. RSA Public Key Token.....	288
82. RSA Private External Key Token Basic Record Format.....	289
83. RSA Private Key Token, 1024-bit Modulus-Exponent external format.....	290
84. RSA Private Key Token, 4096-bit Modulus-Exponent external format.....	291
85. RSA Private Key Token, 4096-bit Chinese Remainder Theorem external format.....	292
86. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external form.....	294
87. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form.....	297
88. RSA Private Internal Key Token Basic Record Format.....	299
89. RSA Private Internal Key Token, 1024-bit X'02' ME Form.....	301
90. RSA Private Internal Key Token, 1024-bit X'06' ME Form.....	302
91. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section internal form.....	304
92. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form.....	306
93. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format.....	308
94. ECC Key Token Format.....	310
95. Associated Data Format for ECC Private Key Token.....	314
96. AESKW Wrapped Payload Format for ECC Private Key Token.....	315
97. Trusted block sections.....	316

98. Trusted block header.....	317
99. Trusted block trusted RSA public-key section (X'11').....	318
100. Trusted block rule section (X'12').....	319
101. Summary of trusted block rule subsection.....	320
102. Transport key variant subsection (X'0001' of trusted block rule section (X'12')).....	321
103. Transport key rule reference subsection (X'0002') of trusted block rule section (X'12').....	322
104. Common export key parameters subsection (X'0003') of trusted block rule section (X'12').....	322
105. Source key rule reference subsection (X'0004' of trusted block rule section (X'12')).....	323
106. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12').....	324
107. Trusted block key label (name) section X'13'.....	326
108. Trusted block information section X'14'.....	326
109. Summary of trusted block information subsections.....	326
110. Protection information subsection (X'0001') of trusted block information section (X'14').....	327
111. Activation and expiration dates subsection (X'0002') of trusted block information section (X'14').....	328
112. Trusted block application-defined data section X'15'.....	328
113. Cryptographic communication vector table.....	329
114. Cryptographic Communication Vector Table Extension.....	331
115. Generic Service Table Block Format.....	331
116. RMF measurements record format.....	332
117. Format of an SMF Type 82 record for subtypes smaller than 40.....	340
118. Format of an SMF Type 82 record for subtypes 40 and higher.....	340
119. SMF record header.....	341
120. ICSF header (for all subtypes 40 or greater).....	341
121. SMF type 82 server user or end user audit section.....	342
122. Tag-Length-Value (TLV) triplet structure (SMF82AUD_TRIPLET).....	342

123. TLV triplet tag values.....	342
124. Tag-Length-Value triplets.....	343
125. Subtype 1 Initialization/Options Refresh.....	344
126. Subtype 7 operational key entry.....	345
127. Subtype 8 Cryptographic key data set refresh.....	345
128. Subtype 9 Dynamic CKDS update.....	346
129. Subtype 13 Dynamic PKDS update.....	346
130. Subtype 14 Cryptographic coprocessor master key entry.....	347
131. Subtype 15 PCI Cryptographic coprocessor retained key create/delete.....	348
132. Subtype 16 PCI Cryptographic Coprocessor TKE.....	349
133. Subtype 16 PCI Cryptographic Coprocessor TKE audit data.....	350
134. Subtype 18 Cryptographic Processor Configuration.....	351
135. Subtype 19 PCI X Cryptographic Coprocessor Timing.....	352
136. Subtype 20 Cryptographic Processor Processing Times.....	353
137. Subtype 21 ICSF Sysplex Group Change.....	354
138. Subtype 22 Trusted Block Create Callable Services.....	355
139. Subtype 23 Token Data Set Update.....	355
140. Subtype 24 Duplicate Tokens Found.....	356
141. Subtype 25 Key Store Policy Key Token Authorization Checking.....	356
142. Subtype 26 Public Key Data Set Refresh.....	357
143. Subtype 27 PKA Key Management Extensions.....	357
144. Subtype 28 High Performance Encrypted Key.....	358
145. Subtype 29 TKE Workstation Audit Record.....	359
146. Subtype 30 Archived and inactive KDS records.....	359
147. Subtype 40 CCA symmetric key lifecycle event.....	360

148. Subtype 41 CCA asymmetric key lifecycle event.....	363
149. Subtype 42 PKCS#11 object lifecycle event.....	365
150. Subtype 43 Regional cryptographic server configuration.....	368
151. Subtype 44 CCA symmetric key usage event.....	369
152. Subtype 45 CCA asymmetric key usage event.....	371
153. Subtype 46 PKCS#11 key usage event.....	373
154. Subtype 47 PKCS#11 no key usage event.....	375

About this information

This information describes how to initialize, customize, operate, and diagnose the z/OS Integrated Cryptographic Service Facility (ICSF). The z/OS Cryptographic Services includes these components:

- z/OS Integrated Cryptographic Service Facility (ICSF)
- 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.

Who should use this information

This information is intended for the system programmer. It describes the tasks that a system programmer might perform:

- Programming installation options, installation-defined callable services, and installation exits
- Creating the data sets that ICSF uses
- Migrating the system from the Cryptographic Unit Support Program (CUSP) and Programmed Cryptographic Facility (PCF) to ICSF
- Migrating to z/OS ICSF
- Starting and stopping ICSF
- Checking event recording
- Planning for security and performance considerations
- Debugging and recovering from problems

Defining and writing installation-defined callable services and installation exit routines is intended to be accomplished primarily by experienced system programmers. This information assumes that the reader has an advanced knowledge of z/OS.

How to use this information

This information is divided into descriptions of these tasks:

- Introducing ICSF
 - [Chapter 1, “Introduction to z/OS ICSF,”](#) on page 1 introduces the cryptographic key data set (CKDS), the public key data set (PKDS), and the token data set (TKDS) and provides basic information about running PCF applications on ICSF and preparing for installation.
- Initializing ICSF
 - [Chapter 2, “Installation, initialization, and customization,”](#) on page 9 describes how to customize SYS1.PARMLIB, create the CKDS, the PKDS, and the TKDS, the installations options data set, the startup procedure, and provide access to the ICSF panels. It also explains how to change the parameters in the installation options data set after the first start and introduces installation exits.
- Migration and coexistence issues

[Chapter 8, “Migration from PCF to z/OS ICSF,”](#) on page 199 describes how to migrate application programs and cryptographic key data set information to z/OS ICSF from the IBM cryptographic products CUSP/PCF.

Chapter 3, “Migration,” on page 53 describes migration to this release of ICSF from previous releases of ICSF.

- Customizing ICSF
 - Chapter 6, “Installation-defined Callable Services,” on page 185 gives information that an experienced system programmer can use to write installation-defined callable services. It also explains how to define these callable services to ICSF and how to write service stubs to access them.
 - Chapter 5, “Installation exits,” on page 131 describes the ICSF installation exits you can use to customize ICSF.
- Operating ICSF
 - Chapter 4, “Operating ICSF,” on page 85 describes how to add and remove cryptographic coprocessors and to start, modify, and stop ICSF and other operating considerations.
 - “ICSF operator commands” on page 90 describes the console commands available for ICSF.
 - “Event recording” on page 114 describes ICSF event recording on the Security Console and SMF.
- Planning ICSF
 - “Security considerations” on page 122 describes methods you can use to protect ICSF resources.
- Diagnosing ICSF
 - “Debugging aids” on page 125 describes the use of component trace and Interactive Problem Control System (IPCS) to debug ICSF.
 - Appendix A, “Diagnosis reference information,” on page 215 maps the cryptographic key data set and the cryptographic communication vector tables as reference information for use in debugging. This appendix also maps CCA key tokens (DES, AES, RSA, and ECC) and trusted blocks.
 - Appendix B, “ICSF SMF records,” on page 339 describes SMF Record type 82, which is used to record information about the events and operations of ICSF. Record type 82 is written to the SMF data set at the completion of certain cryptographic functions.
 - Appendix C, “CICS-ICSF Attachment Facility,” on page 377 defines steps to install the CICS-ICSF Attachment Facility.
 - Appendix D, “Helpful hints for ICSF first time startup,” on page 381 defines helpful hints and that you may encounter when starting ICSF for the first time.
 - Appendix E, “Using AMS REPRO encryption,” on page 389 provides information on using IDCAMS REPRO ENCRYPT and DECRYPT options with ICSF.
 - Appendix F, “Systems without Cryptographic features,” on page 391 describes processing and functionality support for this environment.
 - Appendix G, “Accessibility,” on page 395 contains information on accessibility features in z/OS.
 - “Notices” on page 399 contains information on 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*](#)
- [*z/OS Cryptographic Services ICSF TKE Workstation User's Guide*](#)

This publication also refers to these publications:

- *IBM ES/3090 Processor Complex Recovery Guide*
- *[z/OS Planning for Installation](#)*
- *[z/OS Security Server RACF Auditor's Guide](#)*
- *[z/OS Security Server RACF Command Language Reference](#)*
- *[z/OS Security Server RACF Security Administrator's Guide](#)*
- *[z/OS Security Server RACF Macros and Interfaces](#)*
- *[z/OS Security Server RACF System Programmer's Guide](#)*
- *[z/OS MVS IPCS User's Guide](#)*
- *[z/OS MVS System Codes](#)*
- *[z/OS MVS System Management Facilities \(SMF\)](#)*
- *[z/OS MVS Programming: Extended Addressability Guide](#)*
- *[z/OS MVS Initialization and Tuning Guide](#)*
- *[z/OS MVS Initialization and Tuning Reference](#)*
- *[z/OS DFSMS Access Method Services Commands](#)*
- *[z/OS DFSMS Using Data Sets](#)*
- *IBM Distributed Key Management System, Installation and Customization Guide*
- *OS/VS1 and OS/VS2 MVS Cryptographic Unit Support: Installation Manual*
- *OS/VS1 and OS/VS2 MVS Programmed Cryptographic Facility*
- *IBM 4765 Specification Sheet*
- *IBM CCA Basic Services Reference and Guide for the IBM 4765 PCIe and IBM 4765 PCI-X Cryptographic Coprocessors*
- *IBM 4765 Warranty Information Flyer*
- *IBM 4765 PCIe Cryptographic Coprocessor Installation Manual*

IBM Crypto Education

The [IBM Crypto Education \(www.ibm.com/developerworks/community/groups/community/crypto\)](http://www.ibm.com/developerworks/community/groups/community/crypto) community provides detailed explanations and samples pertaining to IBM cryptographic technology.

How to send your comments to IBM

We invite you to submit comments about the z/OS® product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead [“If you have a technical problem”](#) on page xxiii.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function

If your comment or question is about z/OS itself, submit a request through the [IBM RFE Community](http://www.ibm.com/developerworks/rfe/) (www.ibm.com/developerworks/rfe/).

Feedback on IBM® Knowledge Center function

If your comment or question is about the IBM Knowledge Center functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Knowledge Center Support at ibmkc@us.ibm.com.

Feedback on the z/OS product documentation and content

If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS ICSF System Programmer's Guide, SC14-7507-06
- The section title of the specific information to which your comment relates
- The text of your comment.

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

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

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the [IBM Support Portal](http://support.ibm.com) (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical 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. See [“z/OS ICSF FMIDs” on page 3](#) for more information on z/OS ICSF FMIDs and their relationships to z/OS releases.

ICSF publications can be obtained from:

- The [Resource Link home page \(www.ibm.com/servers/resourcelink\)](http://www.ibm.com/servers/resourcelink). (Select Publications and then select the release that you are interested in under ICSF Publications by FMID.)
- IBM [z/OS downloads \(www.ibm.com/systems/z/os/zos/downloads\)](http://www.ibm.com/systems/z/os/zos/downloads) for Cryptographic Support.

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

Changes made in Cryptographic Support for z/OS V2R1 - z/OS V2R2 (FMID HCR77C0)

This document contains information previously presented in *z/OS ICSF System Programmer's Guide*, SC14-7507-05.

This document is for ICSF FMID HCR77C0. This release of ICSF runs on z/OS V2R1 and z/OS V2R2 and only on zSeries hardware.

The most recent updates are listed at the top of each section.

New

- [“Starting ICSF during IPL-time” on page 86](#) is new (APAR OA55378).
- New CSF.SCSFSTUB dataset.
- New SMF subtypes: 40, 41, 42, 44, 45, 46, and 47.
- New system abend codes are summarized in [“System abend codes” on page 72](#).
- [“ENF signals” on page 129](#) is new.

Changed

- [“Subtype 42” on page 365](#) (APAR OA55958).
- [“Subtype 46” on page 373](#) (APAR OA55958).
- [“PKCS#11 object lifecycle event” on page 365](#) has been updated (APAR OA54346).
- [“PKCS#11 key usage event” on page 373](#) has been updated (APAR OA54346).
- [“Starting and stopping ICSF” on page 85](#) has been updated.
- [“High Performance Encrypted Key \(Subtype 28\)” on page 120](#) has been updated.
- [“Subtype 28” on page 358](#) has been updated.
- SMF subtype 1 is written whenever ICSF is started or the options refresh is performed.
- [“The Cryptographic Key Data Set \(CKDS\)” on page 5](#) has been updated.
- [“The Public Key Data Set \(PKDS\)” on page 6](#) has been updated.
- [“Steps to create the CKDS” on page 12](#) has been updated.

- [“Steps to create the PKDS” on page 15](#) has been updated.
- [“Steps to create the TKDS” on page 18](#) has been updated.
- [“Parameters in the installation options data set” on page 30](#) has been updated.
- [“Callable services” on page 61](#) has been updated.
- [“CICS attachment facility” on page 71](#) has been updated.
- [“SETICSF” on page 95](#) has been updated.
- [“Installing the exits” on page 145](#) has been updated.
- [“Installing the exits” on page 169](#) has been updated.

Deleted

No content was removed from this information.

Changes made in Cryptographic Support for z/OS V1R13 - z/OS V2R2 (FMID HCR77B1)

This document contains information previously presented in *z/OS ICSF System Programmer's Guide*, SC14-7507-03.

This document is for ICSF FMID HCR77B1. This release of ICSF runs on z/OS V1R13, z/OS V2R1, and z/OS V2R2 and only on zSeries hardware.

The most recent updates are listed at the top of each section.

New

- Updated to include information about IBM z13s.
- Added information about the Encrypted PIN Translate Enhanced and Key Encryption Translate services.
- Added information about regional cryptographic servers.
- New system abend code is summarized in [“System abend codes” on page 72](#).
- [“Server hardware” on page 2](#) was updated with information on regional cryptographic servers.
- New system abend codes are summarized in [“System abend codes” on page 72](#).
- [“Command syntax notation” on page 87](#) is new.
- [“ICSF operator commands” on page 90](#) is new and includes the [“Display ICSF” on page 90](#) and the [“SETICSF” on page 95](#) commands.
- [“Adding and removing regional cryptographic servers” on page 108](#) is new.
- [“Configuring ICSF to use TCP/IP for communications with regional cryptographic servers” on page 109](#) is new.
- [“Displaying cryptographic coprocessor status using the DISPLAY ICSF operator command” on page 112](#) is new.
- [“Changing regional cryptographic server status using the SETICSF operator command” on page 112](#) is new.
- [“ICSF header \(for all subtypes 40 or greater\)” on page 341](#) is new.
- [“Regional cryptographic server configuration \(Subtype 43\)” on page 121](#) is new. Details about SMF record subtype 43 can be found in [“Subtype 43” on page 368](#).

Changed

- Terminology changed from open cryptographic services to regional cryptographic services.
- Updates to “Parameters in the installation options data set” on page 30:
 - Six new EMV services added.
 - New MASTERKCVLEN and REMOTEDEVICE keywords added.
 - The deprecation of the HDRDATE keyword. If this option is specified, it will be tolerated, but is no longer supported.
- “Callable services” on page 61 was updated with information on the six new EMV services and updates to Key Generate, PKA Decrypt, and PKA Encrypt.
- “Installing the exits” on page 145 was updated with information on the six new EMV services.
- “Secondary parameter block” on page 154 was updated.
- “Format of the header record of the token data set” on page 221 was updated.

Deleted

No content was removed from this information.

Changes made in Enhanced Cryptographic Support for z/OS V1R13 - z/OS V2R1 (FMID HCR77B0)

This document contains information previously presented in *z/OS ICSF System Programmer's Guide*, SC14-7507-01.

This document is for ICSF FMID HCR77B0. This release of ICSF runs on z/OS V1R13 and z/OS V2R1 and only on zSeries hardware.

New

- “Server hardware” on page 2 was updated to include new information about IBM z13®.
- The following new options have been added in “Parameters in the installation options data set” on page 30:
 - KEYARCHMSG (YES or NO)
 - RNGCACHE(YES or NO)
- New system abend codes summarized in “System abend codes” on page 72.
- New subtype for ICSF SMF record type 82 (52) in “Key Store Policy Archived and Inactive Checking (Subtype 30)” on page 121.
- New services installation exit, “CSF_SERVICE_EXIT - ICSF callable services exit” on page 157.
- New CCVTRLVL field added to “The Cryptographic Communication Vector Table (CCVT)” on page 329.

Changed

- DOMAIN(n) is updated for 256 domain support on IBM z13 in “Parameters in the installation options data set” on page 30.
- “Steps for initializing ICSF” on page 28 has been updated.
- Chapter 3, “Migration,” on page 53 has been updated for FMID HCR77B0.
- “Key store policy” on page 66 has been updated for KGUP, key material archiving, and key material validity.
- “ICSF key data sets” on page 67 has been updated with information about record metadata.

- “Migrating to 24-byte DES master key” on page 70 has been updated.
- “CICS attachment facility” on page 71 has been updated for FMID HCR77B0.
- “Migrating a CKDS and PKDS between a CCF system and a non-CCF system” on page 79 has been updated.
- “Using different configurations” on page 105 is updated for 256 domain support.
- “System Management Facilities (SMF) recording” on page 114 has multiple updates:
 - “Dynamic CKDS Update (Subtype 9)” on page 117
 - “Dynamic PKDS Update (Subtype 13)” on page 117
 - “Key Store Policy Key Token Authorization Checking (Subtype 25)” on page 119
 - “Key Store Policy Archived and Inactive Checking (Subtype 30)” on page 121
 - “Subtype 1” on page 343
 - “Operational Key Part Entry (Subtype 7)” on page 116
 - “Subtype 14” on page 346
 - “Subtype 15” on page 348
 - “Subtype 16” on page 349
 - “Subtype 18” on page 350
 - “Subtype 25” on page 356
 - “Subtype 20” on page 352
- “Common record format (KDSR)” on page 250 has been updated.

Deleted

No content was removed from this information.

Changes made in Cryptographic Support for z/OS V1R13 - z/OS V2R1 (FMID HCR77A1)

This document contains information previously presented in *z/OS ICSF System Programmer's Guide*, SA22-7520-17.

This document is for ICSF FMID HCR77A1. This release of ICSF runs on z/OS V1R13 and z/OS V2R1 and only on zSeries hardware.

The most recent updates are listed at the top of each section.

New information

- Support for the German Banking Industry Committee (Deutsche Kreditwirtschaft (DK)) PIN methods with the application of the PTFs for APARs OA42246, OA43906, and OA44444:
 - Table 3 on page 38 in Chapter 2, “Installation, initialization, and customization,” on page 9 has been updated with new information.
 - Table 9 on page 61 and “CICS attachment facility” on page 71 in Chapter 3, “Migration,” on page 53 has been updated with new information.
 - “Variable-length symmetric key token” on page 261 and “RMF measurements table” on page 332 in Appendix A, “Diagnosis reference information,” on page 215 has been updated with new information.
- Table 3 on page 38 in Chapter 2, “Installation, initialization, and customization,” on page 9 has been updated with new information.

- [Table 9 on page 61 in Chapter 3, “Migration,” on page 53](#) has been updated with new information.
- The 'online' coprocessor status is no longer supported. See [Chapter 3, “Migration,” on page 53](#) for details.
- New reason codes were added to [“System abend codes” on page 72 in Chapter 3, “Migration,” on page 53](#).
- [Table 21 on page 146 in Chapter 5, “Installation exits,” on page 131](#) has been updated with new information.
- The callable services section of [Appendix F, “Systems without Cryptographic features,” on page 391](#) has had ICSF Query Facility2 (CSFIQF2) added to the list for FMID HCR77A1.

Changed information

- [“ICSF key data sets” on page 67 and Chapter 7, “Converting a CKDS from fixed length to variable length record format,” on page 195](#) have multiple updates.
- [“Record type 82 \(52\) — ICSF record” on page 339 in Appendix B, “ICSF SMF records,” on page 339](#) has multiple updates.
- There are multiple changes in this document for the simplification of cryptographic coprocessor configuration. Affected sections are:
 - [“Steps to start ICSF for the first time” on page 28 in Chapter 2, “Installation, initialization, and customization,” on page 9](#).
 - [“SMF records” on page 76](#).
- The [Chapter 2, “Installation, initialization, and customization,” on page 9](#) section, “Parameters in the installation options data set”, has been updated with additional information about the SSM option.
- [Chapter 3, “Migration,” on page 53](#) has been updated for FMID HCR77A1.
- The [Chapter 3, “Migration,” on page 53](#) section [“Migrating from earlier software releases” on page 54](#) overview description has been updated for HCR77A1.
- The [Chapter 4, “Operating ICSF,” on page 85](#) section [“Adding and removing cryptographic coprocessors” on page 106](#) has been completely rewritten.
- The [Chapter 4, “Operating ICSF,” on page 85](#) section [“Component trace” on page 125](#) has been completely rewritten.
- [“The Cryptographic Communication Vector Table \(CCVT\)” on page 329 in Appendix A, “Diagnosis reference information,” on page 215, section “Data areas” on page 329](#) has been updated with new information for field name CCVT1FLG.
- [Appendix B, “ICSF SMF records,” on page 339](#) section [“Record type 82 \(52\) — ICSF record” on page 339](#) has multiple updates.
- [Appendix D, “Helpful hints for ICSF first time startup,” on page 381](#) has been completely rewritten.
- [Appendix F, “Systems without Cryptographic features,” on page 391](#) has been completely rewritten.

Deleted information

- In z/OS V2R1, support for z900/z800 and z990/z890 is removed. HCR77A0 ships in the base of z/OS V2R1 and will continue to support these older hardware environments. HCR77A1 removes support for the hardware present on z900/z800 (CCF and PCICC). z990/z890 (PCIXCC and PCICA) will continue to be supported because HCR77A1 will still run on z/OS V1R13.

Additionally, a number of software functions that rely on the presence of CCF and PCICC features or do not serve a purpose without these features is removed, including the Managing Keys According to the ANSI X9.17 Standard chapter and the Using ICSF with BSAFE appendix.

- Support for the following services has been removed in HCR77A1:
 - ANSI X9.17 EDC Generate (CSNAEGN and CSNGEGN)

- ANSI X9.17 Key Export (CSNAKEX and CSNGKEX)
- ANSI X9.17 Key Import (CSNAKIM and CSNGKIM)
- ANSI X9.17 Key Translate (CSNAKTR and CSNGKTR)
- ANSI X9.17 Transport Key Partial Notarize (CSNATKN and CSNGTKN)
- Ciphertext Translate (CSNBCTT or CSNBCTT1 and CSNECTT or CSNECTT1)
- Transform CDMF Key (CSNBTK and CSNETCK)
- User Derived Key (CSFUDK and CSFUDK6)
- PKSC Interface Callable Service (CSFPKSC)
- The following reason codes are no longer issued:
 - 9 (9)
 - 13 (19)
 - 3C (60)
 - 41 (65)
 - 42 (66)
 - 48 (72)
 - 49 (73)
 - 4A (74)
 - 5A (90)
 - 73 (115)
 - 74 (116)
 - 8C (140)
 - 8D (141)
 - 8E (142)
 - 90 (144)
 - 92 (146)
 - 94 (148)
 - 98 (152)
 - A3 (163)
 - A4 (164)
 - A6 (166)
 - A7 (167)
 - A8 (168)
 - A9 (169)
 - AC (172)
 - AD (173)
 - B6 (182)
 - 103 (259)
 - 104 (260)
 - 107 (263)
 - 108 (264)
 - 109 (265)
 - 10A (266)
 - 197 (407)

- 199 (409)
- 205 (517)
- 207 (519)
- 301 (769)
- 400 (1024)
- 401 (1025)
- 402 (1026)
- 407 (1031)
- 408 (1032)
- 409 (1033)
- 40E (1038)
- 40F (1039)
- 410 (1040)
- 41B (1051)
- 41D (1053)
- 423 (1059)
- 42B (1067)
- 42C (1068)
- 42D (1069)
- 432 (1074)
- 433 (1075)
- 449 (1097)
- 44A (1098)
- 44B (1099)
- 44D (1101)
- 44E (1102)
- 46B (1131)
- 470 (1136)
- 471 (1137)

Chapter 1. Introduction to z/OS ICSF

ICSF is a software element of z/OS. ICSF works with the hardware cryptographic features and the Security Server (RACF element) to provide secure, high-speed cryptographic services in the z/OS environment. ICSF provides the application programming interfaces by which applications request the cryptographic services. ICSF is also the means by which the secure cryptographic features are loaded with master key values, allowing the hardware features to be used by applications. The cryptographic feature is secure, high-speed hardware that performs the actual cryptographic functions. Your processor hardware determines the cryptographic feature available to your applications.

Features

Cryptographic hardware features

This topic describes the cryptographic hardware features available. Information on adding and removing cryptographic coprocessors can be found in *z/OS Cryptographic Services ICSF Administrator's Guide*.

Crypto Express5 adapter (CEX5C, CEX5P, or CEX5A)

The Crypto Express5 adapter is an asynchronous cryptographic coprocessor or accelerator. The adapter contains one cryptographic engine that can be configured as a coprocessor (CEX5C for CCA and CEX5P for PKCS #11) or as an accelerator (CEX5A). It is available on IBM z13 and IBM z13s.

Crypto Express4 adapter (CEX4C, CEX4P, or CEX4A)

The Crypto Express4 adapter is an asynchronous cryptographic coprocessor or accelerator. The adapter may be configured as a CCA coprocessor (CEX4C), an Enterprise PKCS #11 coprocessor (CEX4P), or as an accelerator (CEX4A). It is available on IBM zEnterprise EC12 and IBM zEnterprise BC12.

Crypto Express3 adapter (CEX3C or CEX3A)

The Crypto Express3 adapter is an asynchronous cryptographic coprocessor or accelerator. The adapter contains two cryptographic engines that can be independently configured as a coprocessor (CEX3C) or as an accelerator (CEX3A). It is available on the IBM System z10 Enterprise Class, IBM System z10 Business Class, IBM zEnterprise 196, IBM zEnterprise 114, IBM zEnterprise EC12, and the IBM zEnterprise BC12.

Crypto Express2 adapter (CEX2C or CEX2A)

The Crypto Express2 adapter is an asynchronous cryptographic coprocessor or accelerator. The adapter contains two cryptographic engines that can be independently configured as a coprocessor (CEX2C) or as an accelerator (CEX2A). It is available on the IBM System z9 Enterprise Class, IBM System z9 Business Class, IBM System z10 Enterprise Class, and IBM System z10 Business Class.

CP Assist for Cryptographic Functions (CPACF)

CPACF is a set of cryptographic instructions available on all CPs. Use of the CPACF instructions provides improved performance. The SHA-1 algorithm is always available. Additionally, SHA-224 and SHA-256 algorithms are available on the z9 EC / z9 BC and newer systems. Additionally, SHA-384 and SHA-512 algorithms are available on z10 EC / z10 BC and newer systems.

CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement, feature 3863, provides for clear key DES and TDES instructions. On the z9 EC / z9 BC and later systems, this feature includes clear key

AES for 128-bit keys. On z10 EC / z10 BC and later systems, this feature also includes clear key AES for 192-bit and 256-bit keys.

Server hardware

This topic describes the servers on which the cryptographic hardware features are available.

Regional cryptographic server

Regional cryptographic servers are network-attached, stand-alone devices or dedicated Linux LPARs that perform geography-specific cryptography. Later generations of these servers add international algorithm support. These servers are secure key hardware security modules (HSMs) that operate similar to IBM's PKCS #11 secure coprocessors (CEXnP). They are marketed and serviced by third-party vendors. Currently, the only geography-specific cryptography that is supported by these devices is the Chinese SMx family of algorithms. Secure keys are stored in the TKDS, protected by the Regional Cryptography Server Master Key (RCS-MK).

The network-attached, stand-alone devices require no particular zSeries hardware, but do require communicating with z/OS V1R13 or later and ICSF FMID HCR77B1 or later. ICSF communicates with these devices using TCP/IP, with optional TLS protection. The Linux LPARs require IBM z13 or later hardware. ICSF communicates with the Linux LPARs using TCP/IP, with TLS protection required.

Once configured and online, ICSF makes the algorithms that are offered by these devices available as PKCS #11 vendor-defined extensions.

- For information on configuring these devices, see [*z/OS Cryptographic Services ICSF System Programmer's Guide*](#).
- For information on the algorithms offered, see [*z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*](#) and [*z/OS Cryptographic Services ICSF Application Programmer's Guide*](#).

IBM z13 and IBM z13s

The IBM z13 and IBM z13s provide constraint relief and addresses various customer demands. It has several cryptographic features.

- CP Assist for Cryptographic Functions is implemented on every processor. SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512 secure hashing is directly available to application programs.
- Feature code 3863, CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement - enables clear key DES and TDES instructions on all CPs. AES 128-bit, AES 192-bit and AES 256-bit support is also available.
- Feature code 0890, Crypto Express5 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM z13s can support a maximum of 16 adapters. Each feature code has one hardware adapter which can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.

IBM zEnterprise EC12 (zEC12) and IBM zEnterprise BC12 (zBC12)

The IBM zEnterprise EC12 and IBM zEnterprise BC12 provide constraint relief and addresses various customer demands. It has several cryptographic features.

- CP Assist for Cryptographic Functions is implemented on every processor. SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512 secure hashing is directly available to application programs.
- Feature code 3863, CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement - enables clear key DES and TDES instructions on all CPs. AES 128-bit, AES 192-bit and AES 256-bit support is also available.
- Feature code 0864, Crypto Express3 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM zEnterprise EC12 can support a maximum of 8 adapters. Each feature code has two coprocessors/accelerators.
- Feature code 0865, Crypto Express4 adapter - optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM zEnterprise EC12 can support a maximum of 16

adapters. Each feature code has one hardware adapter which can be configured as a CCA coprocessor, a PKCS #11 coprocessor, or an accelerator.

IBM zEnterprise 196 (z196) and IBM zEnterprise 114 (z114)

The IBM zEnterprise 196 and IBM zEnterprise 114 provide constraint relief and addresses various customer demands. It has several cryptographic features.

- CP Assist for Cryptographic Functions is implemented on every processor. SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512 secure hashing is directly available to application programs.
- Feature code 3863, CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement – enables clear key DES and TDES instructions on all CPs. AES 128-bit, AES 192-bit and AES 256-bit support is also available.
- Feature code 0864, Crypto Express3 adapter – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM zEnterprise 196 and IBM zEnterprise 114 can support a maximum of 8 adapters. Each feature code has two coprocessors/accelerators.

IBM System z10 Enterprise Class (z10EC) and IBM System z10 Business Class (z10 BC)

The IBM System z10 Enterprise Class and IBM System z10 Business Class provide constraint relief and addresses various customer demands. It has several cryptographic features.

- CP Assist for Cryptographic Functions is implemented on every processor. SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512 secure hashing is directly available to application programs.
- Feature code 3863, CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement – enables clear key DES and TDES instructions on all CPs. AES 128-bit, AES 192-bit and AES 256-bit support is also available.
- Feature code 0863, Crypto Express2 adapter – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The z10 EC and z10 BC can support a maximum of 8 adapters. Each feature code has two coprocessors/accelerators.
- Feature code 0864, Crypto Express3 adapter – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The z10 EC and z10 BC can support a maximum of 8 adapters. Each feature code has two coprocessors/accelerators.

IBM System z9 Enterprise Class (z9 EC) and IBM System z9 Business Class (z9 BC)

The IBM System z9 Enterprise Class (z9 EC) and IBM System z9 BC provide constraint relief and addresses various customer demands. It has several cryptographic features.

- CP Assist for Cryptographic Functions is implemented on every processor. SHA-1, SHA-224 and SHA-256 secure hashing is directly available to application programs.
- Feature code 3863, CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement – enables clear key DES and TDES instructions on all CPs. In addition, ICSF supports hardware implementation of AES 128-bit keys and software implementation of AES 192-bit and AES 256-bit key lengths.
- Feature code 0863, Crypto Express2 adapter – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM System z9 BC can support a maximum of 8 adapters. Each feature code has two coprocessors/accelerators.

z/OS ICSF FMIDs

These tables explain the relationships of z/OS releases, ICSF FMIDs and servers.

Table 1. z/OS ICSF FMIDs		
z/OS	ICSF FMID	Web deliverable name
V2R1	HCR77A0	Cryptographic Support for z/OS V1R12-R13.
	HCR77A1	Cryptographic Support for z/OS V1R13 - z/OS V2R1.
	HCR77B0	Enhanced Cryptographic Support for z/OS V1R13 - z/OS V2R1.
	HCR77B1	Cryptographic Support for z/OS V1R13 - z/OS V2R2.
	HCR77C0	Cryptographic Support for z/OS V2R1 - z/OS V2R2.
V2R2	HCR77B0	Enhanced Cryptographic Support for z/OS V1R13 - z/OS V2R1.
	HCR77B1	Cryptographic Support for z/OS V1R13 - z/OS V2R2.
	HCR77C0	Cryptographic Support for z/OS V2R1 - z/OS V2R2.
V2R3	HCR77C0	Cryptographic Support for z/OS V2R1 - z/OS V2R2.

Refer to this chart to determine what release is associated with each ICSF FMID and what server it will run on.

Table 2. FMID and Hardware		
ICSF FMID	Applicable z/OS Releases	Servers where FMID will run
HCR77A0 (Base of z/OS 2.1)	1.12, 1.13, and 2.1	z800, z900, z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, z114, z196, zBC12, zEC12, z13, and z13s.
HCR77A1	1.13 and 2.1	z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, z114, z196, zBC12, zEC12, z13, and z13s.
HCR77B0 (Base of z/OS 2.2)	1.13, 2.1, and 2.2	z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, z114, z196, zBC12, zEC12, z13, and z13s.
HCR77B1	1.13, 2.1, and 2.2	z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, z114, z196, zBC12, zEC12, z13, and z13s.
HCR77C0 (Base of z/OS 2.3)	2.1, 2.2, and 2.3	z9 EC, z9 BC, z10 EC, z10 BC, z114, z196, zBC12, zEC12, z13, and z13s.

ICSF features

ICSF protects data from unauthorized disclosure or modification. It protects data that is stored within a system, stored in a file on magnetic tape off a system, and sent between systems. It can also be used to authenticate identities of senders and receivers and to ensure the integrity of messages transmitted over a network. It uses cryptography to accomplish these functions.

Cryptography enciphers data, using an algorithm and a cryptographic key, so the data is in an unintelligible form. Deciphering data involves reproducing the intelligible data from the unintelligible data. To encipher and decipher data, ICSF uses either the U.S. National Institute of Science and Technology Data Encryption Standard (DES) algorithm, Advanced Encryption Standard (AES), Elliptic Curve Cryptography (ECC) or the RSA algorithm.

ICSF supports several Public Key Algorithms (PKA), which do not require exchanging a secret key. You can use these algorithms to exchange AES or DES secret keys securely and to compute digital signatures for authenticating messages and users. For digital signatures, you use a pair of keys: a private (secret) key to

sign a message and a corresponding public key to verify the signature. ICSF supports the RSA, and ECC algorithms.

You can call an ICSF callable service from an application program to perform a cryptographic function. ICSF uses keys in cryptographic functions to:

- Protect data
- Protect other keys
- Verify that messages were not altered between sender and receiver
- Generate, protect, and verify personal identification numbers (PINs)
- Distribute AES and DES keys
- Generate and verify digital signatures

You use ICSF callable services and programs to generate, maintain, and manage keys that are used in the cryptographic functions. A unique key performs each type of cryptographic function on ICSF. All secret keys are encrypted under another key, a master key or a wrapping key. There are up to four CCA master keys depending on your cryptographic coprocessors: DES, RSA, AES and ECC. All master keys are physically secure within the boundary of the cryptographic coprocessors. Operational secret keys are encrypted under their respective master key.

The P11 master key is used to protect secure PKCS #11 keys. Secure PKCS #11 keys are supported only on features configured for PKCS #11. The P11 master key is physically secure within the boundary of the coprocessors.

The Cryptographic Key Data Set (CKDS)

Cryptographic keys that are protected under the DES or AES master key are stored in a VSAM data set that is called the cryptographic key data set (CKDS). ICSF provides sample CKDS allocation jobs (members CSFCKDS, CSFCKD2, and CSFCKD3) in SYS1.SAMPLIB. An installation is not required to define a CKDS. However, when a CKDS is not defined, secure CCA symmetric key functions are not available and ICSF cannot be used to manage CCA symmetric key tokens. The CKDS contains individual entries for each key that is added to it. You can store all types of operational symmetric keys in the CKDS. Each record in the data set contains the key value encrypted under the master key and other information about the key. ICSF maintains two copies of the CKDS: a disk copy and an in-storage copy.

Callable services use the in-storage copy of the CKDS to perform cryptographic functions. For information on managing and sharing the CKDS in a sysplex environment, see [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

Applications can use the dynamic CKDS update callable services to create, write, read, and delete CKDS records.

There are three formats of the CKDS:

- A fixed-length record format with LRECL=252 (supported by all releases of ICSF). Sample is CSFCKDS.
- A variable-length record format with LRECL=1024 (supported by HCR7780 and later releases). Sample is CSFCKD2.
- The common record format (KDSR) that is common to all key data sets with LRECL=2048 (supported by ICSF FMID HCR77A1 and later). Sample is CSFCKD3.

You should use the most current format, the common record format (KDSR), for all your key data sets because KDSR format supports additional function to manage cryptographic keys. For information on converting your existing CKDS to KDSR format, see [“Migrating to the common record format \(KDSR\) key data set” on page 69](#).

If variable-length AES and HMAC keys are to be stored in the CKDS, you must use the variable-length or KDSR format of the CKDS. These formats can store all symmetric key tokens, both fixed-length and variable-length tokens. The KDSR format allows ICSF to track key usage if so configured.

The Public Key Data Set (PKDS)

RSA and ECC public and private keys and trusted blocks can be stored in a VSAM data set that is called the public key data set (PKDS). ICSF provides sample PKDS allocation jobs (member CSFPKDS) in SYS1.SAMPLIB. An installation is not required to define a PKDS. However, when a PKDS is not defined, secure CCA asymmetric key functions are not available and ICSF cannot be used to manage CCA asymmetric key tokens. The PKDS contains individual entries for each key that is added to it. You can store public key tokens, both external and internal private key tokens, and trusted blocks in the PKDS. ICSF maintains two copies of the PKDS: a disk copy and an in-storage copy.

Callable services use the in-storage copy of the PKDS to perform cryptographic functions. For information on managing and sharing the PKDS in a sysplex environment, see [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

Applications can use the dynamic PKDS update callable services to create, write, read, and delete PKDS records.

There are two formats of the PKDS: the PKDS record format (supported by all releases of ICSF) and the common record format (KDSR) that is common to all key data sets (supported by ICSF FMID HCR77A1 and later). The KDSR format allows ICSF to track key usage if so configured.

You should use the most current format, the common record format (KDSR), for all your key data sets because KDSR format supports additional function to manage cryptographic keys. For information on converting your existing PKDS to KDSR format, see [“Migrating to the common record format \(KDSR\) key data set” on page 69](#).

The Token Data Set (TKDS)

PKCS #11 tokens and objects are stored in a VSAM data set called the token data set (TKDS). ICSF provides sample TKDS allocation jobs (members CSFTKDS and CSFTKD2) in SYS1.SAMPLIB. The TKDS contains individual entries for each token and object that is added to it. ICSF maintains two copies of the TKDS: a disk copy and an in-storage copy. Only token objects are stored in the TKDS. Session objects (which are not persistent) are stored in memory only.

The TKDS must be a key-sequenced data set with spanned variable length records and must be allocated on a permanently resident volume. For information on managing and sharing the TKDS in a sysplex environment, see [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

The TKDS is optional for installations that do not use PKCS #11 services or for installations that use only clear session (non-persistent) PKCS #11 keys.

There are two formats of the TKDS: the TKDS record format (supported by all releases of ICSF), and the common record format (KDSR) that is common to all KDS types (supported by ICSF FMID HCR77A1 and later). KDSR allows ICSF to track key usage if so configured.

You should use the most current format, the common record format (KDSR), for all your key data sets because KDSR format supports additional function to manage cryptographic keys. For information on converting your existing TKDS to KDSR format, see [“Migrating to the common record format \(KDSR\) key data set” on page 69](#).

Additional background information

These topics provide some additional background information about using ICSF with other products, such as the Programmed Cryptographic Facility (PCF).

Running PCF applications on z/OS ICSF

If your installation uses PCF, you can run PCF applications on ICSF. You can use an installation option to specify whether a PCF application runs on ICSF. If you are migrating from PCF, ICSF provides a conversion program that converts a PCF CKDS to ICSF format.

You can use your own installation services and exits to customize ICSF. You can write, define, and call your own installation-defined callable service. You can also write and define exits that ICSF calls during the processing of:

- ICSF mainline
- A callable service
- The PCF CKDS conversion program
- The key generator utility program
- CKDS access

For example, most callable services in ICSF call an exit before and after processing. Such an exit can alter return codes in a service.

ICSF System SVC 143

SVC 143 (0A8F) is an ICSF system SVC that is used by CUSP and PCF macros (GENKEY, RETKEY, CIPHER, and EMK) for SVC entry into ICSF. The SVC allows you to run a CUSP or PCF application on ICSF. See [“Running PCF and z/OS ICSF on the same system” on page 199](#) for more information about running CUSP and PCF applications on ICSF.

SVC 143 is a type 4 SVC and does not get a lock. The General Trace Facility data is:

R15 and R0

No applicable data.

R1

Address of the parameter list. The macro that is called determines the parameter list.

Using RMF and SMF to monitor z/OS ICSF events

You can run ICSF in different configurations and use installation options to affect ICSF performance. While ICSF is running, you can use the Resource Management Facilities (RMF) and System Management Facilities (SMF) to monitor certain events. For example, ICSF records information in the SMF data set when ICSF changes the status of a cryptographic processor or when you enter or change the master key. ICSF also sends information and diagnostic messages to data sets and consoles.

With the availability of cryptographic hardware on an LPAR basis, RMF provides performance monitoring in the Postprocessor Crypto Hardware Activity report. This report is based on SMF record type 70, subtype 2. The Monitor I gathering options on the REPORTS control statement are CRYPTO and NOCRYPTO. Specify CRYPTO to measure cryptographic hardware activity and NOCRYPTO to suppress the gathering. In addition, overview criteria is shown for the Postprocessor in the Postprocessor Workload Activity Report - Goal Mode (WLMGL) report. Refer to [z/OS RMF Programmer's Guide](#), [z/OS RMF User's Guide](#), and [z/OS RMF Report Analysis](#) for additional information.

ICSF also supports enabling RMF to provide performance measurements on ICSF services (Decipher, Digital Signature Generate, Digital Signature Verify, Encipher, FPE Decipher, FPE Encipher, FPE Translate, MAC Generate, MAC Verify, One Way Hash, PIN Translate, and PIN Verify). These measurements are of the PCIXCCs or Crypto Express coprocessors.

For diagnosis monitoring, use Interactive Problem Control System (IPCS) to access the trace buffer and to format control blocks.

Controlling access to ICSF

For security, you should control access to ICSF resources and services. Use a security product like the Security Server (RACF) to protect cryptographic programs, keys, and services. You should also change the value of your master keys periodically.

Steps prior to starting installation

You use either ServerPac or CBPDO to install ICSF as part of the z/OS installation process.

When beginning installation:

1. Refer to [*z/OS Planning for Installation*](#) for installation planning information.
2. Check with your IBM center or search the IBM problem database to find any pertinent Preventative Service Planning (PSP). There may also be HOLDDATA and PSP information for ICSF on the tape.
3. Make sure that you have all needed programs and their corequisites:
 - If you use the Security Server (RACF) and want access control and auditing services for ICSF, you need the Security Server (RACF), an optional feature of z/OS.
 - If you are a Resource Measurement Facility (RMF) user, you need the Resource Measurement Facility option available with z/OS.
4. Collect all required information. The Program Directory lists publications useful during installation.
5. Confirm you have adequate DASD storage and create SMP/E DDDEF entries for each data set. See the Program Directory for details.

Chapter 2. Installation, initialization, and customization

For this topic, you need to understand these terms:

Installation options

You create an installation options data set that specifies these options. They become active when you start ICSF, customizing how ICSF runs on your system.

Startup procedure

You create an ICSF startup procedure. Along with other information, this specifies the name of the installation options data set.

SYS1.SAMPLIB

Contains samples, including an installation options data set, a CKDS allocation job, a PKDS allocation job, a startup procedure, a CICS Wait List data set, and sample JCL for SMP/E Delivery to load keys by using a pass phrase. You can update this code as necessary and generally store the updated code in SYS1.PARMLIB and SYS1.PROCLIB.

SYS1.PARMLIB

Generally contains the installation options data set. The installation options data set can alternately be a member of a partitioned or sequential data set.

SYS1.PROCLIB

Contains the startup procedure.

Steps for installation and initialization

Refer to the *z/OS Program Directory* for installation instructions. Several of the installation steps in the *z/OS Program Directory* refer you to this publication for details. This publication explains these installation steps.

Note: Because it is possible for ICSF control blocks like the DACC and CCVT to persist in storage across an ICSF restart, an IPL is required when installing a new release of ICSF.

1. Customize SYS1.PARMLIB. [“Steps to customize SYS1.PARMLIB” on page 10](#) describes this task.
2. Create the Cryptographic Key Data Set (CKDS). [“Steps to create the CKDS” on page 12](#) describes this. Create the Public Key Data Set (PKDS). [“Steps to create the PKDS” on page 15](#) describes this task.
3. If PKCS #11 support is desired, create the TKDS. [“Steps to create the TKDS” on page 18](#) describes this task.
4. Create the installation options data set. [“Steps to create the installation options data set” on page 21](#) describes this task.
5. Create the startup procedure. [“Steps to create the ICSF startup procedure” on page 25](#) describes this task.
6. Provide access to the ICSF panels. [“Steps to provide access to the ICSF panels” on page 26](#) describes this task.

Note: You only need to perform the first six steps once.

7. Start ICSF for the first time. See [“Steps to start ICSF for the first time” on page 28](#). Once ICSF has been started, Master Keys can be entered.

For additional information on ICSF first time startup, refer to [“Checklist for first-time startup of ICSF” on page 381](#). See [z/OS Cryptographic Services ICSF Administrator's Guide](#) for directions on entering Master Keys.

8. Enter Master Keys.

Other topics in this publication and [z/OS Cryptographic Services ICSF Administrator's Guide](#) provide additional installation information.

For information on installing the CICS-ICSF Attachment Facility, refer to [Appendix C, “CICS-ICSF Attachment Facility,”](#) on page 377.

Steps to customize SYS1.PARMLIB

The installation options data set you will create is generally stored in SYS1.PARMLIB. If your administrator does not have access to SYS1.PARMLIB, you need to use another data set instead.

Update the data set you are using as follows:

1. Add CEE.SCEERUN, CSF.SCSFMODE, and CSF.SCSFSTUB to the LNKST concatenation. This adds the ICSF library to the z/OS library search. This is an example of an ICSF entry to the LNKST concatenation.

```
CSF.SCSFMODE
```

2. APF authorize CSF.SCSFMODE and CSF.SCSFSTUB, if LNKAUTH=APFTAB. This is an example of an ICSF entry for APF authorization.

```
APF ADD DSNAME(CSF.SCSFMODE) VOLUME(*****)
```

3. In the IKJTSOxx parameter, add CSFDAUTH and CSFDPKDS as a value in the AUTHPGM parameter list and in the AUTHTSF parameter list. This is an example of an ICSF entry in the IKJTSOxx member.

```
AUTHPGM NAMES(          /* AUTHORIZED PROGRAMS          */ +
.....
.....
CSFDAUTH              /* ICSF              */ +
CSFDPKDS              /* ICSF              */ +
.....
AUTHTSF NAMES(          /* PROGRAMS TO BE AUTHORIZED WHEN */ +
.....
.....
CSFDAUTH              /* ICSF              */ +
CSFDPKDS              /* ICSF              */ +
.....
```

4. If your application programmers intend to use PKCS #11 token key objects for AES Galois/Counter Mode (GCM) encryption or GMAC generation and have ICSF generate the initialization vectors, then you need to ensure that the first four bytes of the sysplex names (from parmlib member COUPLExx) and the first four bytes of the system name (from the SYSNAME parameter in the IEASYSxx parmlib member) are unique within the scope of the systems that will be sharing these tokens. z/OS currently does not impose any restrictions on uniqueness between sysplex names and system names. Eight character system names have to be unique within a sysplex. ICSF requires that the first four bytes are unique, but this is not enforced by the z/OS operating system.

This needs to be done because, for AES GCM encryption or GMAC generation, the security of the algorithm is dependent on never repeating a key, initialization vector combination for two or more distinct sets of data. In PKCS #11, applications can request that ICSF generate a new (unique) initialization vector each time AES GCM or GMAC is initiated. In fact, this is the only permitted way to perform AES GCM or GMAC when PKCS #11 is operating in FIPS mode. When ICSF generates initialization vectors, it uses the ECVTSPLX (sysplex mode) or CVTSNAME (non-sysplex mode) field as the cryptographic module name. The name ensures uniqueness if such keys are distributed to multiple systems, but only if each system is set with a unique name.

When setting ECVTSPLX or CVTSNAME to unique values, be aware that ICSF uses only the first (left most) 4 characters of these fields. For this reason, these 4 characters must be set to uniquely identify the system.

For example, suppose AES key value 123 is created on the current single-image system (known as System A) and is distributed to another system residing in a Sysplex (known as Sysplex B). Both

systems will be performing GCM encryption where ICSF generates the initialization vectors. To ensure that unique initialization vectors are generated, set CVTSNAME=SYSA on System A and ECVTSPLX=PLXB on Sysplex B.

CVTSNAME is normally set from the SYSNAME=value statement in the IEASYSxx member of "SYS1.PARMLIB". For more information, see [z/OS MVS Initialization and Tuning Reference](#).

ECVTSPLX is normally set from the COUPLE SYSPLEX(value) in the COUPLExx member of "SYS1.PARMLIB". For more information, see [z/OS MVS Setting Up a Sysplex](#).

Note:

1. If you will be using the TKE workstation on this host, you should also add CSFTTKE as a value in the AUTHCMD parameter list.
2. To change the active IKJTSOxx member of SYS.PARMLIB without an IPL, use the PARMLIB UPDATE command.

[z/OS MVS Initialization and Tuning Guide](#) and [z/OS MVS Initialization and Tuning Reference](#) provide more information.

Creating the CKDS

Installations need to understand and plan for the system resources required for managing the CKDS copy in virtual storage, particularly when the installation is deploying a very large CKDS. Refer to “[ICSF system resource planning for the CKDS](#)” on page 11 for guidelines. Once you understand these guidelines, refer to “[Steps to create the CKDS](#)” on page 12 for step-by-step instructions.

ICSF system resource planning for the CKDS

Like the PKDS and TKDS, ICSF manages a mirror copy of the CKDS data set in protected, private virtual storage to optimize cryptographic workload access to symmetric keys in the normal course of workload operation. This copy is kept current as keys are dynamically added to, and removed from, the active CKDS key store. Like any set of control information that is maintained in virtual storage, the in-storage CKDS copy must be accommodated with sufficient system central storage and auxiliary paging space resources.

Installations need to understand and plan for the system resources that are required for managing the CKDS copy in virtual storage, particularly when the installation is deploying a large CKDS. Note that “very large” is a relative assessment depending upon the installation, and might be expressed, for example, in terms of tens or hundreds of thousands of symmetric keys in the CKDS, or even millions of keys.

An in-storage copy of a CKDS that is not experiencing significant dynamic key creation or deletion activity consumes a stable amount of virtual storage, and therefore a stable amount of system backing resource. However, certain occasional but unavoidable ICSF functions such as CKDS refresh do generate a significant spike in the amount of used virtual storage, and therefore a greater temporary demand for system resources backing that virtual storage.

Given these circumstances, it is important to calculate and plan for the system central storage and auxiliary paging space that is required to support an active in-storage copy. For a CKDS shared across a sysplex environment, every active ICSF in the sysplex has an equivalent resource requirement.

Each symmetric key in the CKDS is managed with one VSAM record. Installations need to plan for the appropriate amount of combined central storage and auxiliary paging space for each VSAM record, per active ICSF. The following formula is provided to help you calculate the required system virtual storage backing resource for an active in-storage CKDS. In this formula HI-A-RBA is the allocated relative byte address for the data component of a CKDS VSAM data set. The IDCAMS LISTCAT command output for a CKDS VSAM data set can be consulted to determine the HI-A-RBA value for the data component. The %Free Space used in this formula represents the percentage of free space in the CKDS VSAM data set. The IDCAMS EXAMINE DATATEST command output can be consulted to determine the percentage of free space.

$$\text{HI-A-RBA} \times ((100 - \% \text{Free Space}) / 100) \times 6$$

For example, the central storage and auxiliary paging space requirement for a CKDS VSAM data set with a HI-A-RBA of 481,787,904 for its data component entry and 16 percent free space can be calculated as follows.

$$481,787,904 \times ((100 - 16) / 100) \times 6 = 2,428,211,036.16 \text{ bytes}$$

This CKDS VSAM data set requires 2.26 Gigabytes of combined central storage and auxiliary paging space for system backing resource.

As is the case with all virtual storage usage, central storage is the preferred medium to optimize the workload performance, and to avoid system paging overhead. Excessive system paging due to any virtual storage usage can cause degradation across the workload and system operation, and an extreme shortage of central storage and auxiliary paging space can lead to a catastrophic system failure.

Note: The output from the preceding formulas should be added to the outputs calculated from the formulas in “ICSF system resource planning for the PKDS” on page 15 and “ICSF system resource planning for the TKDS and session object memory areas” on page 17. This gives you the required system virtual storage backing resource for all of ICSF’s KDS data sets. This value represents the required amount of virtual storage for a given instance of ICSF. For a set of KDS data sets shared across a sysplex environment, every active ICSF in the sysplex has an equivalent resource requirement.

Additional CKDS performance considerations

IBM recommends that installations that deploy a fixed-length format CKDS with millions of symmetric keys do not enable CKDS MAC authentication or disable it if it is already enabled. CKDS MAC authentication adds an additional coprocessor request for each VSAM data set read/write operation. There is a significant performance implication for CKDS MAC authentication that would be greatly magnified with such a large CKDS.

Steps to create the CKDS

The CKDS must be a key-sequenced data. There are three formats:

- A fixed length record format with LRECL=252.
- A variable length record format with LRECL=1024.
- The common record format (KDSR) which is common to all key data sets with LRECL=2048.

Allocate the CKDS on a permanently resident volume.

Attention: Ensure that this volume is not subject to data set migration. If the CKDS is migrated, message CSFM450E is issued and ICSF ends.

For detailed information about calculating space for a VSAM data set and an explanation of keyed-direct update processing and what happens when control area and control interval splits occur, see [*z/OS DFSMS Access Method Services Commands*](#).

1. Determine the amount of primary space you need to allocate for the CKDS. This should reflect the total number of entries you expect the data set to contain originally. Besides transport keys, PIN keys, data-encrypting keys, data-translating keys, and MAC keys, the CKDS contains a header record and system keys. ICSF no longer uses the system keys as of ICSF FMID HCR77A1, but they remain for older releases which may share the CKDS.

Fixed length record format:

Each record is 252 bytes long. Allocate space for all of the installation and system keys you expect to store in the CKDS.

Variable length record format:

The minimum size of a record will be 276 bytes. Records containing fixed-length DES and AES keys will be 332 bytes long. Records containing variable-length symmetric key tokens may be up to 993 bytes long. Allocate space for all of the installation and system keys you expect to store in the CKDS.

KDSR format:

The minimum size of a record will be 188 bytes. Records containing fixed-length DES and AES keys will be 244 bytes long or 300 bytes long if the original record had user data. Records containing variable-length symmetric key tokens may be up to 905 bytes long. In addition, installations may add metadata to any record. If you are planning to add metadata, account for the size of the metadata in the length of records. Allocate space for all of the installation and system keys you expect to store in the CKDS.

2. Determine the amount of secondary space to allocate for CKDS. This should reflect the total number of entries you expect to add to the data set.

To access keys, VSAM uses the key label as the VSAM key. This means that VSAM adds keys to the data set in collating sequence. That is, if two keys named A and B are in the data set, A appears earlier in the data set than B. As a result, adding keys to the data set can cause multiple VSAM control interval splits and control area splits. For example, a split might occur if the data set contains keys A, B, and E and you add C. In this case, C must be placed between B and E. These splits can leave considerable free space in the data set and can affect KGUP performance.

The amount of secondary space you allocate must take into account the number of control interval and control area splits that might occur. If the disk copy of the CKDS uses a significant amount of secondary space, you can copy it into another disk copy that you created with more primary space. You can do this by using the Access Method Services (AMS) REPRO command or the AMS EXPORT/IMPORT commands.

The BUFFERSPACE parameter on the AMS DEFINE CLUSTER command (required by Step “3” on page 13) lets VSAM optimize space for control area and control interval splits.

3. Create an empty VSAM data set to use as the CKDS. ICSF provides a sample job to define the CKDS in member CSFCKDS of SYS1.SAMPLIB.

Use the AMS DEFINE CLUSTER command to define the data set and to allocate its space.

Note: To improve security and reliability of the data that is stored on the CKDS:

- Use the ERASE and WRITECHECK parameters on the AMS DEFINE CLUSTER command. ERASE overwrites data records with binary zeros when the CKDS cluster is deleted. WRITECHECK provides hardware verification of all data that is written to the data set.
- Create a Security Server (RACF) data set profile for the CKDS. Ensure that no one has access to the CKDS data set by protecting the CKDS data set name resource in the DATASET class. If a data set profile is used, as opposed to using the PROTECTALL(FAIL) option for example, the profile should have a UACC of NONE.

Fixed length record format:

Allocate a disk copy of the CKDS by defining a VSAM cluster as in this SYS1.SAMPLIB CSFCKDS member sample:

```
//CSFCKDS JOB <JOB CARD PARAMETERS>
//*****
//* Licensed Materials - Property of IBM *
//* 5650-ZOS *
//* Copyright IBM CORP. 2002, 2015 *
//* *
//* This JCL defines a VSAM CKDS capable only of fixed-length records*
//* *
//* CAUTION: This is neither a JCL procedure nor a complete JOB. *
//* Before using this JOB step, you will have to make the following *
//* modifications: *
//* *
//* 1) Add the job parameters to meet your system requirements. *
//* 2) Be sure to change CSF to the appropriate HLQ if you choose *
//* not to use the default. *
//* 3) Change XXXXXX to the valid where you want your CKDS to *
//* reside. The CKDS needs to be on a permanently resident *
//* volume. *
//* *
//* NOTE: This JCL is specific for creating a CKDS capable of only *
//* fixed-length records. There are samples for each of the *
//* other key data sets and formats. *
//*
```

```

//*****
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE CLUSTER (NAME(CSF.CSFCKDS)          -
                    VOLUMES(XXXXXX)            -
                    RECORDS(100 50)            -
                    RECORDSIZE(252,252)        -
                    KEYS(72 0)                 -
                    FREESPACE(10,10)           -
                    SHAREOPTIONS(2))           -
    DATA (NAME(CSF.CSFCKDS.DATA)             -
          BUFFERSPACE(100000)                 -
          ERASE                                -
          WRITECHECK)                          -
    INDEX (NAME(CSF.CSFCKDS.INDEX))
/*

```

Variable length record format:

Allocate a disk copy of the CKDS by defining a VSAM cluster as in this SYS1.SAMPLIB CSFCKD2 member sample:

```

//CSFCKD2 JOB <JOB CARD PARAMETERS>
//*****
/* Licensed Materials - Property of IBM *
/* 5650-ZOS *
/* Copyright IBM CORP. 2010, 2013 *
/* *
/* This JCL defines a VSAM CKDS capable of variable-length records *
/* *
/* CAUTION: This is neither a JCL procedure nor a complete JOB. *
/* Before using this JOB step, you will have to make the following *
/* modifications: *
/* *
/* 1) Add the job parameters to meet your system requirements. *
/* 2) Be sure to change CSF to the appropriate HLQ if you choose *
/* not to use the default. *
/* 3) Change XXXXXX to the valid where you want your CKDS to *
/* reside. The CKDS needs to be on a permanently resident *
/* volume. *
/* *
/* NOTE: This JCL is specific for creating a CKDS capable of *
/* variable-length records, in non-KDSR format. There are *
/* samples for each of the other key data sets and formats. *
/* *
//*****
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE CLUSTER (NAME(CSF.CSFCKDS)          -
                    VOLUMES(XXXXXX)            -
                    RECORDS(100 50)            -
                    RECORDSIZE(332,1024)        -
                    KEYS(72 0)                 -
                    FREESPACE(10,10)           -
                    SHAREOPTIONS(2,3))           -
    DATA (NAME(CSF.CSFCKDS.DATA)             -
          BUFFERSPACE(100000)                 -
          ERASE                                -
          WRITECHECK)                          -
    INDEX (NAME(CSF.CSFCKDS.INDEX))
/*

```

KDSR record format:

Allocate a disk copy of the CKDS by defining a VSAM cluster as in this SYS1.SAMPLIB CSFCKD3 member sample:

```

//CSFCKD3 JOB <JOB CARD PARAMETERS>
//*****
/* Licensed Materials - Property of IBM *
/* 5650-ZOS *
/* Copyright IBM CORP. 2013 *
/* *
/* This JCL defines a VSAM CKDS capable of variable-length records *
/* in common record format *
/* *
/* CAUTION: This is neither a JCL procedure nor a complete JOB. *
/* Before using this JOB step, you will have to make the following *

```

```

/** modifications: *
/** *
/** 1) Add the job parameters to meet your system requirements. *
/** 2) Be sure to change CSF to the appropriate HLQ if you choose *
/** not to use the default. *
/** 3) Change XXXXXX to the valid where you want your CKDS to *
/** reside. The CKDS needs to be on a permanently resident *
/** volume. *
/** *
/** NOTE: This JCL is specific for creating a CKDS capable of *
/** variable-length records, in KDSR format. There are *
/** samples for each of the other key data sets and formats. *
/** *
/*******
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE CLUSTER (NAME(CSF.CSFCKDS) -
                    VOLUMES(XXXXXX) -
                    RECORDS(100 50) -
                    RECORDSIZE(372,2048) -
                    KEYS(72 0) -
                    FREESPACE(10,10) -
                    SHAREOPTIONS(2,3)) -
    DATA (NAME(CSF.CSFCKDS.DATA) -
          BUFFERSPACE(100000) -
          ERASE -
          WRITECHECK) -
    INDEX (NAME(CSF.CSFCKDS.INDEX))
/*

```

You can change and use the Job Control Language according to the needs of your installation. Please note that the JCL to define the CKDS differs from the JCL that defines the PKDS (RECORDSIZE and CISZ parameters). For more information about allocating a VSAM data set, see [z/OS DFSMS Access Method Services Commands](#).

Creating the PKDS

Installations need to understand and plan for the system resources required for managing the PKDS copy in virtual storage, particularly when the installation is deploying a very large PKDS. Refer to [“ICSF system resource planning for the PKDS” on page 15](#) for guidelines. Once you understand these guidelines, refer to [“Steps to create the PKDS” on page 15](#) for step-by-step instructions.

ICSF system resource planning for the PKDS

Like the CKDS and TKDS, ICSF manages a mirror copy of the PKDS data set in protected, private virtual storage to optimize cryptographic workload access to asymmetric keys. Again, similar to the CKDS, the in-storage PKDS copy must be accommodated with sufficient system central storage and auxiliary paging space resources. The same formula that is used in the system resource planning section for the CKDS can be used to estimate the virtual storage requirement for an existing, stable PKDS (one that is not experiencing significant dynamic asymmetric key creation or deletion activity).

$$HI-A-RBA \times ((100 - \%Free\ Space) / 100) \times 6$$

As described in [“ICSF system resource planning for the CKDS” on page 11](#), the output from running the IDCAMS LISTCAT and EXAMINE DATATEST commands against a PKDS VSAM data set can be consulted to determine the data set's data component HI-A-RBA and the percentage of free space in the data set.

Note: The output from the preceding formula should be added to the outputs calculated from the formulas in [“ICSF system resource planning for the CKDS” on page 11](#) and [“ICSF system resource planning for the TKDS and session object memory areas” on page 17](#). This gives you the required system virtual storage backing resource for all of ICSF's KDS data sets. This value represents the required amount of virtual storage for a given instance of ICSF. For a set of KDS data sets shared across a sysplex environment, every active ICSF in the sysplex has an equivalent resource requirement.

Steps to create the PKDS

The PKDS must be allocated and the PKDS data set name must be specified on the PKDSN parameter of the options data set when you first start ICSF.

The PKDS must be a key-sequenced data set with variable length records. Allocate the PKDS on a permanently resident volume.

1. Determine the amount of primary space you need to allocate for the PKDS.

This should reflect the total number of entries you expect the data set to contain originally. The PKDS will contain both public and private PKA keys. Each record has a maximum size of 3.5 KB. The average record length for a private key is 1.4 KB, and for a public key is 0.5 KB. Allocate space for a minimum of two private keys, one for digital signatures, and another for encipherment. In addition, allocate enough space for the number of public keys you expect to store in the PKDS. The number of public keys varies from system to system. Generally, only those keys that are received from other users or systems are stored in the PKDS. The public keys are used to send messages to the owners of the public keys. In addition, installations may add metadata to any record. If you are planning to add metadata, account for the size of the metadata in the length of records.

2. Determine the amount of secondary space to allocate for the PKDS.

This should reflect the total number of entries you expect to add to the data set. For detailed information about calculating space for a VSAM data set, see [z/OS DFSMS Access Method Services Commands](#).

To access keys, VSAM uses the key label as the VSAM key. This means that VSAM adds keys to the data set in collating sequence. That is, if two keys named A and B are in the data set, A appears earlier in the data set than B. As a result, adding keys to the data set can cause multiple VSAM control interval splits and control area splits. For example, a split might occur if the data set contains keys A, B, and E and you add C. In this case, C must be placed between B and E.

The amount of secondary space you allocate must take into account the number of control interval and control area splits that might occur. If the PKDS uses a significant amount of secondary space, you can copy it into another disk copy that you created with more primary space. You can do this by using the Access Method Services (AMS) REPRO command or the AMS EXPORT/IMPORT commands.

The BUFFERSPACE parameter on the AMS DEFINE CLUSTER command (required by Step “3” on page 16) lets VSAM optimize space for control area and control interval splits. For a detailed explanation of keyed-direct update processing and what happens when control area and control interval splits occur, see [z/OS DFSMS Access Method Services Commands](#).

3. Create an empty VSAM data set to use as the PKDS. Use the AMS DEFINE CLUSTER command to define the data set and to allocate its space. ICSF provides a sample job to define the PKDS in member CSFPKDS of SYS1.SAMPLIB.

Note: To improve security and reliability of the data that is stored on the PKDS:

- Use the ERASE parameter on the AMS DEFINE CLUSTER command. ERASE overwrites data records with binary zeros when the PKDS cluster is deleted.
- Create a Security Server (RACF) data set profile for the PKDS. Ensure that no one has access to the PKDS data set by protecting the PKDS data set name resource in the DATASET class. If a data set profile is used, as opposed to using the PROTECTALL(FAIL) option for example, the profile should have a UACC of NONE.
- The CISZ(8192) coded in this sample in the DATA section is a hardcoded requirement.

4. Allocate a disk copy of the PKDS by defining a VSAM cluster as in this SYS1.SAMPLIB CSFPKDS member sample:

```
//CSFPKDS JOB <JOB CARD PARAMETERS>
//*****
//* Licensed Materials - Property of IBM *
//* 5650-ZOS *
//* Copyright IBM CORP. 2002, 2015 *
//* *
//* This JCL defines a VSAM PKDS *
//* *
//* CAUTION: This is neither a JCL procedure nor a complete JOB. *
//* Before using this JOB step, you will have to make the following *
//* modifications: *
//* *
//* 1) Add the job parameters to meet your system requirements. *
```



```

/* 2) Be sure to change CSF to the appropriate HLQ if you choose *
/* not to use the default. *
/* 3) Change XXXXXX to the volid where you want your PKDS to *
/* reside. The PKDS needs to be on a permanently resident *
/* volume. *
/* *
/* NOTE: This JCL is specific for creating a PKDS. There are *
/* samples for each of the other key data sets and formats. *
/* *
/******
//DEFINE EXEC PGM=IDCAMS,REGION=64M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE CLUSTER (NAME(CSF.CSFPKDS) -
        VOLUMES(XXXXXX) -
        RECORDS(100 50) -
        RECORDSIZE(800,3800) -
        KEYS(72 0) -
        FREESPACE(0,0) -
        SHAREOPTIONS(2,3)) -
    DATA (NAME(CSF.CSFPKDS.DATA) -
        BUFFERSPACE(100000) -
        ERASE -
        CISZ(8192)) -
    INDEX (NAME(CSF.CSFPKDS.INDEX))
/*

```

You can change and use the Job Control Language according to the needs of your installation. Note that the JCL to define the PKDS differs from the JCL that defines the CKDS (RECORDSIZE and CISZ parameters). For more information about allocating a VSAM data set, see [z/OS DFSMS Access Method Services Commands](#).

Creating the TKDS

TKDS Installations need to understand and plan for the system resources required for managing the TKDS copy in virtual storage, particularly when the installation is deploying a very large TKDS. Refer to “[ICSF system resource planning for the TKDS and session object memory areas](#)” on page 17 for guidelines. Once you understand these guidelines, refer to “[Steps to create the TKDS](#)” on page 18 for step-by-step instructions.

ICSF system resource planning for the TKDS and session object memory areas

Like the CKDS and PKDS, ICSF manages a mirror copy of the TKDS data set in protected, private virtual storage to optimize cryptographic workload access to persistent PKCS #11 objects (keys, certificates, and so on). Also, like the CKDS and PKDS, the in-storage TKDS copy must be accommodated with sufficient system central storage and auxiliary paging space resources. Unfortunately, the variable length nature of PKCS #11 objects makes resource estimating for the TKDS difficult. The best way to estimate the virtual storage requirement for an existing, stable TKDS (one that is not experiencing significant dynamic PKCS #11 object creation or deletion activity) is to determine the actual size of the used DATA portion of the TKDS and multiply this by 3. The following formula is provided to help you calculate the required system virtual storage backing resource for an active in-storage TKDS. In this formula HI-A-RBA is the allocated relative byte address for the data component of a TKDS VSAM data set. The IDCAMS LISTCAT command output for a TKDS VSAM data set can be consulted to determine the HI-A-RBA value for the data component. The %Free Space used in this formula represents the percentage of free space in the TKDS VSAM data set. The IDCAMS EXAMINE DATATEST command output can be consulted to determine the percentage of free space.

$$\text{HI-A-RBA} \times ((100 - \% \text{Free Space}) / 100) \times 3$$

For example, if the DATA HI-A-RBA has the value 1622016 with 56% free space, then the virtual storage requirement estimate would be $1622016 \times (44/100) \times 3 = 4282122$ bytes or 4182 Kilobytes.

In addition to the persistent PKCS #11 objects that are stored in the TKDS, applications can also make use of temporary (session) objects. These too occupy ICSF protected, private virtual storage and should be accounted for. However, since these objects are not stored in the TKDS, it is impossible to estimate their virtual storage requirements without having some knowledge of the applications that are using PKCS #11. Fortunately, most applications that use PKCS #11 use only a few PKCS #11 session objects and

their storage requirements are already factored into the preceding TKDS estimate. However, some applications, such as TCP/IP's IPsec, use session objects exclusively, and can use many of them. Estimating the virtual storage requirements for these is beyond the scope of this document. Applications that use PKCS #11 session objects have an overall upper limit of 128 Megabytes per application address space for session objects.

Note: The output from the preceding formula should be added to the outputs calculated from the formulas in “ICSF system resource planning for the CKDS” on page 11 and “ICSF system resource planning for the PKDS” on page 15. This gives you the required system virtual storage backing resource for all of ICSF's KDS data sets. This value represents the required amount of virtual storage for a given instance of ICSF. For a set of KDS data sets shared across a sysplex environment, every active ICSF in the sysplex has an equivalent resource requirement.

Steps to create the TKDS

To enable applications to create and use persistent PKCS #11 tokens and objects using the PKCS #11 services, the TKDS must be allocated and the TKDS data set name must be specified on the TKDSN parameter of the options data set when you first start ICSF.

The TKDS must be a key-sequenced data set with variable length records. Allocate the TKDS on a permanently resident volume.

For detailed information about calculating space for a VSAM data set and an explanation of keyed-direct update processing and what happens when control area and control interval splits occur, see [z/OS DFSMS Access Method Services Commands](#).

1. Determine the amount of primary space you need to allocate for the TKDS.

This should reflect the total number of entries you expect the data set to contain originally. The TKDS will contain PKCS #11 tokens and objects. Each record has a maximum size of 32 KB. A record for a token will use 0.1 KB. The minimum size of a record for objects is: Data: 1 KB, Secret Key: 1.1 KB, Public Key: 1.5 KB, Private Key: 3.4 KB, Certificate: 1 KB, Domain Parameter: 1.5KB. Allocate enough space for the number of tokens to be supported and for the number of objects to be created. In addition, installations may add metadata to any record. If you are planning to add metadata, account for the size of the metadata in the length of records. Note that session objects are not stored in the TKDS.

2. Determine the amount of secondary space to allocate for the TKDS.

This should reflect the total number of entries you expect to add to the data set.

To access tokens and objects, VSAM uses the token handle or object handle as the VSAM key. This means that VSAM adds objects to the data set in collating sequence. That is, if two objects named A and B are in the data set, A appears earlier in the data set than B. As a result, adding objects to the data set can cause multiple VSAM control interval splits and control area splits. For example, a split might occur if the data set contains objects A, B, and E and you add C. In this case, C must be placed between B and E.

The amount of secondary space you allocate must take into account the number of control interval and control area splits that might occur. If the TKDS uses a significant amount of secondary space, you can copy it into another disk copy that you created with more primary space. You can do this by using the Access Method Services (AMS) REPRO command or the AMS EXPORT/IMPORT commands.

The BUFFERSPACE parameter on the AMS DEFINE CLUSTER command (required by Step “3” on page 18) lets VSAM optimize space for control area and control interval splits.

3. Create an empty VSAM data set to use as the TKDS. Use the AMS DEFINE CLUSTER command to define the data set and to allocate its space. ICSF provides a sample job to define the TKDS in member CSFTKDS of SYS1.SAMPLIB.

Note: To improve security and reliability of the data that is stored on the TKDS:

- Use the ERASE parameter on the AMS DEFINE CLUSTER command. ERASE overwrites data records with binary zeros when the TKDS cluster is deleted.

- Create a Security Server (RACF) data set profile for the TKDS. Ensure that no one has access to the TKDS data set by protecting the TKDS data set name resource in the DATASET class. If a data set profile is used, as opposed to using the PROTECTALL(FAIL) option for example, the profile should have a UACC of NONE.

4. Allocate a disk copy of the TKDS by defining a VSAM cluster with one of the following samples:

SYS1.SAMPLIB CSFTKDS member sample is used to define a TKDS in non-KDSR format:

```
//CSFTKDS JOB <JOB CARD PARAMETERS>
//*****
//* Licensed Materials - Property of IBM *
//* 5650-ZOS *
//* Copyright IBM CORP. 2007, 2013 *
//* *
//* This JCL defines a VSAM TKDS *
//* *
//* CAUTION: This is neither a JCL procedure nor a complete JOB. *
//* Before using this JOB step, you will have to make the following *
//* modifications: *
//* *
//* 1) Add the job parameters to meet your system requirements. *
//* 2) Be sure to change CSF to the appropriate HLQ if you choose *
//* not to use the default. *
//* 3) Change XXXXXX to the valid where you want your TKDS to *
//* reside. The TKDS needs to be on a permanently resident *
//* volume. *
//* *
//* NOTE: This JCL is specific for creating a TKDS. There are *
//* samples for each of the other key data sets and formats. *
//* *
//*****
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE CLUSTER (NAME(CSF.CSFTKDS) -
        VOLUMES(XXXXXX) -
        RECORDS(100 50) -
        RECORDSIZE(2200,32756) -
        KEYS(72 0) -
        FREESPACE(0,0) -
        SPANNED -
        SHAREOPTIONS(2,3)) -
        DATA (NAME(CSF.CSFTKDS.DATA) -
        BUFFERSPACE(100000) -
        ERASE) -
        INDEX (NAME(CSF.CSFTKDS.INDEX))
/*
```

SYS1.SAMPLIB CSFTKD2 member sample is used to define a TKDS in KDSR format:

```
//CSFTKD2 JOB <JOB CARD PARAMETERS>
//*****
//* Licensed Materials - Property of IBM *
//* 5650-ZOS *
//* Copyright IBM CORP. 2013 *
//* *
//* This JCL defines a VSAM TKDS which is initialized to use common *
//* record format *
//* *
//* CAUTION: This is neither a JCL procedure nor a complete JOB. *
//* Before using this JOB step, you will have to make the following *
//* modifications: *
//* *
//* 1) Add the job parameters to meet your system requirements. *
//* 2) Be sure to change CSF to the appropriate HLQ if you choose *
//* not to use the default. *
//* 3) Change XXXXXX to the valid where you want your TKDS to *
//* reside. The TKDS needs to be on a permanently resident *
//* volume. *
//* *
//* NOTE: This JCL is specific for creating a TKDS which is *
//* initialized to use common record format. There are *
//* samples for each of the other key data sets and formats. *
//* *
//*****
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
```

```

DEFINE CLUSTER (NAME(CSF.CSFTKDS)          -
                VOLUMES(XXXXXX)            -
                RECORDS(100 50)             -
                RECORDSIZE(2200,32756)      -
                KEYS(72 0)                  -
                FREESPACE(0,0)              -
                SPANNED                     -
                SHAREOPTIONS(2,3))          -
DATA (NAME(CSF.CSFTKDS.DATA)              -
      BUFFERSPACE(100000)                  -
      ERASE)                                -
INDEX (NAME(CSF.CSFTKDS.INDEX))

/*
//*-----*
//* Repro header record into the TKDS      *
//*-----*
//MKHEAD   EXEC PGM=IEBGENER
//SYSPRINT DD  SYSOUT=*
//SYSUT1    DD  *

//SYSUT2    DD  DSN=&&GENTMP,UNIT=SYSDA,DISP=(,PASS),
//          DCB=(RECFM=FB,LRECL=156,BLKSIZE=1560),SPACE=(TRK,(1,1))
//SYSIN     DD  *
GENERATE MAXFLDS=10,MAXLITS=156
RECORD FIELD=(20,X'00000000000000000000000000000000',,1),
        FIELD=(20,X'00000000000000000000000000000000',,21),
        FIELD=(20,X'E3C8C4D9000000000000000000000000',,41),
        FIELD=(20,X'00000000000000000000000000000000',,61),
        FIELD=(16,X'00000000000000000000000000000000',,81),
        FIELD=(16,X'00000000000000000000000000000000',,97),
        FIELD=(4,X'0000009C',,113),
        FIELD=(16,X'00000000000000000000000000000000',,117),
        FIELD=(20,X'00000000000000000000000000000000',,133),
        FIELD=(4,X'00000200',,153)

/*
//REPROKSD EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//SYSDATA   DD  DSN=*.MKHEAD.SYSUT2,DISP=(OLD,DELETE)
//SYSIN     DD  *
REPRO INFILE(SYSDATA) -
      OUTDATASET(CSF.CSFTKDS)
/*

```

You can change and use the Job Control Language according to the needs of your installation. For more information about allocating a VSAM data set, see [z/OS DFSMS Access Method Services Commands](#).

ICSF system resource planning for random number generation

Several ICSF callable services support psuedo-random number generation on behalf of system and application requests. ICSF's random number generation implementation utilizes a minimum virtual storage footprint of 256 kilobytes. To avoid system paging overhead, installations should plan for 256 kilobytes of central storage to back this footprint. This should be sufficient for most workloads, but for some workloads that are excessively heavy with multitasking random number generation requests, ICSF may dynamically extend that footprint 64 kilobytes at a time to optimize random number request handling.

In some cases, the system or application random number request may require that FIPS (Federal Information Processing Standards) certified random content be provided. In other cases, FIPS certified random content is not required. In either case, ICSF may employ one of multiple techniques to derive the random content. For both FIPS certified random content and for non-FIPS certified random content, the availability of CCA and/or PKCS #11 coprocessors enables ICSF to derive the random content without imposing significant CPU overhead on the system. Either type of coprocessor can be exploited for non-FIPS certified content, but only a PKCS #11 coprocessor can be used to avoid CPU cycles for FIPS certified random content.

Installations may wish to plan for CCA and/or PKCS #11 coprocessor availability to avoid potentially excessive CPU cycles being exhausted on random number content generation.

Steps to create the installation options data set

The *installation options data set* is a file that you create that contains installation options. It becomes active when you start ICSF.

- The installation options data set can be a member of PARMLIB, a member of a partitioned data set, or a sequential data set.
- The format of each record in the data set must be fixed length or fixed block length.
- A physical line in the data set is 80 characters long. The system ignores any characters in positions 72 to 80 of the line.
- A logical line is one or more physical lines. You can group physical lines into a logical line by placing a comma at the end of the information. Only a comment can appear after the comma. The system ignores any other information between the comma and column 71.
- Continuation causes the next physical line to append immediately following the comma. The system removes all leading blanks on the next physical line.
- You can delimit comments by /* and */ and include them anywhere within the text. A comment cannot span physical records. The system removes comments from a logical line before parsing it. It ignores physical lines that contain only comments.
- Specify only one option setting or keyword on a logical line. (If you specify more than one, the system ignores all but the last one on the line. The system reports syntax errors, but the errors do not cause it to stop interpreting the file.)

ICSF provides a sample installation options data set. The sample data set uses the recommended values for each option.

1. When you are starting ICSF for the first time:
 - a. Change the name of the data set on the CKDSN and PKDSN statements to the name of the empty VSAM datasets you created previously (in Step “3” on page 13 and Step “4” on page 16).
 - b. For a complete description of options you may want to change after the first start, see “Customizing ICSF after the first start” on page 30.)
2. Store the updated data set in SYS1.PARMLIB.

Note: For convenience, the installation options data set generally resides in SYS1.PARMLIB. If your cryptographic administrator does not have update access to SYS1.PARMLIB, store installation options in another data set, and RACF-protect it.

The sample installation options data set is as follows in SYS1.SAMPLIB: CSFPRM00

```
/******  
/*      LICENSED MATERIALS - PROPERTY OF IBM      */  
/*      */  
/*      5650-ZOS      */  
/*      */  
/*      COPYRIGHT IBM CORP. 1990, 2013      */  
/*      */  
/*      THIS IS A SAMPLE OF THE ICSF OPTIONS DATASET      */  
/*      */  
/******  
CKDSN(CSF.CSFCKDS)  
PKDSN(CSF.CSFPKDS)  
COMPAT(NO)  
SSM(NO)  
CHECKAUTH(NO)  
CTRACE(CTICSF00)  
USERPARM(USERPARM)  
REASONCODES(ICSF)
```

Note: See “Parameters in the installation options data set” on page 30 for descriptions of these parameters.

Use of system symbols in the options data set is supported. System symbols can be used as values for any of the parameters. System symbols must be no more than 8 characters.

Note: ICSF allows the CKDS, PKDS and TKDS data set names to be a maximum of 44 characters with up to 21 qualifiers. Also, the first character must be alphabetic.

See “Parameters in the installation options data set” on page 30 for additional information.

This example shows how system symbols could be used for the CKDS and PKDS data set names. You could use a SYS1.PARMLIB(IEASYMxx) file and modify CSFPRM00.

IEASYMxx file could contain:

```
/*-----*/
/* SYSTEM SYMBOLS FOR ICSF CRYPTO */
/*-----*/
SYSDDEF
  SYMDEF(&CKDSN001='CSF')
  SYMDEF(&CKDSN002='CSFCKDS')
  SYMDEF(&PKDSN001='CSF')
  SYMDEF(&PKDSN002='CSFCKDS')
```

CSFPRM00 could be modified as follows.

```
/*-----*/
/* LICENSED MATERIALS - PROPERTY OF IBM */
/* */
/* 5650-ZOS */
/* */
/* COPYRIGHT IBM CORP. 1990, 2013 */
/* */
/* THIS IS A SAMPLE OF THE ICSF OPTIONS DATASET */
/* */
/*-----*/
CKDSN(&CKDSN001..&CKDSN002)
PKDSN(&PKDSN001..&PKDSN002)
COMPAT(NO)
SSM(NO)
CHECKAUTH(NO)
CTRACE(CTICSF00)
USERPARM(USERPARM)
REASONCODES(ICSF)
```

This example shows how system symbols could be used for the Regional Cryptographic Server (RCS) port numbers. You could use a SYS1.PARMLIB(IEASYMxx) file and modify CSFPRM00.

IEASYMxx file could contain:

```
/*-----*/
/* SYSTEM SYMBOLS FOR ICSF CRYPTO */
/*-----*/
SYSDDEF
  SYMDEF(&RDPORT='1125')
```

CSFPRM00 could be modified as follows.

```
/*-----*/
/* LICENSED MATERIALS - PROPERTY OF IBM */
/* */
/* 5650-ZOS */
/* */
/* COPYRIGHT IBM CORP. 2015 */
/* */
/* THIS IS A SAMPLE OF THE ICSF OPTIONS DATASET */
/* */
/*-----*/
CKDSN(CSF.CSFCKDS)
PKDSN(CSF.CSFCKDS)
TKDSN(CSF.CSFCKDS)
COMPAT(NO)
SSM(NO)
CHECKAUTH(NO)
CTRACE(CTICSF00)
USERPARM(USERPARM)
REASONCODES(ICSF)
REMOTEDevice(1,MY.SERVER.DOMAIN.COM,&RDPORT,8)
```

When the machine or partition is IPLed, specify within the load parameter the symbol file that should be used. For example, if the previous symbol file was called IEASYM01, then within the load member, the IEASYM entry might look like

```
IEASYM(00,01); where 00 denotes the IEASYM00 file (usually the system default)
and 01 denotes the IEASYM01 file.
```

When the machine or partition is IPLed, specify within the load parameter the symbol file that should be used. For example, if the previous symbol file was called IEASYM01, then within the load member, the IEASYM entry might look like IEASYM(00,01); where 00 denotes the IEASYM00 file (usually the system default) and 01 denotes the IEASYM01 file.

Creating an ICSF CTRACE configuration data set

Starting with ICSF FMID HCR77A1, ICSF CTRACE support has been enhanced to support configurable ICSF CTRACE options from PARMLIB. During SMP/E install, a default CTICSF00 PARMLIB member is installed in SYS1.PARMLIB. The CTICSF00 PARMLIB member provides default component trace values for ICSF. By default, ICSF CTRACE support will trace with the KdsIO, CardIO, RdIO, and SysCall filters using a 2M buffer. Configurable options are commented out within this PARMLIB member to provide examples of how to turn them on.

Note: Beginning with FMID HCR77A1, ICSF needs to have read access to all data sets in the PARMLIB concatenation to access the CTRACE parmlib member CTICSF00.

The CTICSF00 PARMLIB member can be used to create customized ICSF CTRACE Configuration Data Sets in PARMLIB. A customized ICSF CTRACE Configuration Data Set can then be specified in the ICSF Options Data Set using the new CTRACE option.

For example, CTRACE(CTICSFxx), where xx is any 2 characters that were used when copying the default CTICSF00 parmlib member.

Component tracing is active when ICSF starts using the trace options defined in the CTICSFxx PARMLIB member, where 00 is the default. If the specified PARMLIB member is incorrect or absent, ICSF CTRACE will attempt to use the default CTICSF00 PARMLIB member. If the CTICSF00 PARMLIB member is incorrect or absent, ICSF CTRACE will perform tracing using an internal default set of trace options. The operator can specify trace options individually on the TRACE CT command, or can specify the name of a CTICSFxx PARMLIB member containing the desired trace options. Using a PARMLIB member on the TRACE CT command can help minimize operator intervention and avoid syntax or keystroke errors.

The contents of the CTICSF00 PARMLIB member, is as follows:

```
/***START OF SPECIFICATIONS*****//
/*
/* $MAC (CTICSF00) COMP(05101) PROD(CSF):
/*
/**01* MACRO NAME: CTICSF00
/*
/**01* DESCRIPTIVE NAME: CTRACE Options for ICSF Startup
/*
/**01* COPYRIGHT:
/*
/*    LICENSED MATERIALS - PROPERTY OF IBM
/*
/*    5650-ZOS
/*
/*    COPYRIGHT IBM CORP. 2015
/*
/*    STATUS = HCR77B1
/*
/**01* FUNCTION:
/*    Define the default ICSF CTRACE options
/*
/**01* COMPONENT: 05101 (CSF)
/*
/**01* DISTRIBUTION LIBRARY:  PARMLIB
/*
/***END OF SPECIFICATIONS*****//
TRACEOPTS
/*-----*/
/*    ON OR OFF: PICK 1
/*-----*/
/*
/*    ON
/*    OFF
/*-----*/
/*    ASID: 1 TO 16, 2-HEXBYTE VALUES
/*
```

```

/*-----*/
/*          ASID(0042,0043,0044)          */
/*-----*/
/* JOBNAME: 1 TO 16, 8 BYTE VALUES          */
/* This option takes 1 to 16 comma-separated 8 byte values. Each */
/* value specified represents a jobname that should be traced by */
/* ICSF CTRACE support. Additionally, other jobnames that begin */
/* with the same characters will also be traced. For example, if */
/* a USERID is specified, all TSO jobs matching USERIDc, where */
/* 'c' is a character between A-Z will be traced, and, all Unix */
/* processes matching USERIDn, where 'n' is a number from 0-9 */
/* will be traced.          */
/*-----*/
/*          JOBNAME(USERID,JOBNAME1)          */
/*-----*/
/* BUFSIZE: A VALUE IN RANGE 16K TO 16M          */
/*-----*/
/*          BUFSIZE(2M)          */
/*-----*/
/* OPTIONS: NAMES OF FUNCTIONS TO BE TRACED, OR "ALL", OR "MIN" */
/*-----*/
/*          OPTIONS(          */
/*              'ALL'          */
/*              , 'KDSIO'      */
/*              , 'CARDIO'     */
/*              , 'SYSCALL'    */
/*              , 'DEBUG'      */
/*              , 'RDIO'       */
/*              , 'RDDATA'     */
/*              , 'MIN'        */
/*              )          */
/*-----*/
OPTIONS('KDSIO','CARDIO','SYSCALL','RDIO')

```

TRACEOPTS - This option takes a value of either ON or OFF. Turning this option OFF reduces ICSF CTRACE to use a minimal set of tracing. Turning this option OFF disables ICSF CTRACE. When OFF is specified all other trace options within the PARMLIB options data set should be commented out

ASID - This option takes 1 TO 16 comma-separated 2-hexbyte values. Each value specified represents an address space ID that should be traced by ICSF CTRACE support

JOBNAME - This option takes 1 TO 16 comma-separated 8 byte values. Each value specified represents a jobname that should be traced by ICSF CTRACE support. Additionally, other jobnames that begin with the same characters will also be traced. For example, if a USERID is specified, all TSO jobs matching USERIDc, where 'c' is a character between A-Z will be traced, and, all Unix processes matching USERIDn, where 'n' is a number from 0-9 will be traced.

BUFSIZE - This option takes a value in the range between 16K to 16M, where K represents kilobytes and M represents megabytes. This value is used to specify the ICSF CTRACE buffer size to be allocated.

OPTIONS - This option is used to specify the ICSF CTRACE filters to use for tracing. A comma-separated list of filter names, each enclosed with single quotes, may be specified. The following filters are supported by this option:

ALL - This filter provides output for all ICSF trace records regardless of their filter specification.

CARDIO - This filter traces activity with requests to cryptographic coprocessors.

DEBUG - This filter provides granular trace output for debugging specific ICSF modules. This filter should only be turned on at the direction of IBM service professionals. Turning this level of tracing on may degrade ICSF performance.

KDSIO - This filter traces update activity to the CKDS, PKDS, and TKDS.

MIN - This filter traces a minimum set of operations that are not covered by the other filters.

RDDATA - This filter traces remote device request and response messages.

RDIO - This filter traces activity pertaining to remote device I/O events.

SYSCALL - This filter traces entry and exit from ICSF callable services.

The TRACEENTRY option in the ICSF Options Data Set has been deprecated. If this option is specified, it will be ignored and will produce a CSFO0212 message.

Steps to create the ICSF startup procedure

ICSF provides this job control language program. You can use this code as the basis for your startup procedure.

- member CSF in SYS1.SAMPLIB

```
//CSF PROC  
//CSF EXEC PGM=CSFINIT,REGION=0M,TIME=1440,MEMLIMIT=NOLIMIT  
//CSFPARM DD DSN=SYS1.PARMLIB(CSFPRM00),DISP=SHR
```

Store this startup PROC in SYS1.PROCLIB (or another suitable library).

1. Change or use the sample startup procedure according to your needs.
 - a. In the sample code, the first line is the PROC statement. You can add one or more procedure variables to the PROC statement. For example, you can allow the system operator to decide at start time which member of the installation options data set to use. This example allows the operator to enter START CSF,M=CSFPRM00, specifying an alternate set of start-up options.

```
//CSF PROC M=CSFPRM00  
.  
.  
.  
//CSFPARM DD DSN=MY.ICSF.PARM(&M),DISP=SHR
```

You can use the same principle to change the name of a sequential data set, if you are not using a partitioned data set.

- b. The last line is the CSFPARM DD statement. The sample code specifies SYS1.PARMLIB as the data set where the installation options data set is stored. If you stored the installation options data set elsewhere, replace SYS1.PARMLIB with the name of the data set where you stored the installation options.
 - c. The CSFPARM DD statement also specifies member CSFPRM00 as the name of the installation options data set. If you used a different name when you created the installation options data set (or any time you want to use other options), change this member name.
2. Store your startup procedure in SYS1.PROCLIB (or another suitable library) with a member name of your choice. (Depending on installation standards, possible names include CSF, CSFPROD, and CRYPTO.)
 3. If you use Security Server (RACF), you may need to update the RACF Started Procedure Table if you define a new started task:
 - a. Add the new started task name
 - b. Add a RACF userid to associate with the started task. See [z/OS Security Server RACF System Programmer's Guide](#) for more information.
 - c. Optionally, you can add a RACF group name.

Notes:

- SAF uses the userid associated with the ICSF address space when accessing the CKDS and PKDS named in the installation options data set both at ICSF startup and when performing coordinated functions (Coordinated Change-MK, Coordinated Refresh, or Coordinated Convert). When you perform a non-coordinated CKDS or PKDS task (Initialize, Change MK, Refresh, Convert), SAF uses the identity associated with the invoker (TSO userid when using panels under TSO/E or the userid associated with the batch address space when using a batch job).
- If you specify a REMOTEDEVICE entry in the ICSF installation options data set, ICSF will attempt to connect to this device using TCP/IP. Additional setup is required. For more information, see [“Adding and removing regional cryptographic servers”](#) on page 108.

Steps to provide access to the ICSF panels

To provide a way for the administrator to access the ICSF panels, you can create an ICSF option on the ISPF Primary Option Menu. Access the code for the ISPF Primary Option Menu panel body and perform these steps:

1. Under the % OPTION ==> _ZCMD line, add this line:

```
% <option value> - ICSF Panels
```

You can specify either a letter or number for the option value. Do not use an option value that already exists in the menu.

2. On the &ZSEL= TRANS(&ZQ line, add this information:

```
<option value>,'PANEL(CSF@PRIM) NEWAPPL(CSF)'
```

The option value should be the same value as the option value you chose to use in the preceding step.

When you access the ISPF Primary Option Menu panel, the ICSF panels option appears on the menu. You can choose the ICSF option value to access the ICSF panels.

You must also update the logon procedure that is used by ICSF administrators who will use the ICSF panels. For example:

```
//SYSPROC DD ...
.
.
//          DD DSN=CSF.SCSFCLIO,DISP=SHR
.
.
//ISPPLIB DD ...
.
.
//          DD DSN=CSF.SCSFPNL0,DISP=SHR
.
.
//ISPMLIB DD ...
.
.
//          DD DSN=CSF.SCSFMSG0,DISP=SHR
.
.
//ISPSLIB DD ...
.
.
//          DD DSN=CSF.SCSFSKL0,DISP=SHR
.
.
// ISPTLIB
.
.
//          DD DSN=CSF.SCSFTLIB,DISP=SHR
.
.
.
```

An alternate method to access the ICSF panels is to use ISPF LIBDEF. Here is a sample clist.

```
/* Rexx */
/* IBMs ICSF */

address ispexec

"LIBDEF ISPPLIB DATASET ID('CSF.SCSFPNL0') STACK"
"LIBDEF ISPMLIB DATASET ID('CSF.SCSFMSG0') STACK"
```

```

"LIBDEF ISPSLIB DATASET ID('CSF.SCSFSKLO') STACK"
"LIBDEF ISPTLIB DATASET ID('CSF.SCSFTLIB') STACK"

address tso "ALTLIB ACTIVATE APPLICATION(CLIST)
             DATASET('CSF.SCSFCLI0') "
"SELECT PANEL(CSF@PRIM) NEWAPPL(CSF) PASSLIB"
address tso "ALTLIB DEACTIVATE APPLICATION(CLIST)"

"LIBDEF ISPSLIB"
"LIBDEF ISPPLIB"
"LIBDEF ISPMLIB"
"LIBDEF ISPTLIB"

```

The *z/OS Program Directory* lists additional installation steps and some of these steps depend on the system from which you are migrating. See the *z/OS Program Directory*, other topics in this publication, and *z/OS Cryptographic Services ICSF Administrator's Guide* for details about the remaining steps.

Requiring signature verification for ICSF module CSFINPV2

If your installation needs to operate z/OS PKCS #11 in compliance with the FIPS 140-2 standard, then the integrity of the cryptographic functions shipped by IBM must be verified at your installation during ICSF startup. The load module that contains the software cryptographic functions is SYS1.SIEALNKE(CSFINPV2), and this load module is digitally signed when it is shipped from IBM. Using RACF, you can verify that the module has remained unchanged from the time it was built and installed on your system. To do this, you create a profile in the PROGRAM class for the CSFINPV2 module, and use this profile to indicate that signature verification is required before the module can be loaded.

To require signature verification for ICSF module CSFINPV2:

1. Make sure that RACF has been prepared to verify signed programs. As described in *z/OS Security Server RACF Security Administrator's Guide*, a security administrator prepares RACF to verify signed programs by creating a key ring for signature verification, and adding the code-signing CA certificate that is supplied with RACF to the key ring. If RACF has been prepared to verify signed programs, there will be a key ring dedicated to signature verification, the code-signing CA certificate will be attached to the key ring, and the PROGRAM class will be active.
 - a. If RACF has been prepared to verify signed programs, the discrete profile IRR.PROGRAM.SIGNATURE.VERIFICATION in the FACILITY class will specify the name of the signature-verification key ring. To determine if a signature key ring is already active, enter the command:

```
RLIST FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION
```

If there is no discrete profile with this name, have your security administrator prepare RACF to verify signed programs using the information in *z/OS Security Server RACF Security Administrator's Guide*.

- b. If the signature verification key ring exists, the RLIST command will display information for the discrete profile IRR.PROGRAM.SIGNATURE.VERIFICATION in the FACILITY class. The name of the signature verification key ring and the name of the key ring owner will be included in the APPLICATION DATA field of the RLIST command output. Using this information, enter the RACDCERT LISTRING command to make sure the code-signing CA certificate is attached to the key ring:

```
RACDCERT ID(key-ring-owner) LISTRING(key-ring-name)
```

The label of the code-signing CA certificate is 'STG Code Signing CA'. If this label is not shown in the RACDCERT LISTRING command output, have your security administrator prepare RACF to verify signed programs using the information in *z/OS Security Server RACF Security Administrator's Guide*.

- c. Program control must be active in order for RACF to perform signature verification processing. To make sure the PROGRAM class is active, enter the SETROPTS LIST command.

```
SETROPTS LIST
```

The ACTIVE CLASSES field of the command output should include the PROGRAM class. If it does not, have your security administrator prepare RACF to verify signed programs using the information in *z/OS Security Server RACF Security Administrator's Guide*.

2. Create a profile for the CSFINPV2 program module in the PROGRAM class, indicating that the program must be signed. The following command specifies that the program should fail to load if the signature cannot be verified for any reason. This command also specifies that all signature verification failures should be logged.

Note: Due to space constraints, this command example appears on two lines. However, the RDEFINE command should be entered completely on one line.

```
RDEFINE PROGRAM CSFINPV2 ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))
```

You will need to activate your profile changes in the PROGRAM class.

```
SETROPTS WHEN(PROGRAM) REFRESH
```

Steps to start ICSF for the first time

Now that you have created the key data sets, the installation data set, the started procedure, and the ICSF management panels, you can start ICSF.

For additional information on starting ICSF for the first time, see Appendix D, “Helpful hints for ICSF first time startup,” on page 381.

- Created an empty data set for use as a CKDS
- Specified the CKDS name in the installation options data set
- Created an empty data set for use as a PKDS
- Specified the PKDS name in the installation options data set
- If PKCS #11 support is desired, create the TKDS
- Created a startup procedure
- Installed ICSF

Steps for initializing ICSF

You must initialize ICSF and the cryptographic coprocessors:

1. Enter the START command and the startup procedure name. In this example, CSF is the name of the startup procedure.

```
START CSF
```

When you start ICSF, you specify the name of the ICSF startup procedure you created (see “Steps to create the ICSF startup procedure” on page 25). See “Starting and stopping ICSF” on page 85 for more information about starting and stopping ICSF.

Note: To reuse ASIDs, the REUSASID parameter can be added to the START comment:

```
START CSF,REUSASID=YES
```

2. Access the ICSF panels to define a master key and initialize the CKDS and PKDS. For a description of how to use the ICSF panels to define a master key and initialize the CKDS and PKDS at first-time startup, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

If you intend to use secure key PKCS #11 services, you will also need to initialize the TKDS. This step is optional and may be deferred until a later time. Initializing the TKDS requires entering the master key using a TKE workstation. For more information, see *z/OS Cryptographic Services ICSF TKE Workstation User's Guide*.

When defining a master key by specifying master key parts, **make sure the key parts are recorded and saved in a secure location**. When you are entering the key parts for the first time, be aware that **you may need to reenter these same key values at a later date to restore master key values that have been cleared**. If defining a master key using a pass phrase, realize that the same pass phrase will always produce the same master key values, and is therefore as critical and sensitive as the master key values themselves. Make sure you save the pass phrase so that you can later reenter it if needed. Because of the sensitive nature of the pass phrase, make sure you secure it in a safe place.

3. When you start ICSF for the first time, you will see different messages depending on your system hardware. The following examples show the messages returned on a IBM zEnterprise EC12 machine with one Crypto Express4 CCA coprocessor and one Crypto Express4 EP11 cryptographic coprocessor.
 - First time startup messages before master keys have been loaded and the CKDS, PKDS, and TKDS have not been initialized:

```
S CSF
CSFM608I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM608I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES SUCCESSFUL.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS4 COPROCESSOR 4Cxx, SERIAL NUMBER nnnnnnnn.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS4 COPROCESSOR 4Pxx, SERIAL NUMBER nnnnnnnn.
CSFM131E CRYPTOGRAPHY - SECURE KEY PKCS11 SERVICES ARE NOT AVAILABLE.
CSFM102I TOKEN DATA SET, CSF.TKDS IS NOT INITIALIZED FOR SECURE KEY PKCS11.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CKDS IS NOT INITIALIZED.
CSFM101E PKA KEY DATA SET, CSF.PKDS IS NOT INITIALIZED.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
CSFM001I ICSF INITIALIZATION COMPLETE
```

- First time startup messages before master keys have been loaded and sharing an initialized CKDS, PKDS, and TKDS:

```
S CSF
CSFM608I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM608I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES SUCCESSFUL.
CSFM124I MASTER KEY P11 ON CRYPTO EXPRESS4 COPROCESSOR 4Pxx, SERIAL NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY DES ON CRYPTO EXPRESS4 COPROCESSOR 4Cxx, SERIAL NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY AES ON CRYPTO EXPRESS4 COPROCESSOR 4Cxx, SERIAL NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY RSA ON CRYPTO EXPRESS4 COPROCESSOR 4Cxx, SERIAL NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY ECC ON CRYPTO EXPRESS4 COPROCESSOR 4Cxx, SERIAL NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
CSFM001I ICSF INITIALIZATION COMPLETE
```

- Normal ICSF restart messages. Master key registers are valid and match the CKDS/PKDS/TKDS:

```
S CSF
CSFM608I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM608I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES SUCCESSFUL.
CSFM129I MASTER KEY P11 ON CRYPTO EXPRESS4 COPROCESSOR 4Pxx, SERIAL NUMBER nnnnnnnn, IS CORRECT.
CSFM129I MASTER KEY DES ON CRYPTO EXPRESS4 COPROCESSOR 4Cxx, SERIAL NUMBER nnnnnnnn, IS CORRECT.
CSFM129I MASTER KEY AES ON CRYPTO EXPRESS4 COPROCESSOR 4Cxx, SERIAL NUMBER nnnnnnnn, IS CORRECT.
CSFM129I MASTER KEY RSA ON CRYPTO EXPRESS4 COPROCESSOR 4Cxx, SERIAL NUMBER nnnnnnnn, IS CORRECT.
CSFM129I MASTER KEY ECC ON CRYPTO EXPRESS4 COPROCESSOR 4Cxx, SERIAL NUMBER nnnnnnnn, IS CORRECT.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS4 COPROCESSOR 4Cxx, SERIAL NUMBER nnnnnnnn.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS4 COPROCESSOR 4Pxx, SERIAL NUMBER nnnnnnnn.
CSFM132I SECURE KEY PKCS11 SERVICES AVAILABLE.
CSFM400I CRYPTOGRAPHY - SERVICES ARE NOW AVAILABLE.
```

```

CSFM130I CRYPTOGRAPHY - RSA SERVICES ARE AVAILABLE.
CSFM130I CRYPTOGRAPHY - DES SERVICES ARE AVAILABLE.
CSFM130I CRYPTOGRAPHY - ECC SERVICES ARE AVAILABLE.
CSFM127I CRYPTOGRAPHY - AES SERVICES ARE AVAILABLE.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE
AVAILABLE.
CSFM001I ICSF INITIALIZATION COMPLETE

```

Notes:

1. When you are starting ICSF for the first time and loading the first master key and initializing one or more CKDS, PKDS, or TKDS, you provide the name of the empty VSAM data set you defined previously (see [“Steps to create the PKDS” on page 15 step 3](#)) to use for the CKDS, PKDS, and TKDS when starting ICSF.
2. While ICSF processes the data set, it requires exclusive use so that no one can make changes while the data set is read. ICSF releases the data set when it completes startup processing.
3. During CKDS, PKDS, and TKDS initialization or refresh, ICSF reads the CKDS, PKDS, or TKDS into extended private storage. Specify MEMLIMIT=NOLIMIT to ensure that ICSF does not run out of virtual storage.
4. You can also write application programs to call services to perform cryptographic functions. See [“Exits for the services” on page 132](#) for details.

Customizing ICSF after the first start

The startup procedure includes a CSFPARM DD statement, which gives the name of the installation options data set. The installation options data set includes a CKDSN option, which gives the names of the CKDS, and a PKDSN option, which gives the name of the PKDS.

After the first start, whenever you restart ICSF, the CKDS and PKDS named in the installation options data set becomes the active in-storage CKDS and PKDS.

In order for changes to the installation options dataset to take effect, **stop and restart ICSF**. A subset of option parameters in the installation options data set are refreshable starting with ICSF FMID HCR77C0. See the SETICSF command or ICSF Multi-Purpose Service (CSFMPS and CSFMPS6) for details. To change the active in-storage CKDS or PKDS, stop and restart ICSF, or use the REFRESH option of the Master Key Management panel.

Parameters in the installation options data set

The installation options data set is an intended programming interface.

When specifying parameter values within parentheses, leading and trailing blanks are ignored. Embedded blanks may cause unpredictable results.

Support is provided for the use of system symbols in the installation options data set. System symbols can be used as values for any of the parameters. System symbols are specified in the IEASYMxx member of SYS1.PARMLIB; the IEASYM statement of the LOADxx member of SYS1.PARMLIB specifies the IEASYMxx member or members to be used for the resolution of system symbols. This example shows the use of a system symbol for specifying the domain to be used for the start of ICSF:

```
DOMAIN(&PARDOM.)
```

When the Installation Options Data Set is processed during the start of ICSF, the value of the system symbol PARDOM will be substituted as the value of the DOMAIN parameter.

For the first start, you specified an empty VSAM data set name for the CKDS in the CKDSN option and an empty VSAM data set name for the PKDS in the PKDSN option. You may want to change these and other options for subsequent starts. Here is a complete list of installation options:

AUDITKEYLIFECKDS(TOKEN(YES or NO),LABEL(YES or NO))

Provides a set of options that control auditing of events related to the lifecycle of symmetric CCA tokens. The audit logs are in the form of Type 82 SMF records.

TOKEN(YES or NO)

Controls lifecycle auditing of CKDS tokens.

Value

Indication

YES

Indicates ICSF should audit lifecycle events related to CKDS tokens. An SMF type 82 subtype 40 record is logged for each event.

NO

No lifecycle auditing of CKDS tokens occurs.

LABEL(YES or NO)

Controls lifecycle auditing of CKDS labels.

Value

Indication

YES

Indicates ICSF should audit lifecycle events related to CKDS labels. An SMF type 82 subtype 40 record is logged for each event. The subtype 40 record replaces the subtype 9 record.

NO

No lifecycle auditing of CKDS labels occurs. ICSF continues to log an SMF type 82 subtype 9 record for CKDS updates.

If the AUDITKEYLIFECKDS option is not specified, the default is AUDITKEYLIFECKDS (TOKEN(NO),LABEL(NO)).

Note:

1. An event that involves a token is considered to be any request that uses a token as opposed to a label. This is true regardless of Key Store Policy enablement.
2. If auditing of CKDS labels is enabled, the Key Generator Utility Program (KGUP) needs access to the CSFGKF profile in the CSFSERV class in order to generate the key fingerprint for keys it processes.

For more information about the events that are audited as well as the information contained in the audit record, see Appendix B in [z/OS Cryptographic Services ICSF System Programmer's Guide](#) for the description for the subtype 40 record.

The auditing of key lifecycle events can also be controlled via the SETICSF operator command. See the description of the SETICSF command in [z/OS Cryptographic Services ICSF System Programmer's Guide](#) for more information.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

AUDITKEYLIFEPKDS(TOKEN(YES or NO),LABEL(YES or NO))

Provides a set of options that control auditing of events related to the lifecycle of asymmetric CCA tokens. The audit logs are in the form of Type 82 SMF records.

TOKEN(YES or NO)

Controls lifecycle auditing of PKDS tokens.

Value

Indication

YES

Indicates ICSF should audit lifecycle events related to PKDS tokens. An SMF type 82 subtype 41 record is logged for each event.

NO

No lifecycle auditing of PKDS tokens occurs.

LABEL(YES or NO)

Controls lifecycle auditing of PKDS labels.

Value

Indication

YES

Indicates ICSF should audit lifecycle events related to PKDS labels. An SMF type 82 subtype 41 record is logged for each event. The subtype 41 record replaces the subtype 13 record.

NO

No lifecycle auditing of PKDS labels occurs. ICSF continues to log an SMF type 82 subtype 13 record for PKDS updates.

If the AUDITKEYLIFEPKDS option is not specified, the default is AUDITKEYLIFEPKDS (TOKEN(NO),LABEL(NO)).

Note: An event that involves a token is considered to be any request that uses a token as opposed to a label. This is true regardless of Key Store Policy enablement.

For more information about the events that are audited as well as the information contained in the audit record, see Appendix B in *z/OS Cryptographic Services ICSF System Programmer's Guide* for the description for the subtype 41 record.

The auditing of key lifecycle events can also be controlled via the SETICSF operator command. See the description of the SETICSF command in *z/OS Cryptographic Services ICSF System Programmer's Guide* for more information.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

AUDITKEYLIFETKDS(TOKENOBJ(YES or NO),SESSIONOBJ(YES or NO))

Provides a set of options that control auditing of events related to the lifecycle of PKCS #11 objects. The audit logs are in the form of Type 82 SMF records.

TOKENOBJ(YES or NO)

Controls lifecycle auditing of PKCS #11 token objects.

Value

Indication

YES

Indicates ICSF should audit lifecycle events related to PKCS #11 token objects. An SMF type 82 subtype 42 record is logged for each event. The subtype 42 record replaces the subtype 23 record.

NO

No lifecycle auditing of PKCS #11 token objects occurs. ICSF continues to log an SMF type 82 subtype 23 record for TKDS updates.

SESSIONOBJ(YES or NO)

Controls lifecycle auditing of PKCS #11 session objects.

Value

Indication

YES

Indicates ICSF should audit lifecycle events related to PKCS #11 session objects. An SMF type 82 subtype 42 record is logged for each event.

NO

No lifecycle auditing of PKCS #11 session objects occurs.

If the AUDITKEYLIFETKDS option is not specified, the default is AUDITKEYLIFETKDS (TOKENOBJ(NO),SESSIONOBJ(NO)).

For more information about the events that are audited as well as the information contained in the audit record, see Appendix B in *z/OS Cryptographic Services ICSF System Programmer's Guide* for the description for the subtype 42 record.

The auditing of key lifecycle events can also be controlled via the SETICSF operator command. See the description of the SETICSF command in *z/OS Cryptographic Services ICSF System Programmer's Guide* for more information.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

AUDITKEYUSGCKDS(TOKEN(YES or NO),LABEL(YES or NO),INTERVAL(n))

Provides a set of options that control auditing of events related to the usage of symmetric CCA tokens. The audit logs are in the form of Type 82 SMF records.

TOKEN(YES or NO)

Controls usage auditing of CKDS tokens.

Value

Indication

YES

Indicates ICSF should audit usage events related to CKDS tokens. An SMF type 82 subtype 44 record is logged for each event.

NO

No usage auditing of CKDS tokens occurs.

LABEL(YES or NO)

Controls usage auditing of CKDS labels.

Value

Indication

YES

Indicates ICSF should audit usage events related to CKDS labels. An SMF type 82 subtype 44 record is logged for each event.

NO

No usage auditing of CKDS labels occurs.

INTERVAL(n)

Defines the time interval over which the audit records are aggregated. Specify *n* as a decimal value in hours from 1 through 24.

If the AUDITKEYUSGCKDS option is not specified, the default is AUDITKEYUSGCKDS(TOKEN(NO),LABEL(NO),INTERVAL(24)).

Note: An event that involves a token is considered to be any request that uses a token as opposed to a label. This is true regardless of Key Store Policy enablement.

For more information about the information contained in the audit record, see Appendix B in *z/OS Cryptographic Services ICSF System Programmer's Guide* for the description for the subtype 44 record.

The auditing of key usage events can also be controlled via the SETICSF operator command. See the description of the SETICSF command in *z/OS Cryptographic Services ICSF System Programmer's Guide* for more information.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

AUDITKEYUSGPKDS(TOKEN(YES or NO),LABEL(YES or NO),INTERVAL(n))

Provides a set of options that control auditing of events related to the usage of asymmetric CCA tokens. The audit logs are in the form of Type 82 SMF records.

TOKEN(YES or NO)

Controls usage auditing of PKDS tokens.

Value

Indication

YES

Indicates ICSF should audit usage events related to PKDS tokens. An SMF type 82 subtype 45 record is logged for each event.

NO

No usage auditing of PKDS tokens occurs.

LABEL(YES or NO)

Controls usage auditing of PKDS labels.

Value

Indication

YES

Indicates ICSF should audit usage events related to PKDS labels. An SMF type 82 subtype 45 record is logged for each event.

NO

No usage auditing of PKDS labels occurs.

INTERVAL(n)

Defines the time interval over which the audit records are aggregated. Specify *n* as a decimal value in hours from 1 through 24.

If the AUDITKEYUSGPKDS option is not specified, the default is AUDITKEYUSGPKDS(TOKEN(NO),LABEL(NO),INTERVAL(24)).

Note: An event that involves a token is considered to be any request that uses a token as opposed to a label. This is true regardless of Key Store Policy enablement.

For more information about the information contained in the audit record, see Appendix B in *z/OS Cryptographic Services ICSF System Programmer's Guide* for the description for the subtype 45 record.

The auditing of key usage events can also be controlled via the SETICSF operator command. See the description of the SETICSF command in *z/OS Cryptographic Services ICSF System Programmer's Guide* for more information.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

AUDITPKCS11USG(TOKENOBJ(YES or NO),SESSIONOBJ(YES or NO),NOKEY(YES or NO),INTERVAL(n))

Provides a set of options that control auditing of usage events related to PKCS #11 services. The audit logs are in the form of Type 82 SMF records.

TOKEN(YES or NO)

Controls usage auditing of PKCS #11 token objects.

Value

Indication

YES

Indicates ICSF should audit usage events related to PKCS #11 token objects. An SMF type 82 subtype 46 record is logged for each event.

NO

No usage auditing of PKCS #11 token objects occurs.

SESSIONOBJ(YES or NO)

Controls usage auditing of PKCS #11 session objects.

Value

Indication

YES

Indicates ICSF should audit usage events related to PKCS #11 session objects. An SMF type 82 subtype 46 record is logged for each event.

NO

No usage auditing of PKCS #11 session objects occurs.

NOKEY(YES or NO)

Controls usage auditing of PKCS #11 services that do not involve an object.

Value

Indication

YES

Indicates ICSF should audit relevant usages that do not pertain to a PKCS #11 object. Relevant usages include use of the PKCS #11 One-way hash, sign, or verify (CSFPFRF) and PKCS #11 Pseudo-random function (CSFPOWH) services. An SMF type 82 subtype 47 record is logged for each event.

NO

No usage auditing of PKCS #11 services that do not involve an object occurs.

INTERVAL(n)

Defines the time interval over which the audit records are aggregated. Specify *n* as a decimal value in hours from 1 through 24.

If the AUDITPKCS11USG option is not specified, the default is AUDITPKCS11USG(TOKENOBJ(NO),SESSIONOBJ(NO),NOKEY(NO), INTERVAL(24)).

For more information about the information contained in the audit record, see Appendix B in [z/OS Cryptographic Services ICSF System Programmer's Guide](#) for the description for the subtypes 46 and 47 records.

The auditing of key usage events can also be controlled via the SETICSF operator command. See the description of the SETICSF command in [z/OS Cryptographic Services ICSF System Programmer's Guide](#) for more information.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

BEGIN(fmid)

Specifies that parameters following this BEGIN parameter are supported in release *fmid* and later. There must be an END statement to complete the current section. If not, an error message will be issued and ICSF will terminate.

There may be any number of BEGIN/END pairs in the data set, but they cannot be nested within each other. A BEGIN must have a matching END before another BEGIN can be specified.

If the release of ICSF you are running is at this release or later, the parameters will be parsed and processed. If release of ICSF you are running is an earlier release, the parameters will be ignored.

It recommended that when your systems are all running releases that support newer parameters that the BEGIN/END pair be removed.

The following FMIDs are supported: HCR7740, HCR7750, HCR7751, HCR7770, HCR7780, HCR7790, HCR77A0, HCR77A1, HCR77B0, HCR77B1, and HCR77C0.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

Here is an example of the usage of the BEGIN/END parameters.

```
parameter4      /* parameter4 is supported by all releases */
BEGIN(HCR7751)
parameter1      /* parameter1 added in HCR7751 */
parameter3      /* parameter3 added in HCR7751 */
END
BEGIN(HCR7770)
parameter2      /* parameter2 added in HCR7770 */
```

```
END
parameter5      /* parameter5 is supported by all releases */
```

CHECKAUTH(YES or NO)

Indicates whether ICSF performs security access control checking of Supervisor State or System Key callers. If you specify CHECKAUTH(YES), ICSF issues RACROUTE calls to perform the security access control checking and the results are logged in RACF SMF records that are cut by RACF. If you specify CHECKAUTH(NO), the authorization checks against resources in the CSFSERV, CSFKEYS, and XCSFKEY classes are not performed.

If you do not specify the CHECKAUTH option, the default is CHECKAUTH(NO).

If you configure CHECKAUTH(YES) in the ICSF options dataset, the Health Checker address space user identity must be authorized to the CSFRKL profile in class CSFSERV for the ICSFMIG7731_ICSF_RETAINED_RSAKEY migration check to successfully execute. However, you have no action to take if you choose not to run the migration check. If you configure CHECKAUTH(NO), there is no requirement to authorize the Health Checker user identity for any ICSF profiles or classes, since the check routine executes in supervisor state. This is not an implementation consideration, but rather a check deployment or activation time customer administration consideration.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

CKDSN(data-set-name)

Specifies the CKDS name the system uses to start ICSF. Whenever you restart ICSF, the CKDS named in the CKDSN option becomes the active in-storage CKDS. (At first-time startup, you should specify the name of an empty VSAM data set you created to use as the CKDS.)

If you do not specify this keyword, you will not be able to use secure CCA symmetric keys or use ICSF to manage CCA symmetric keys. ICSF must be restarted in order to switch between having a CKDS and not having a CKDS.

See [“Steps to create the installation options data set” on page 21](#) for the data set naming format requirements.

CKTAUTH(YES or NO)

This keyword is no longer supported, but is tolerated.

COMPAT(YES, NO, or COEXIST)

Indicates whether ICSF runs in compatibility mode, non-compatibility mode, or coexistence mode with PCF.

YES

Indicates **compatibility mode**.

In compatibility mode, you can run a PCF application on ICSF because ICSF supports the PCF macros. You do not have to reassemble PCF applications to do this. You cannot start PCF at the same time as ICSF on the same operating system.

NO

Indicates **non-compatibility mode**. In noncompatibility mode, you run PCF applications on PCF and ICSF applications on ICSF. You cannot run PCF applications on ICSF because ICSF does not support the PCF macros in this mode. PCF can be started at the same time as ICSF on the same operating system. You can start ICSF and then start PCF, or you can start PCF and then start ICSF.

You should use noncompatibility mode unless you are migrating from PCF to ICSF.

COEXIST

Indicates **coexistence mode**.

In coexistence mode, you can run a PCF application on PCF, or you can reassemble the PCF application to run on ICSF. To do this, you reassemble the application against coexistence macros that are shipped with ICSF. You can start PCF at the same time as ICSF on the same operating system.

If you do not specify the COMPAT option, the default value is COMPAT(NO). See [“Running PCF and z/OS ICSF on the same system”](#) on page 199 for a complete description of the COMPAT options.

When you initialize ICSF for the first time, noncompatibility mode must be active. Therefore, at first-time startup, you must specify COMPAT(NO)

or allow the default to be used.

COMPENC(DES or CDMF)

This keyword is no longer supported, but is tolerated.

CTRACE(CTICSFxx)

Specifies the CTICSFxx ICSF CTRACE configuration data set to use from PARMLIB. CTICSF00 is the default ICSF CTRACE configuration data set that is installed with ICSF FMID HCR77A1 and later releases. CTICSF00 may be copied to create new PARMLIB members using the naming convention of CTICSFxx, where xx is a unique value specified by the user.

This parameter is optional. If the specified PARMLIB member is incorrect or absent, ICSF CTRACE will attempt to use the default CTICSF00 PARMLIB member. If the CTICSF00 PARMLIB member is incorrect or absent, ICSF CTRACE will perform tracing using an internal default set of trace options. By default, ICSF CTRACE support will trace with the KdsIO, CardIO, and SysCall filters using a 2M buffer. For more information refer to [“Creating an ICSF CTRACE configuration data set”](#) on page 23s.

DEFAULTWRAP(internal_wrapping_method,external_wrapping_method)

Specifies the default key wrapping for DES keys. Any token generated or updated by a service will be wrapped using the specified method unless overridden by rule array keyword or a skeleton token. The default wrapping method for internal and external tokens is specified independently.

Valid values for *internal_wrapping_method* and *external_wrapping_method* are:

ORIGINAL

Specifies the original CCA token wrapping be used: ECB wrapping for DES.

ENHANCED

Specifies the new X9.24 compliant CBC wrapping is used. The enhanced wrapping method is available only on IBM zEnterprise 196, IBM zEnterprise 114 and newer servers.

If the DEFAULTWRAP parameter is not specified, the default wrapping method is ORIGINAL for both internal and external tokens.

During initialization, ICSF changes the setting of the default wrapping method for all CCA coprocessors to match the value that is specified by this parameter.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

DOMAIN(n)

Specifies the number of the domain that you want to use for this start of ICSF. You can specify only one domain in the options data set. The domain value must match the activation profile.

DOMAIN is an optional parameter. The DOMAIN parameter is only required if more than one domain is specified as the usage domain on the PR/SM panels. If specified in the options data set, it will be used and it must be one of the usage domains for the LPAR.

If DOMAIN is not specified in the options data set, ICSF determines which domains are available in this LPAR. If only one domain is defined for the LPAR, ICSF will use it. If more than one is available, ICSF will issue error message CSFM409E.

The cryptographic processors support multiple sets of master key registers, which the specific domain values identify.

- The PCIXCC/CEX2C has master key registers for the DES-MK, AES-MK and RSA-MK master keys. Each domain has a master key register for the current, new, and old DES-MK, AES-MK and RSA-MK.
- CCA cryptographic coprocessors that are CEX3C or later have master key registers for the DES-MK, AES-MK, RSA-MK, and ECC-MK master keys. Each domain has a master key for the current, new, and old DES-MK, AES-MK, RSA-MK, and ECC-MK.

- The PKCS #11 cryptographic coprocessors have master key registers for the P11-MK master key. Each domain has a master key for the current and new P11-MK.

Note: The domain number that ICSF uses has no meaning for regional cryptographic servers. Regional cryptographic servers use the port number to identify the master key register to use.

For more information about partitions and running different configurations, see [z/OS Cryptographic Services ICSF Overview](#).

If you run ICSF in compatibility or coexistence mode, you cannot change the domain number without re-IPLing the system. A re-IPL ensures that a program does not access a cryptographic service with a key that is encrypted under a different master key. If you are certain that no cryptographic applications are still running, you can:

1. Stop CSF
2. Start CSF in COMPAT(NO) mode with a different domain number
3. Stop CSF
4. Start CSF in compatibility or coexistence mode with a different domain number.

END

Specifies the end of a section of parameters for the *fmid* from the **BEGIN(fmid)**. There must be a **BEGIN(fmid)** prior to the END. There must be an END for each **BEGIN(fmid)**. See the description for BEGIN for an example of the usage of the BEGIN and END parameters.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

EXIT(ICSF-name,load-module-name,FAIL(fail-option))

Indicates information about an installation exit.

The ICSF *-name* is the identifier for each exit. [Table 3 on page 38](#) lists all the ICSF exit names and explains when ICSF calls each exit. The load module name is the name of the module that contains the exit. The name can be any valid name your installation chooses.

Using the FAIL keyword of the EXIT statement, you specify the action ICSF, the KGUP, or the PCF conversion program takes if the exit ends abnormally. The fail action that you specify applies to subsequent calls of the exit. If an exit ends abnormally, ICSF takes a system dump. The exit is protected with an ESTAE or the ICSF service functional recovery routine (FRR).

In general, you can specify one of these values for a fail option:

NONE

No action is taken. The exit can be called again and will end abnormally again.

EXIT

The exit is no longer available to be called again.

SERVICE

The service or program that called the exit is no longer available to be called again.

ICSF

ICSF or the key generator utility program or the PCF conversion program is ended, depending on the exit.

Some fail options are not valid for a specific exit. If you specify a fail option that is not valid, ICSF uses the next valid fail option. For example, if SERVICE is not a valid fail option for an exit, ICSF uses the EXIT option. EXIT is responsible for logging in SMF the results of any authorization checks that are made.

Table 3. Exit identifiers and exit invocations	
Exit identifiers	Exit invocations
CSFAPG	Gets control during the Authentication Parameter Generate callable service.

Table 3. Exit identifiers and exit invocations (continued)

Exit identifiers	Exit invocations
CSFCKC	Gets control during the CVV key combine callable service.
CSFCKDS	Gets control when a callable service retrieves an entry from the CKDS.
CSFCKI	Gets control during the clear key import callable service.
CSFCKM	Gets control during the multiple clear key import callable service.
CSFCONVX	Gets control when you run the PCF CKDS conversion program.
CSFCPA	Gets control during the clear PIN generate alternate callable service.
CSFCPE	Gets control during the clear PIN encrypt callable service.
CSFCSG	Gets control during the VISA CVV service generate callable service.
CSFCSV	Gets control during the VISA CVV service verify callable service.
CSFCTT2	Gets control during the cipher text translate2 service.
CSFCTT3	Gets control during the cipher text translate2 (with alet) service.
CSFCVE	Gets control during the cryptographic variable encipher callable service.
CSFCVT	Gets control during the control vector translate callable service.
CSFDCM	Gets control during the Derive ICC MK callable service.
CSFDCO	Gets control during the decode callable service.
CSFDDPG	Gets control during the DK Deterministic PIN Generate callable service.
CSFDEC	Gets control during the decipher callable service.
CSFDEC1	Gets control during the decipher (with ALET) callable service.
CSFDKG	Gets control during the diversified key generate callable service.
CSFDKG2	Gets control during the Diversified Key Generate2 callable service.
CSFDKX	Gets control during the data key export callable service.
CSFDKM	Gets control during the data key import callable service.
CSFDMP	Gets control during the DK Migrate PIN callable service.
CSFDPC	Gets control during the DK PIN Change callable service.
CSFDPCG	Gets control during the DK PRW CMAC Generate callable service.
CSFDPMT	Gets control during the DK PAN Modify in Transaction callable service.
CSFDPNU	Gets control during the DK PRW Card Number Update callable service.
CSFDPT	Gets control during the DK PAN Translate callable service.
CSFDRP	Gets control during the DK Regenerate PRW callable service.
CSFDPV	Gets control during the DK PIN Verify callable service.
CSFDRPG	Gets control during the DK Random PIN Generate callable service.
CSFDSG	Gets control during the digital signature generate service.
CSFDSK	Gets control during the Derive Session Key callable service.
CSFDSV	Gets control during the digital signature verify callable service.

Table 3. Exit identifiers and exit invocations (continued)	
Exit identifiers	Exit invocations
CSFEAC	Gets control during the EMV Transaction Service callable service.
CSFECO	Gets control during the encode callable service.
CSFEDC	Gets control during the compatibility service for the PCF CIPHER macro.
CSFEDH	Gets control during the ECC Diffie-Hellman callable service.
CSFEMK	Gets control during the compatibility service for the PCF EMK macro.
CSFENC	Gets control during the encipher callable service.
CSFENC1	Gets control during the encipher (with ALET) callable service.
CSFEPG	Gets control during the encrypted PIN generate callable service.
CSFESC	Gets control during the EMV Scripting Service callable service.
CSFEVF	Gets control during the EMV Verification Functions callable service.
CSFEXIT1	Gets control after the operator issues the START command, but before processing takes place. Note: You must not specify an EXIT statement for the first mainline exit, CSFEXIT1.
CSFEXIT2	Gets control after ICSF reads and interprets the installation options data set.
CSFEXIT3	Gets control before ICSF completes initialization.
CSFEXIT4	Gets control after the operator issues the STOP command to stop ICSF.
CSFEXIT5	Gets control when the operator issues the MODIFY command to modify ICSF.
CSFFPED	Gets control during the FPE decipher callable service.
CSFFPEE	Gets control during the FPE encipher callable service.
CSFFPET	Gets control during the FPE translate callable service.
CSFGIM	Gets control during the Generate Issuer MK callable service.
CSFGKC	Gets control during the compatibility service for the PCF GENKEY macro.
CSFHMG	Gets control during the HMAC generate callable service.
CSFHMV	Gets control during the HMAC Verify callable service.
CSFKDSL	Gets control during the Key Data Set List callable service.
CSFKDMR	Gets control during the Key Data Set Metadata Read callable service.
CSFKDMW	Gets control during the Key Data Set Metadata Write callable service.
CSFKET	Gets control during the Key Encryption Translate callable service.
CSFKEX	Gets control during the key export callable service.
CSFKGN	Gets control during the key generate callable service.
CSFKGN2	Gets control during the key generate2 callable service.
CSFKGUP	Gets control during key generator utility program initialization, processing, and termination.
CSFKIM	Gets control during the key import callable service.

<i>Table 3. Exit identifiers and exit invocations (continued)</i>	
Exit identifiers	Exit invocations
CSFKPI	Gets control during the key part import callable service.
CSFKPI2	Gets control during the key part import2 callable service.
CSFKRC	Gets control during the CKDS key record create callable service.
CSFKRC2	Gets control during the CKDS key record create2 callable service.
CSFKRD	Gets control during the CKDS key record delete callable service.
CSFKRR	Gets control during the CKDS key record read callable service.
CSFKRR2	Gets control during the CKDS key record read2 callable service.
CSFKRW	Gets control during the CKDS key record write callable service.
CSFKRW2	Gets control during the CKDS key record write2 callable service.
CSFKTR	Gets control during the key translate callable service.
CSFKTR2	Gets control during the key translate2 callable service.
CSFKYT	Gets control during the key test callable service.
CSFKYT2	Gets control during the key test2 callable service.
CSFKYTX	Gets control during the key test extended callable service.
CSFMDG	Gets control during the MDC generate callable service.
CSFMDG1	Gets control during the MDC generate (with ALET) callable service.
CSFMGN	Gets control during the MAC generate callable service.
CSFMGN1	Gets control during the MAC generate (with ALET) callable service.
CSFMGN2	Gets control during the MAC Generate2 callable service.
CSFMGN3	Gets control during the MAC Generate3 callable service.
CSFMPS	Gets control during the ICSF Multi-Purpose Service.
CSFMVR	Gets control during the MAC verify callable service.
CSFMVR1	Gets control during the MAC verify (with ALET) callable service.
CSFMVR2	Gets control during the MAC Verify2 callable service.
CSFMVR3	Gets control during the MAC Verify3 callable service.
CSFPGN	Gets control during the Clear PIN generate callable service.
CSFPRR2	Gets control during the PKDS Key Record Read2 callable service.
CSFPTR	Gets control during the encrypted PIN translate callable service.
CSFPTR	Gets control during the Encrypted PIN Translate Enhanced callable service.
CSFPVR	Gets control during the encrypted PIN verify callable service.
CSFRTC	Gets control during the compatibility service for the PCF RETKEY macro.
CSFSKM	Gets control during the multiple secure key import callable service.
CSFSRRW	Gets control when an access to a single record in the CKDS is made using the key entry hardware.
CSFOWH	Gets control during the one-way hash generate callable service.

<i>Table 3. Exit identifiers and exit invocations (continued)</i>	
Exit identifiers	Exit invocations
CSFOWH1	Gets control during the one-way hash generate (with ALET) callable service.
CSFPCI	Gets control during the PCI interface callable service.
CSFPCU	Gets control during the PIN Change/Unblock callable service.
CSFPEX	Gets control during the prohibit export callable service.
CSFPEXX	Gets control during the prohibit export extended callable service.
CSFPFO	Gets control during the Recover PIN From Offset callable service.
CSFPKD	Gets control during the PKA decrypt callable service.
CSFPKE	Gets control during the PKA encrypt callable service.
CSFPKG	Gets control during the PKA key generate callable service.
CSFPKI	Gets control during the PKA key import callable service.
CSFPKT	Gets control during the PKA key translate callable service.
CSFPKTC	Gets control during the PKA key token change callable service.
CSFPKRC	Gets control during the PKDS key record create callable service.
CSFPKRD	Gets control during the PKDS key record delete callable service.
CSFPKRR	Gets control during the PKDS key record read callable service.
CSFPKRW	Gets control during the PKDS key record write callable service.
CSFPKX	Gets control during the PKA Public Key Extract callable service.
CSFRKA	Gets control during the restrict key attribute callable service.
CSFRKD	Gets control during the retained key delete callable service.
CSFRKL	Gets control during the retained key list callable service.
CSFRKX	Gets control during the remote key export callable service.
CSFRNG	Gets control during the random number generate callable service.
CSFRNGL	Gets control during the random number generate long callable service.
CSFSBC	Gets control during the SET block compose callable service.
CSFSBD	Gets control during the SET block decompose callable service.
CSFSKI	Gets control during the secure key import callable service.
CSFSKI2	Gets control during the secure key import2 callable service.
CSFSKY	Gets control during the secure messaging for keys callable service.
CSFSMG	Gets control during the symmetric MAC generate callable service.
CSFSMG1	Gets control during the symmetric MAC generate (with ALET) callable service.
CSFSMV	Gets control during the symmetric MAC verify callable service.
CSFSMV1	Gets control during the symmetric MAC verify (with ALET) callable service.
CSFSPN	Gets control during the secure messaging for PINs callable service.
CSFSXD	Gets control during the Symmetric Key Export with Data callable service.

Table 3. Exit identifiers and exit invocations (continued)	
Exit identifiers	Exit invocations
CSFSYG	Gets control during the symmetric key generate callable service.
CSFSYI	Gets control during the symmetric key import callable service.
CSFSYI2	Gets control during the symmetric key import2 callable service.
CSFSYX	Gets control during the symmetric key export callable service.
CSFT31I	Gets control during the TR-31 import callable service.
CSFT31X	Gets control during the TR-31 export callable service.
CSFTBC	Gets control during the trusted block create callable service.
CSFTRV	Gets control during the transaction validation callable service
CSFUKD	Gets control during the Unique Key Derive callable service

See Chapter 5, “Installation exits,” on page 131 for a detailed description of each ICSF exit, including the valid fail options.

Note: z/OS no longer ships IBM-supplied security exit routines; the security exit points remain. Users of z/OS should use Security Server (RACF) or an equivalent product to obtain access checking of services and keys. ICSF no longer needs these exit routines.

FIPSMODE(YES, FAIL(fail-option) or COMPAT, FAIL(fail-option) or NO,FAIL(fail-option))

Indicates whether z/OS PKCS #11 services must run in compliance with the Federal Information Processing Standard Security Requirements for Cryptographic Modules, referred to as FIPS 140-2. FIPS 140-2, published by the National Institute of Standards and Technology (NIST), is a standard that defines rules and restrictions for how cryptographic modules should protect sensitive or valuable information. The standard is available at [Security Requirements For Cryptographic Modules \(nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf\)](http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf).

By configuring z/OS PKCS #11 services to operate in compliance with FIPS 140-2 specifications, installations or individual applications can use the z/OS PKCS #11 services in a way that allows only the cryptographic algorithms (including key sizes) approved by the standard, and restricts access to the algorithms that are not approved. For more information, see [z/OS Cryptographic Services ICSF Writing PKCS #11 Applications](#).

YES

Indicates that the z/OS PKCS #11 services will operate in *FIPS standard mode*. Any application using the PKCS #11 services will be forced to use those services in a FIPS-compliant manner. Applications will not have access to the algorithms or key sizes not approved by FIPS 140-2. In addition, ICSF initialization will test that it is running on an IBM Z® model type, and a version and release of z/OS, that supports FIPS. If so, then ICSF will perform a series of cryptographic known answer tests as required by the FIPS 140-2 standard. If any of these initialization tests should fail, the action the ICSF initialization process takes will depend on the *fail-option* specified.

The *fail-option* is either YES or NO and indicates the action that the ICSF initialization process should take if any of the initialization tests should fail.

YES

Indicates ICSF is to terminate abnormally if there is a failure in any of the tests performed.

NO

Indicates ICSF initialization processing is to continue even if there is a failure in any of the tests performed. However, PKCS #11 support will be limited or nonexistent depending on the test that failed:

- If ICSF is running on an IBM Z model type or with a version of z/OS that does not support FIPS, most FIPS processing is bypassed. PKCS #11 callable services will be available, but ICSF will not adhere to FIPS 140 restrictions. Requests to generate or use a key with

CKA_IBM_FIPS140=TRUE or those requests that explicitly ask for FIPS processing will result in a failure return code.

- If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable.

COMPAT

Indicates that the z/OS PKCS #11 services will operate in *FIPS compatibility mode*. This mode is intended for installations where only certain z/OS PKCS #11 applications must comply with the FIPS 140-2 standard, while other applications do not. In this mode, the PKCS #11 services can be further configured so that the applications that do not need to comply with the FIPS 140-2 standard are not restricted from using any of the PKCS #11 algorithms, while applications that must comply with the standard are restricted from using the non-approved algorithms. By default, the COMPAT option will have the same effect as the YES option, and all applications using the PKCS #11 services will be forced to use those services in a FIPS-compliant manner. However, additional specifications can be made:

- at the PKCS #11 token and application level, by creating FIPSEXEMPT.token-label resource profiles in the CRYPTOZ class. A FIPSEXEMPT.token-label resource exists for each token. User IDs with READ access authority to a FIPSEXEMPT.token-label are exempt from FIPS compliance, while user IDs with access authority NONE can only use the PKCS #11 services in a FIPS-compliant manner.
- within applications themselves for individual keys. When an application creates a key, the application can specify that the key must be used in a FIPS 140-2 compliant fashion. The application can specify this by setting the Boolean key attribute CKA_IBM_FIPS140 to TRUE.

When the COMPAT option is specified, ICSF initialization will test that it is running on an IBM Z model type, and a version and release of z/OS, that supports FIPS. If so, then ICSF will perform a series of cryptographic known answer tests as required by the FIPS 140-2 standard. If any of these initialization tests should fail, the action the ICSF initialization process takes will depend on the *fail-option* specified.

The *fail-option* is either YES or NO and indicates the action that the ICSF initialization process should take if any of the initialization tests should fail.

YES

Indicates ICSF is to terminate abnormally if there is a failure in any of the tests performed.

NO

Indicates ICSF initialization processing is to continue even if there is a failure in any of the tests performed. However, PKCS #11 support will be limited or nonexistent depending on the test that failed:

- If ICSF is running on an IBM Z model type or with a version of z/OS that does not support FIPS, most FIPS processing is bypassed. PKCS #11 callable services will be available, but ICSF will not adhere to FIPS 140 restrictions. Requests to generate or use a key with CKA_IBM_FIPS140=TRUE or those requests that explicitly ask for FIPS processing will result in a failure return code.
- If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable.

NO

Indicates that ICSF should operate in FIPS no enforcement mode, also known as FIPS on-demand mode. Applications may request strict adherence to FIPS 140 restrictions when requesting ICSF services. However, applications not requesting FIPS processing are not required to adhere to FIPS 140 restrictions. FIPSEXEMPT.token-label profiles, if they exist in the CRYPTOZ class, will not be examined. If ICSF is running on an IBM Z model type that does not support FIPS, requests to generate or use a key with CKA_IBM_FIPS140=TRUE or those requests that explicitly ask for FIPS processing will result in a failure return code.

ICSF initialization will test that it is running on an IBM Z model type and version/release of z/OS that supports FIPS. If so, ICSF initialization will also perform a series of cryptographic known answer self tests. Should a test fail, the action ICSF initialization takes is dependent on the fail option.

The *fail-option* is either YES or NO and indicates the action that the ICSF initialization process should take if any of the initialization tests should fail.

YES

Indicates ICSF is to terminate abnormally if there is a failure in any of the tests performed.

NO

Indicates ICSF initialization processing is to continue even if there is a failure in any of the tests performed. However, PKCS #11 support will be limited or nonexistent depending on the test that failed:

- If ICSF is running on an IBM Z model type or with a version of z/OS that does not support FIPS, most FIPS processing is bypassed. PKCS #11 callable services will be available, but ICSF will not adhere to FIPS 140 restrictions. Requests to generate or use a key with CKA_IBM_FIPS140=TRUE or those requests that explicitly ask for FIPS processing will result in a failure return code.
- If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable.

If the FIPSMODE option is not specified, the default is FIPSMODE(NO, FAIL(NO)).

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

HDRDATE(YES or NO)

This keyword is no longer supported, but is tolerated.

KDSREFDAYS(n)

Specifies, in days, how often a record should be written for a reference date/time change. A key is referenced when it is used to perform a cryptographic operation. If a key is referenced ICSF will check the date and time the key was referenced previous to the current reference. If the number of days between the current date and time and the date and time the key was last referenced is greater than or equal to the number of days specified in the KDSREFDAYS installation option then the key reference date/time in the KDS will be updated to the current date and time. Otherwise the reference date/time will remain the same. Note, in this context days are 24 hour periods not necessarily beginning or ending at midnight.

For example: If KDSREFDAYS(7) was specified and a key was referenced on Monday, January 1st at 8 AM, and the reference date/time for the key was updated at that time, then any key reference before Monday, January 8th at 8 AM (7 days) will not update the reference date/time in the key record. If the key is referenced again at 7:50 AM on Monday, January 8th, the reference date/time for the key in the KDS will remain January 1st at 8 AM because fewer than seven days have passed. The reference date/time will not be updated until the next time the key is used again Monday, January 8th at 8 AM or after.

KDSREFDAYS applies to all KDS that are in the format that supports key reference tracking. In an environment of mixed KDS formats, where some support reference date tracking and some do not (for example, the CKDS supports reference date tracking, but the PKDS does not) key references will not be tracked for keys in a KDS does not support it, regardless on the value of KDSREFDAYS, until that KDS is updated to the new format. In a SYSPLEX, all systems must be started with the same value of KDSREFDAYS to ensure proper tracking of reference date/times.

KDSREFDAYS(0) means that ICSF will not keep track of key reference dates. The default is KDSREFDAYS(1). The maximum value allowed is KDSREFDAYS(30).

Note: Updates to records using the Key Generator Utility Program (KGUP) are not subject to the value specified in the KDSREFDAYS option. All updates made via KGUP will update the reference date/time if the CKDS is in a format that supports reference date tracking (KDSR).

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

KEYARCHMSG(YES or NO)

Controls whether a joblog message is issued when an application successfully references a key data set record that has been archived. The message is only issued for the first successful reference of a record. The results of the service request is not affected by this control. The default is NO.

**Value
Indication****YES**

ICSF issues a message the first time an archived record is referenced by an application.

NO

ICSF does not issue a message when an archived record is referenced by an application.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

KEYAUTH(YES or NO)

This keyword is no longer supported, but is tolerated.

MASTERKCVLEN(2 or 3 or 4 or 5 or 6 or ALL)

Defines the number of hexadecimal digits to display on the ICSF Coprocessor Hardware Status panel (CSFCMP40) for the verification and hash patterns for the master keys. The patterns are also referred to as key check values. When an integer value is specified, that number of digits will be displayed. When ALL is specified, all the digits will be displayed.

The default is ALL.

This option can be used for compliance with the ISO11568 standard for the display of the key check values for master keys.

Note: This option has no affect on the output of the DISPLAY ICSF,MKS command.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

MAXLEN(n)

Defines the maximum length of characters in a text string, including any necessary padding, for some callable service requests. For example, this option defines the maximum length of the text the encipher service encrypts for each call. Specify *n* as a decimal value from 1024 through 2147483647. If you do not specify the MAXLEN option, the default value is MAXLEN(65535).

The MAXLEN parameter may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647). If a value greater than this is specified, an error will result and ICSF will not start.

Note: MAXLEN is no longer displayed on the Installation Option Display panel.

MAXSESSOBJECTS(n)

Defines the maximum number of PKCS #11 session objects and states an unauthorized (problem state, non-system key) application may own at any one time. Specify *n* as a decimal value from 1024 through 2147483647. If you do not specify the MAXSESSOBJECTS option, the default value is MAXSESSOBJECTS(65535).

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

PKDSCACHE

This keyword is no longer supported, but is tolerated.

PKDSN(data-set-name)

Specifies the PKDS name the system uses to start ICSF. Whenever you restart ICSF, the PKDS named in the PKDSN option becomes the active PKDS. (At first-time startup, you should specify the name of an empty VSAM data set you created to use as the PKDS.)

If you do not specify this keyword, you will not be able to use secure CCA asymmetric keys or use ICSF to manage CCA asymmetric keys. ICSF must be restarted in order to switch between having a PKDS and not having a PKDS.

See [“Steps to create the installation options data set” on page 21](#) for the data set naming format requirements.

REASONCODES(ICSF or TSS)

Specifies which set of reason codes are to be returned from callable services. If you do not specify the REASONCODES option, the default of REASONCODES(ICSF) is used. If you specify REASONCODES(TSS), reason codes used by the IBM 4765 PCIe, IBM 4767 PCIe, and IBM 4764 PCI-X cryptographic coprocessors will be returned. If there is a 1-to-1 mapping, the codes will be converted.

If you specified REASONCODES(ICSF) and your service was processed on a CCA coprocessor, a cryptographic coprocessor reason code may be returned if there is no corresponding ICSF reason code.

REMOTEDEVICE(index-number, ip-addr-or-hostname, port-number, number-sockets)

Specifies the connection information for a remote regional cryptographic server device that ICSF is to use for regional cryptographic server requests. There may be up to 16 of these entries.

Notes:

- Each regional cryptographic server (as identified by *ip-addr-or-hostname* and *port-number*) must be configured identically regarding master keys and other settings. An incorrect master key value will cause the connection to not be used.
- For use of standalone, network-attached regional cryptographic servers, IBM zEnterprise EC12 or later hardware is required as well as servers running z/OS V1R13 or later and ICSF FMID HCR77B1 or later.
- For use of Linux LPAR regional cryptographic servers, IBM z13 or later hardware is required as well as servers running z/OS V1R13 or later and ICSF FMID HCR77B1 or later.

The options are as follows:

index-number

Specify a number between 1 and 16, inclusive. Each operational REMOTEDEVICE must have a unique number. For indexes that are repeated, ICSF will only save the last one specified. Additionally, if remote devices are shared between sysplex members, it is strongly recommended that the same index number is used for each member. This simplifies remote device management using the SETICSF operator command.

ip-addr-or-hostname

Specify either the dotted-decimal Internet protocol (IP) version 4 address or the hostname of the remote device. Each *ip-addr-or-hostname* must locate a single device with fixed serial number. Reverse proxy arrangements where one *ip-addr-or-hostname* is backed by multiple devices (with different serial numbers) is not supported. The opposite arrangement (one serial number assigned to multiple *ip-addr-or-hostnames*) is supported, but not recommended.

Notes:

- Internet protocol (IP) version 6 is not supported.
- Hostnames are not case-sensitive and are stored and displayed by ICSF in lowercase.
- For long hostnames, the REMOTEDEVICE entry may be split at any comma to span multiple physical records. For example:

```
REMOTEDEVICE(5,some.very.long.hostname.company.com,  
6901,8)
```

port-number

Specify the port number to be used in conjunction with the IP address or hostname when connecting.

Note: No two ICSF instances may share the same port on a regional cryptographic server. Additionally, it is expected that different workloads (for example, ICSF instances using different token data sets) sharing a regional cryptographic server would use different master keys (RCS-MKs) and that the required RCS-MK for the TKDS would be assigned on a per port basis.

number-sockets

Specify the maximum number of sockets ICSF is to open for connections with the remote device. This is a value between 1 and 8, inclusive. Multiple sockets are required in order for ICSF to process multiple simultaneous requests. Consult the remote device's documentation to determine this value. There is an ICSF limit of 8 sockets per server or port. If you desire more than 8 socket connections to a single server, define multiple REMOTEDEVICE entries for the server, assigning a unique port number for each entry. Make sure the same master key is defined for each port that will be connected to systems sharing the same TKDS.

RNGCACHE(YES or NO)

Indicates whether ICSF should maintain a cache of random numbers to be used by services that require them. When YES is specified for this option, a noticeable performance improvement may be realized by workloads requesting a significant amount of random data.

If you do not specify the RNGCACHE option, the default value is RNGCACHE(YES).

Value

Indication

YES

ICSF maintains a random number cache.

NO

ICSF does not maintain a random number cache.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

SERVICE(service-number,load-module-name,FAIL(fail-option))

Indicates information about an installation-defined service.

ICSF allows an installation to define its own service similar to an ICSF callable service. The *service-number* specifies a number that identifies the service to ICSF. The valid service numbers are 1 through 32767, inclusive. This set of service numbers is valid for both installation-defined services and UDX services. The service number of an installation-defined service must not be the same as the service number of a UDX service. The *load-module-name* is the name of the module that contains the service. During ICSF startup, ICSF loads this module and binds it to the *service-number* you specified.

The *fail-option* is YES or NO, indicating the action ICSF should take if loading the service ends abnormally.

YES

Specifies that ICSF ends abnormally if your service cannot be loaded.

NO

Specifies that ICSF continues to start if your service cannot be loaded.

If the service itself ends abnormally, ICSF does not end, but takes a system dump instead. The ICSF service functional recovery routine (FRR) protects the service.

See Chapter 6, “Installation-defined Callable Services,” on page 185 for a description of how to write and run an installation-defined callable service.

SSM(YES or NO)

Specifies whether or not an installation can enable special secure mode (SSM) while running ICSF. SSM lowers the security of your system to let you enter clear keys and generate clear PINs. You must enable SSM for KGUP to permit generation or entry of clear keys and to enable the secure key import, secure key import2, multiple secure key import, or clear pin generate callable services.

YES

Indicates that you can enable the SSM.

NO

Indicates that you cannot enable the SSM.

If you do not specify the SSM option, the default value is SSM(NO).

The SSM option can be changed from NO to YES while ICSF is running by defining the CSF.SSM.ENABLE SAF profile within the XFACILIT resource class. To revert to your startup option, delete the CSF.SSM.ENABLE profile. The XFACILIT class must be refreshed after each change for it to take effect.

Note: When using the SAF profiles to set the SSM, all ICSF instances sharing the SAF database will be affected.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

When the CSF.SSM.ENABLE SAF profile is defined within the XFACILIT resource class, attempts to update the SSM option using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6) will be ignored. The SSM option value will be saved and used should the CSF.SSM.ENABLE SAF profile ever be deleted.

SYSplexCKDS(YES or NO,FAIL(fail-option))

SYSplexCKDS(YES,FAIL(fail-option))

ICSF will join the ICSF sysplex group SYSICSF and this system will participate in sysplex-wide consistency for CKDS data.

SYSplexCKDS(YES,FAIL(YES))

Indicates ICSF initialization will end abnormally if the ICSF cross-system services environment cannot be established during ICSF initialization due to a failure creating the CKDS latch set or a failure to join the ICSF sysplex group.

SYSplexCKDS(YES,FAIL(NO))

Indicates ICSF initialization processing will continue even if the request to join the ICSF sysplex group fails. The system will not be notified of updates to the CKDS by other members of the ICSF sysplex group. A value of either FAIL(YES) or FAIL(NO) will be ignored with SYSplexCKDS(NO,...).

SYSplexCKDS(NO,FAIL(fail-option))

CKDS update processing proceeds as it does today (i.e. no Cross-System Services task will be initialized, nor will any XCF signalling be performed when an update to a CKDS record occurs).

If you do not specify the SYSplexCKDS option, the default value is SYSplexCKDS(NO,FAIL(NO)).

SYSplexPKDS(YES or NO,FAIL(fail-option))

ICSF will join the ICSF sysplex group SYSICSF and this system will participate in sysplex-wide consistency for PKDS data.

SYSplexPKDS(YES,FAIL(fail-option))

ICSF will join the ICSF sysplex group SYSICSFP and this system will participate in sysplex-wide consistency for PKDS data.

SYSplexPKDS(YES,FAIL(YES))

Indicates ICSF initialization will fail to join the sysplex if the ICSF cross-system services environment cannot be established during ICSF initialization due to a failure creating the PKDS latch set or a failure to join the ICSF sysplex group.

SYSplexPKDS(YES,FAIL(NO))

Indicates ICSF initialization processing will continue even if the request to join the ICSF sysplex group fails. The system will not be notified of updates to the PKDS by other members of the ICSF sysplex group. A value of either FAIL(YES) or FAIL(NO) will be ignored with SYSplexPKDS(NO,...).

SYSplexPKDS(NO,FAIL(fail-option))

PKDS update processing proceeds without trying to join the ICSF sysplex group.

If you do not specify the **SYSPLEXPBKDS** option, the default value is **SYSPLEXPBKDS(NO,FAIL(NO))**.
SYSPLEXTKDS(YES or NO,FAIL(fail-option))

ICSF will join the ICSF sysplex group SYSICSF and this system will participate in sysplex-wide consistency for TKDS data.

Note: TKDSN needs to be specified for this to work. See **TKDSN(data-set-name)**.

SYSPLEXTKDS(NO,FAIL(fail-option))

Indicates no XCF signalling will be performed when an update to a TKDS record occurs.

SYSPLEXTKDS(YES,FAIL(fail-option))

Indicates the system will be notified of updates made to the TKDS by other members of the sysplex who have also specified SYSPLEXTKDS(YES,FAIL(fail-option)).

SYSPLEXTKDS(YES,FAIL(YES))

Indicates ICSF will terminate abnormally if there is a failure creating the TKDS latch set.

SYSPLEXTKDS(YES,FAIL(NO))

Indicates ICSF initialization processing will continue even if the request to join the ICSF sysplex group fails. This system will not be notified of updates to the TKDS by other members of the ICSF sysplex group.

If you do not specify the SYSPLEXTKDS option, the default value is SYSPLEXTKDS(NO,FAIL(NO)) is the default.

TKDSN(data-set-name)

The name of an existing TKDS or an empty VSAM data set to be used as the TKDS. To enable applications to create and use persistent PKCS #11 tokens and objects that use the PKCS #11 services, this option must be specified.

See [“Steps to create the installation options data set” on page 21](#) for the data set naming format requirements.

TRACEENTRY(n)

This keyword is no longer supported, but is tolerated.

UDX(UDX-id,service-number,load-module-name,'comment_text',FAIL(fail-option))

ICSF allows the development of User Defined Extensions for the coprocessors. The *UDX-id* is supplied to the installation when the UDX is developed. The *service-number* specifies a number that identifies the service to ICSF. The valid service numbers are 1 to 32767, inclusive. This set of service numbers is valid for both installation-defined services and UDX services. The service number of a UDX service must not be the same as the service number of an installation-defined service. The *load-module-name* is the name of the module that contains this service. During ICSF startup, ICSF loads this module and binds it to the service-number that was specified. A *comment* may be specified. The positional parameter is required. The comment consists of up to 40 EBCDIC characters, and may include imbedded blank characters. The comment text is enclosed by single quotes. If no comment text is desired, two contiguous single quotes should be specified.

The *fail-option* is YES or NO, indicating the action ICSF should take if loading the service ends abnormally. If the service itself ends abnormally, ICSF does not end, but takes a system dump instead.

YES

Specifies that ICSF ends abnormally if your service cannot be loaded.

NO

Specifies that ICSF continues to start if your service cannot be loaded.

The User Defined Extension (UDX) is responsible for logging in SMF the results of any authorization checks that were made.

USERPARM(value)

Specifies an 8-byte field for installation use. The Installation Option Display panel displays this value, which is stored in the Cryptographic Communication Vector Table (CCVT) in the *CCVT_USERPARM*

field. An application program or installation exit can examine this field and use it to set system environment information. The default is eight blanks.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

WAITLIST(data_set_name)

This optional parameter can be used if you have ICSF with CICS installed. It specifies a customer modifiable data set will be used to determine names of the services to be placed into the ICSF CICS Wait List. A sample data set is provided by ICSF via member CSFWTL01 of SYS1.SAMPLIB. The sample data set contains the same entries as the default ICSF CICS Wait List (i.e., the data set contains the names of all ICSF callable services which, by default, will be driven through the CICS TRUE). Non-CICS customers will not need to specify the WAITLIST keyword. The WAITLIST option should be added to the Installation Options data set under these conditions.

- CICS customers who do not want to make use of CICS TRUE must either not enable the TRUE or must specify a WAITLIST keyword and point to an empty wait list data set (or specify WAITLIST(DUMMY)) in the Installation Options data set.
- CICS customers who wish to modify the ICSF default CICS Wait List should modify the sample Wait List data set supplied in member CSFWTL01 of SYS1.SAMPLIB. The WAITLIST keyword in the Installation Options Data Set should be set to point to this modified data set.

To ensure maximum performance, any existing CICS applications which invoke any of the ICSF services in the Wait List that were linked with ICSF stubs prior to HCR7770 should be re-linked with the current ICSF stubs. For additional information on the CICS Attachment Facility, see [Appendix C, “CICS-ICSF Attachment Facility,”](#) on page 377.

Starting with ICSF FMID HCR77C0, the value for this option can be updated without restarting ICSF by using either the SETICSF command or the ICSF Multi-Purpose service (CSFMPS or CSFMPS6).

Improving CKDS performance

To improve the performance of CKDS operations during KGUP runs, use the Batch Local Shared Resource (BLSR) with Deferred Write for all KGUP runs. See [z/OS in IBM Knowledge Center \(www.ibm.com/support/knowledgecenter/SSLTBW/welcome\)](http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome) for more information.

Dispatching priority of ICSF

To avoid performance problems, the dispatching priority of ICSF should be set at least as high as that of the highest task using ICSF.

Creating ICSF exits and generic services

You need not code any exits or generic services before using ICSF productively.

Developing callable service exits and generic services requires skill in assembler programming in a cross memory environment. To help with testing, the system programmer might want to use the WTO macro with the LINKAGE=BRANCH keyword to issue console messages while in cross-memory mode. (See [“Service exits”](#) on page 135 for more information.)

Chapter 3. Migration

This topic describes migration considerations.

Your plan for migrating to the new level of ICSF should include information from a variety of sources. These sources of information describe topics such as coexistence, service, hardware and software requirements, installation and migration procedures, and interface changes.



Attention: Although you are migrating to a new release, you should review the information in Chapter 2, “Installation, initialization, and customization,” on page 9; especially review customization steps that may have changed since your last migration.

If this migration also includes a hardware upgrade be sure to have your Master Keys available. Once Migration is complete, the Master Keys may need to be loaded and set. Review Chapter 2, “Installation, initialization, and customization,” on page 9 for information on setting Master Keys.

An IPL is required when installing a new release of ICSF (it is possible for ICSF control blocks like the DACC and CCVT to persist in storage across an ICSF restart).

Consult these documents for information on migration and installation:

z/OS Migration

This publication describes the migration tasks for z/OS at a system and element level.

This publication, which is supplied with your product order, provides information about installing your z/OS system. In addition to specific information about ICSF, this publication contains information about all of the z/OS elements. Consult the z/OS Migration publication for the release of z/OS running on your system.

z/OS Planning for Installation

This publication describes the installation requirements for z/OS at a system and element level. It includes hardware, software, and service requirements for both the driving and target systems. It also describes any coexistence considerations and actions.

Program Directory for Cryptographic Support for z/OS V1R13 - z/OS V2R2

This publication describes the program installation and maintenance requirements. It contains information about the material and procedures associated with the installation of ICSF.

The publications can be obtained from:

- The [Resource Link home page \(www.ibm.com/servers/resourcelink\)](http://www.ibm.com/servers/resourcelink). (Select Publications and then select the release that you are interested in under ICSF Publications by FMID.)
- IBM z/OS downloads (www.ibm.com/systems/z/os/zos/downloads) for Cryptographic Support for z/OS V1R13 - z/OS V2R2.

ServerPac Installing Your Order

This is the order-customized, installation publication for using the ServerPac Installation method. Be sure to review 'Appendix A. Product Information', which describes data sets supplied, jobs or procedures that have been completed for you, and product status. IBM may have run jobs or made updates to PARMLIB or other system control data sets. These updates could affect your migration.

Terminology

This topic describes some terms you may need to know as you use this publication.

Migration

Activities that relate to the installation of a new version or release of a program to replace a previous level. Completion of these activities ensures that the applications and resources on your system will function correctly at the new level.

Coexistence

Two or more systems at different levels (for example, software, service or operational levels) that share resources. Coexistence includes the ability of a system to respond in these ways to a new function that was introduced on another system with which it shares resources: ignore a new function, terminate gracefully, support a new function. These are examples of multisystem configurations in which resource sharing can occur:

- A single system running multiple LPARs
- A single processor that is time-sliced to run different levels of the system (for example, during different times of the day)
- Two or more systems running separate processors
- A Parallel Sysplex configuration (also includes a basic sysplex)

Migrating from earlier software releases

These topics describe common activities and considerations that should be considered when you migrate from an earlier release of ICSF to FMID HCR77C0.

Actions to perform before installing ICSF FMID HCR77C0

This topic describes migration actions that you can perform on your current (old) system. You do not need the ICSF FMID HCR77C0 level of code to make these changes, and the changes do not require the ICSF FMID HCR77C0 level of code to run once they are made.

Note: You may have already performed these migration actions if you previously migrated to ICSF FMIDs HCR77A1, HCR77B0, or HCR77B1.

ICSF: Detect any coprocessor that will not become active when ICSF FMID HCR77A1 or later is started

Description

For ICSF FMIDS HCR7780, HCR7790, and HCR77A0, the activation procedure was designed to maximize the number of active coprocessors by selecting the set of master keys that are available on the majority of coprocessors. A DES master key is no longer required in order for a coprocessor to become active. Instead, any one of four master keys – the DES master key, the AES master key, the RSA master key (which in earlier releases was called the asymmetric master key), or the ECC master key – is enough for a coprocessor to become active. However, because the goal is to select the combination of master keys that will maximize the number of active coprocessors, if a certain master key is not set on all the same coprocessors, that master key support will not be available.

Starting with FMID HCR77A1, the activation procedure now uses the master key verification patterns (MKVP) in the header record of the CKDS and PKDS to determine which coprocessors become active. If the MKVP of a master key is in the CKDS or PKDS, that master key must be loaded and the verification pattern of the current master key register must match the MKVP in the CKDS or PKDS. If all of the MKVPs in the CKDS and PKDS match the current master key registers, the coprocessor will become active. Otherwise, the status is master keys incorrect. This applies to all master keys that the coprocessor supports. When there is a MKVP in the CKDS or PKDS and the coprocessor does not support that master key, it is ignored. When a MKVP is not in the CKDS or PKDS, the master key is ignored.

If there are no MKVPs in the CKDS and PKDS, the coprocessor will be active. If the CKDS is initialized without any MKVPs, the CKDS cannot be used on a system that has cryptographic features installed.

Table 4 on page 55 provides more details about this migration action. Use this information to plan your changes to the system.

Table 4. Information about this migration action

Element or feature:	Cryptographic Services
When change was introduced:	Cryptographic Support for z/OS V1R13 - z/OS V2R1 web deliverable (FMID HCR77A1), which installs on z/OS V1R13 or z/OS V2R1.
Applies to migration from:	z/OS V2R1 and z/OS V1R13, both without the Cryptographic Support for z/OS V1R13 - z/OS V2R1 web deliverable (FMID HCR77A1) or a later ICSF web deliverable installed.
Timing:	Before installing FMID HCR77A1 or later ICSF FMIDs.
Is the migration action required?	Yes, if migrating from an ICSF FMID older than HCR77A1 to ICSF FMID HCR77A1 or later and if you are affected by the change in the way master keys are processed to determine which coprocessors become active.
Target system hardware requirements:	None.
Target system software requirements:	None.
Other system (coexistence or fallback) requirements:	None.
Restrictions:	None.
System impacts:	None.
Related IBM Health Checker for z/OS check:	Use check ICSFMIG77A1_COPROCESSOR_ACTIVE to determine which coprocessors will not become active when Cryptographic Support for z/OS V1R13 - z/OS V2R1 Web Deliverable (FMID HCR77A1) is started. This check is delivered in APAR OA42011 available for ICSF FMIDs HCR7770, HCR7780, HCR7790 and HCR77A0.

Steps to take

Run the migration check ICSFMIG77A1_COPROCESSOR_ACTIVE to find any coprocessors that will not become active when you start ICSF FMID HCR77A1 or a later ICSF web deliverable.

Reference information

For more information, see the following reference:

- For information about IBM Health Checker, see *IBM Health Checker for z/OS User's Guide*.

ICSF: Detect TKDS objects that are too large for the new KDSR record format in ICSF FMID HCR77A1 or later

Description

In ICSF FMID HCR77A1, ICSF added a common key data set record format for CCA key tokens and PKCS #11 tokens and objects. This new record format adds new fields for key utilization and metadata.

Because of the size of the new fields, some existing PKCS #11 objects in the TKDS might cause ICSF to fail. If you do not have a Token Data Set (TKDS) with PKDS #11 objects in it, there is no need to run this check.

The problem exists for TKDS object records with large objects. The `User` data field in the existing record will cause the TKDS not be loaded if the object size is greater than 32,520 bytes. The `TKDSREC_LEN` field in the record has the size of the object. If the `User` data field is not empty and the size of the object is greater than 32,520 bytes, the TKDS cannot be loaded.

Note that ICSF does not provide any interface to modify the `User` data field in the TKDS object record. A field can be created using IDCAMS. Check the contents of the `User` data field and determine if the information in the field is valuable. If you want to preserve the data, consider how the information can be stored other than in the object record. The field can only be modified by editing the record. For information about the TKDS object record, see “Token data set (TKDS) format” on page 221. The IBM Health Checker migration check, `ICSMIG77A1_TKDS_OBJECT` detects any TKDS object that is too large to allow the TKDS is read into storage during ICSF initialization starting with ICSF FMID HCR77A1. This migration check is available for ICSF FMIDs HCR7770, HR7780, HCR7790, and HCR77A0 through APAR OA42011.

Table 5 on page 56 provides more details about this migration action. Use this information to plan your changes to the system.

Table 5. Information about this migration action

Element or feature:	Cryptographic Services
When change was introduced:	Cryptographic Support for z/OS V1R13 – z/OS V2R1 web deliverable (FMID HCR77A1), which installs on z/OS V1R12, z/OS V1R13 or z/OS V2R1.
Applies to migration from:	z/OS V2R1 and z/OS V1R13, both without the Cryptographic Support for z/OS V1R13 - z/OS V2R1 web deliverable (FMID HCR77A1) installed.
Timing:	Before installing FMID HCR77A1 or later ICSF FMIDs.
Is the migration action required?	Yes, if migrating from an ICSF FMID older than HCR77A1 to ICSF FMID HCR77A1 or later and if you affected by the record format changes.
Target system hardware requirements:	None.
Target system software requirements:	None.
Other system (coexistence or fallback) requirements:	None.
Restrictions:	None.
System impacts:	None.
Related IBM Health Checker for z/OS check:	Use the IBM Health Checker migration check <code>ICSMIG77A1_TKDS_OBJECT</code> to detect any TKDS object with a value in the <code>User</code> data field that is too large to preserve in the <code>User</code> data field of the new format record. This migration check is available for FMIDs HCR7770, HR7780, HCR7790, and HCR77A0 through APAR OA42011.

Steps to take

Run the migration check ICSFMIG77A1_TKDS_OBJECT to detect if TKDS objects are too large for the new record format in FMID HCR77A1.

Note: ICSF does not provide any interface to modify the User data field in the TKDS object record. A flat file can be created using IDCAMS. Check the contents of the User data field and determine if the information in the field is valuable. If you want to preserve the data, consider how the information can be stored other than in the object record. The field can only be modified by editing the record. For information about the TKDS object record, see [“Token data set \(TKDS\) format”](#) on page 221.

Reference information

For more information, see the following references:

- For information about the TKDS object record, see [“Token data set \(TKDS\) format”](#) on page 221.
- For information about IBM Health Checker, see [IBM Health Checker for z/OS User's Guide](#).

Actions to perform before the first start of ICSF FMID HCR77C0

This topic describes migration actions that you can perform after you have installed ICSF FMID HCR77C0, but before the first time you start it. These actions might require the ICSF FMID HCR77C0 level of code to be installed, but does not require it to be started.

Note: You may have already performed these migration actions if you previously migrated to ICSF FMIDs HCR77A1, HCR77B0, or HCR77B1.

ICSF: Deprecated parameters in installation options data set

Description

The ICSF installation options data set parameters COMPENC and PKDSCACHE were deprecated in FMID HCR7751, parameters CKTAUTH, KEYAUTH, and TRACEENTRY were deprecated in FMID HCR77A1, and parameter HDRDATE was deprecated in FMID HCR77B1.

Table 6. Information about this migration action

Element or feature:	Cryptographic Services.
When change was introduced:	ICSF FMID HCR77B1.
Applies to migration from:	All ICSF FMIDs prior to FMID HCR77B1.
Timing:	Before the first start of FMID HCR77B1 or later ICSF FMIDs.
Is the migration action required?	Yes.
Target system hardware requirements:	None.
Target system software requirements:	None.
Other system (coexistence or fallback) requirements:	None.
Restrictions:	None.
System impacts:	None.
Related IBM Health Checker for z/OS check:	None.

Steps to take

Edit the ICSF installation options data set and remove all the deprecated parameters.

Note: ICSF will start with the deprecated parameters in the ICSF installation options data set, but the parameters are ignored and message CSFO0212 is issued for each deprecated parameter.

Reference information

For more information, see [“Customizing ICSF after the first start” on page 30.](#)

ICSF: Determine if applications using hash services have archived hashes of long data

Description

Due to service introduced by APAR OA43937, new Hash Method Rule keywords for the ICSF One-Way Hash Generate (CSNBOWH or CSNBOWH1 and CSNEOWH or CSNEOWH1) and PKCS11 One-Way Hash Services (CSFPOWH and CSFPOWH6) will support generation of legacy hash values for verification of archived hash values generated from pre-OA43937 releases of ICSF FMIDs HCR7770 through HCR77A1.

Note: This correction of hashing function does not apply to the case where the sum of the length of hashed text over a series of chained calls exceeds 256 megabytes (or 512, as described further in this topic), but no single invocation supplies an input *text_length* that exceeds 256 (or 512) megabytes. Correct hashes are created when no single invocation of the callable services exceeds the described limit prior to (and after) application of the PTFs for OA43937.

Applications that wish to verify archived hash values created by pre-OA43937 FMID HCR7770 through FMID HCR77A1 releases of ICSF callable services One-Way Hash Generate and PKCS11 One-Way Hash may need to invoke these callable services with new rule array keywords that support the creation of legacy hash values. The hash generated using the new rule array keywords must be used to verify the archived hash values.

The ICSF Callable Services One-Way Hash Generate and PKCS11 One-Way Hash, sign, or verify have corrected the way they create hash values when the length of the text on a single invocation of one of these services supplies an input *text_length* that equals or exceeds 256 megabytes (512 megabytes on z990/z890 or later hardware on FMID HCR7770). The hashing services are corrected with the application of the PTFs for OA43937.

Table 7 on page 58 provides more details about this migration action. Use this information to plan your changes to the system.

Table 7. Information about this migration action	
Element or feature:	Cryptographic Services.
When change was introduced:	PTFs for OA43937, which are applicable to: ICSF FMIDs HCR7770 - HCR77A1 (z/OS V1R12 - z/OS V2R1).
Applies to migration from:	ICSF FMIDs HCR7770 - HCR77A1, without the PTF for OA43937.
Timing:	Before the first start of FMID HCR77A1 or later ICSF FMIDs.
Is the migration action required?	Yes, if migrating from an ICSF FMID older than HCR77A1 to ICSF FMID HCR77A1 or later and if you have archived hash values created before the installation of the PTFs for OA43937 which meet the length restrictions described here.
Target system hardware requirements:	None.

Table 7. Information about this migration action (continued)

Target system software requirements:	None.
Other system (coexistence or fallback) requirements:	None.
Restrictions:	None.
System impacts:	If you do not use the legacy rule array keywords for affected applications, then the application may fail to verify the legacy hashes/signatures.
Related IBM Health Checker for z/OS check:	None.

Steps to take

Follow these steps:

1. Identify if your application needs to verify archived hash values created by either of the ICSF callable service One-Way Hash Generate (CSNBOWH or CSNBOWH1 and CSNEOWH or CSNEOWH1) or PKCS11 One-Way Hash (CSFPOWH and CSFPOWH6) on releases pre-OA43937 at FMID HCR7770 through FMID HCR77A1. (See the ICSF Application Programmer's Guide documentation changes in this APAR for new ICSF callable service keywords that support the creation of hashes for the verification of archived hash values and the input text length requirements.)
2. If your application has these archived hash values and intends to verify them, then invocations of ICSF callable services One-Way Hash Generate, PKCS11 One-Way Hash, sign, or verify that create hashes for verification of the archived hash values may need to be updated to use the new legacy rule array keywords (ONLY if those archived hash values were created with input text length exceeding the limits described).

Reference information

For more information, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

Actions to perform after the first start of ICSF FMID HCR77C0

This topic describes migration actions that you can perform only after you have started ICSF FMID HCR77C0. You need ICSF FMID HCR77C0 started to perform these actions.

Note: You may have already performed these migration actions if you previously migrated to ICSF FMIDs HCR77A1, HCR77B0, or HCR77B1.

ICSF: Accommodate the TRACEENTRY option deprecation

Description

In ICSF FMID HCR77A1 and later, option TRACEENTRY has been deprecated and ICSF CTRACE support has been enhanced to support configurable ICSF CTRACE options from PARMLIB. A default CTICSF00 PARMLIB member is installed in SYS1.PARMLIB. The CTICSF00 PARMLIB member provides default component trace values for ICSF. By default, ICSF CTRACE support will trace with the KdsIO, CardIO, and SysCall filters using a 2M buffer. Configurable options are commented out within this PARMLIB member to provide examples of how to turn them on.

Table 8 on page 59 provides more details about this migration action. Use this information to plan your changes to the system.

Table 8. Information about this migration action

Element or feature:	Cryptographic Services
----------------------------	------------------------

Table 8. Information about this migration action (continued)

When change was introduced:	Cryptographic Support for z/OS V1R13 - z/OS V2R1 web deliverable (FMID HCR77A1), which installs only on z/OS V1R13 or z/OS V2R1.
Applies to migration from:	z/OS V2R1 and z/OS V1R13 without the Cryptographic Support for z/OS V1R13 - z/OS V2R1 web deliverable (FMID HCR77A1). Note that when the Cryptographic Support for z/OS V1R13 - z/OS V2R1 Web deliverable (FMID HCR77A1) or later is not installed, this migration item is not applicable.
Timing:	After the first start of ICSF FMID HCR77B0.
Is the migration action required?	Yes, if you have installed the Cryptographic Support for z/OS V1R13 - z/OS V2R1 web deliverable (FMID HCR77A1) or later to handle TKDS with PKDS #11 objects for the new format in FMID HCR77A1 or later.
Target system hardware requirements:	None.
Target system software requirements:	None.
Other system (coexistence or fallback) requirements:	None.
Restrictions:	None.
System impacts:	If the TRACEENTRY option is specified it will be ignored and will produce message CSFO0212 at startup; processing continues.
Related IBM Health Checker for z/OS check:	None.

Steps to take

You can code the new CTRACE option within a BEGIN(HCR77A1) END option pair in a options data set shared between multiple releases of ICSF.

- If you share the installation options data set between FMID HCR77A1 and pre-FMID HCR77A1 systems, you can continue to supply the TRACEENTRY option at the lower-level systems as it is ignored, and processing will continue on the FMID HCR77A1 systems.
- If your installation cannot tolerate the CSFO0212 message that is issued at startup, you need to use different installation option data sets. Note that new CTRACE options will be in effect:
 - Review the default CTRACE options to ensure that they are satisfactory for your system.
 - Make any necessary changes. Use the CTICSF00 PARMLIB to create customized ICSF CTRACE Configuration Data Sets in PARMLIB. You can use the new CTRACE option to specify the customized ICSF CTRACE Configuration Data Set in the ICSF Options Data Set.

For example, you can specify CTRACE (CTICSFxx), where xx is any two characters that were used when copying the default CTICSF00 parmlib member.

Component tracing is active when ICSF starts using the trace options defined in the CTICSFxx PARMLIB member, where 00 is the default. If the CTICSF00 PARMLIB member is incorrect or missing, ICSF CTRACE performs tracing using an internal default set of trace options. The operator can specify trace options individually on the TRACE CT command or specify the name of a CTICSFxx PARMLIB member containing the desired trace options. Using a PARMLIB member on the TRACE CT command can help minimize operator intervention and avoid syntax or keystroke errors

Reference information

For more information, see the following references:

- [z/OS Cryptographic Services ICSF Administrator's Guide](#)
- For IBM Health Checker, see [IBM Health Checker for z/OS User's Guide](#).

Callable services

The following table summarizes the new and changed callable services for ICSF FMID HCR77C0. For complete reference information on these callable services, refer to *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Table 9. Summary of new and changed ICSF callable services

Callable service	FMID	Description
Digital Signature Generate	HCR77C0	Changed: Support for RSA-PSS digital signature scheme.
Digital Signature Verify	HCR77C0	Changed: Support for RSA-PSS digital signature scheme.
Key Data Set List	HCR77C0	Changed: New option to list unsupported CCA key in CKDS and PKDS.
PKA Key Generate	HCR77C0	Changed: Support for additional RSA public exponent values.
PKA Key Token Build	HCR77C0	Changed: Support for additional RSA public exponent values. Support for RSA-PSS digital signature scheme.
PKA Key Translate	HCR77C0	Changed: Support for RSA-PSS digital signature scheme.
ECC Diffie-Hellman	HCR77B1	Changed: Support for new derivation algorithm.
Encrypted PIN Translate Enhanced	HCR77B1	New: Reformat a PIN block where the PAN data is encrypted Visa Data Secure Platform (Visa DSP) processing.
Key Encryption Translate	HCR77B1	New: Change the method of encryption of DES key material.
Key Test2	HCR77B1	Changed: Support new key check value algorithm based on CMAC for DES and AES.
PKA Key Token Build	HCR77B1	Changed: Support for key derivation section for EC private keys added.
Symmetric Key Decipher	HCR77B1	Changed: Support Galois/Counter Mode for AES.
Symmetric Key Decipher	HCR77B1	Changed: Support Galois/Counter Mode for AES.
Field Level Decipher	HCR77B0	New: Decrypt data base fields, preserving the format of the fields using the VISA Format Preserving Encryption algorithm.
Field Level Encipher	HCR77B0	New: Encrypt data base fields, preserving the format of the fields using the VISA Format Preserving Encryption algorithm.
FPE Decipher	HCR77B0	New: Decrypt payment card data using Visa Data Secure Platform (Visa DSP) processing.
FPE Encipher	HCR77B0	New: Encrypt payment card data using Visa Data Secure Platform (Visa DSP) processing.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
FPE Translate	HCR77B0	New: Translate payment card data from encryption under one key to encryption under another key using Visa Data Secure Platform (Visa DSP) processing.
ICSF Multi-Purpose Service	HCR77B0	New: Validate the keys in the active CKDS or PKDS.
Key Data Set List	HCR77B0	New: Generate a list of labels or handles that match a label filter and metadata search criteria in an active key data set.
Key Data Set Metadata Read	HCR77B0	New: Read metadata for a record in an active key data set.
Key Data Set Metadata Write	HCR77B0	New: Add, delete, and change metadata for a list of records in an active key data set.
PKCS #11 One-way hash generate	HCR77B0	Changed: Legacy hash rules added.
PKCS11 One-way hash, sign, or verify	HCR77B0	Changed: Legacy hash rules added.
Authentication Parameter Generate	HCR77A1	New: Generate an authentication parameter (AP) and return it encrypted under a supplied encrypting key.
ICSF Query Facility 2	HCR77A1	New: Provides information on the cryptographic environment as currently known by ICSF.
Recover PIN From Offset	HCR77A1	New: Calculate an encrypted customer-entered PIN from a PIN generating key, account information, and an offset, returning the PIN properly formatted and encrypted under a PIN encryption key.
Symmetric Key Export with Data	HCR77A1	New: Export a symmetric key encrypted using an RSA key, inserted in a PKCS#1 block type 2, with some extra data supplied by the application.
Cipher Text Translate2 and Cipher Text Translate2 with alet	HCR77A0	New: Translates the user-supplied ciphertext from one key to another key.
Control Vector Generate	HCR77A0	Changed: <ul style="list-style-type: none"> • Support CIPHERXI, CIPHERXL and CIPHERXO key types. • Support DOUBLE-O rule_array keyword.
Derive ICC MK	HCR77A0	New: This service generates an ICC master key from an issuer master key.
Derive Session Key	HCR77A0	New: Use this callable service to derive a session key from either an issuer master key or an ICC master key.
Diversified Key Generate2	HCR77A0	New: Derive keys using a key-generating key.
DK Deterministic PIN Generate	HCR77A0	New: Generate a PIN using a secret key.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
DK Migrate PIN	HCR77A0	New: Generate a PIN reference value (PRW) for an existing IOS-1 PIN block.
DK PAN Translate	HCR77A0	New: Modify the PAN of an account while keeping the same PIN.
DK PIN Change	HCR77A0	New: Allow a customer to select a personal PIN.
DK PIN Verify	HCR77A0	New: Verify an ISO-1 PIN.
DK PAN Modify in Transaction	HCR77A0	New: This service is used to obtain a new PIN reference value (PRW) for an existing PIN when the account information has changed.
DK PRW Card Number Update	HCR77A0	New: Generate a PIN reference value (PRW) when a replacement card is being issued.
DK PRW CMAC Generate	HCR77A0	New: Generate a message authentication code (MAC) over specific values involved in an account number change transaction.
DK Random PIN Generate	HCR77A0	New: Generate a random PIN and PIN reference value.
DK Regenerate PRW	HCR77A0	New: Generate a new PIN reference value for a changed account number.
ECC Diffie-Hellman	HCR77A0	Changed: <ul style="list-style-type: none"> • Support CIPHERXI, CIPHERXL and CIPHERXO key types. • Support creation of DES keys with guaranteed unique key halves.
EMV Scripting Service	HCR77A0	New: This service simplifies EMV Scripting.
EMV Transaction Service	HCR77A0	New: This service simplifies EMV Authorization Request Cryptogram (ARQC) and Authorization Response Cryptogram (ARPC) transaction processing.
EMV Verification Functions	HCR77A0	New: This service provides additional functions used by MasterCard for their EMV cards in addition to application cryptograms and scripting.
Generate Issuer MK	HCR77A0	New: This callable service helps with the initial steps of EMV setup by generating and storing the issuer master keys.
Key Export	HCR77A0	Changed: Support CIPHERXI, CIPHERXL and CIPHERXO key types.
Key Generate	HCR77A0	Changed: <ul style="list-style-type: none"> • Generate DES DATAC, DATAM, and CIPHER keys as a single key in key forms OP, IM, and EX. • Support CIPHERXI, CIPHERXL and CIPHERXO key types. • Support DOUBLE-O key_length.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
Key Generate2	HCR77A0	Changed: <ul style="list-style-type: none"> • Support generating AES DKYGENKY, MAC, PINCALC, PINPROT, and PINPRW keys for DK AES PIN services. • Support generating AES CIPHER keys for use in Cipher Text Translate2 callable service.
Key Import	HCR77A0	Changed: Support CIPHERXI, CIPHERXL and CIPHERXO key types.
Key Part Import2	HCR77A0	Changed: Support AES DKYGENKY, MAC, PINCALC, PINPROT, and PINPRW keys for DK AES PIN services.
Key Test2	HCR77A0	Changed: Support AES DKYGENKY, MAC, PINCALC, PINPROT, and PINPRW keys for DK AES PIN services.
Key Token Build	HCR77A0	Changed: <ul style="list-style-type: none"> • Support CIPHERXI, CIPHERXL and CIPHERXO key types. • Support DOUBLE-O rule_array keyword.
Key Token Build2	HCR77A0	Changed: <ul style="list-style-type: none"> • Support generating AES DKYGENKY, MAC, PINCALC, PINPROT, and PINPRW key tokens. • Support C-XLATE keyword for AES CIPHER key type.
Key Translate2	HCR77A0	Changed: Support changing variable-length key tokens with variable-length payloads to fixed-length payloads.
ICSF Query Facility	HCR77A0	Changed: Retrieve weak PIN table from coprocessor.
MAC Generate2	HCR77A0	New: Generate a MAC using AES or HMAC keys.
MAC Verify2	HCR77A0	New: Verify a MAC using AES or HMAC keys.
Multiple Secure Key Import	HCR77A0	Changed: Support CIPHERXI, CIPHERXL and CIPHERXO key types
PKA Decrypt	HCR77A0	Changed: Support formatting data as RSA-OAEP block and both SHA-1 and SHA-256 hashing.
PKA Encrypt	HCR77A0	Changed: Support formatting data as RSA-OAEP block and both SHA-1 and SHA-256 hashing.
PKA Key Generate	HCR77A0	Changed: Support generating RSA keys that can be wrapped by AES keys.
PKA Key Import	HCR77A0	Changed: Support importing RSA keys that are wrapped by an AES key-encrypting key.
PKA Key Token Build	HCR77A0	Changed: Support building RSA-AESC and RSA-AESM skeleton tokens.
PKA Key Token Change	HCR77A0	Changed: Support reenciphering RSA keys wrapped by an ECC master key.
PKA Key Translate	HCR77A0	Changed: Support translating the object protection key (OPK) in a RSA private key token from a DES key to an AES key.

Table 9. Summary of new and changed ICSF callable services (continued)

Callable service	FMID	Description
PKCS #11 Private Key Structure Decrypt	HCR77A0	New: Decrypt data using a clear private key structure.
PKCS #11 Private Key Structure Sign	HCR77A0	New: Sign data using a clear private key structure.
PKCS #11 Public Key Structure Encrypt	HCR77A0	New: Encrypt data using a public key structure.
PKCS #11 Public Key Structure Verify	HCR77A0	New: Verify a signature using a public key structure.
Restrict Key Attribute	HCR77A0	Changed: <ul style="list-style-type: none"> • Support AES DKYGENKY, MAC, PINCALC, PINPROT, and PINPRW keys for DK AES PIN services. • Support C-XLATE rule_array keyword for AES CIPHER keys. • Support DOUBLE-O rule_array keyword for DES keys.
Secure Key Import	HCR77A0	Changed: Support CIPHERXI, CIPHERXL and CIPHERXO key types.
Secure Key Import2	HCR77A0	Changed: Support AES DKYGENKY, MAC, PINCALC, PINPROT, and PINPRW keys for DK AES PIN services.
Symmetric Key Export	HCR77A0	Changed: Support AES DKYGENKY, MAC, PINCALC, PINPROT, and PINPRW keys for DK AES PIN services.
Symmetric Key Import2	HCR77A0	Changed: Support AES DKYGENKY, MAC, PINCALC, PINPROT, and PINPRW keys for DK AES PIN services.
Unique Key Derive	HCR77A0	New: Use the Unique Key Derive callable service to derive a key using the Base Derivation Key and the Derivation Data. The following key types can be derived: <ul style="list-style-type: none"> • CIPHER • ENCIPHER • DECIPHER • MAC • MACVER • IPINENC • OPINENC • DATA token containing a PIN Key

Identification of cryptographic features

Starting in ICSF FMID HCR77B0, the prefix used to identify Crypto Express2, Crypto Express3, and Crypto Express4 adapters has changed. [Table 10 on page 66](#) lists the prefix for these adapters for FMIDs prior to HCR77B0 and the prefix for these adapters for FMID HCR77B0 and later releases. This change applies to ICSF messages, panels, and publications. The TKE workstation uses this same identification starting with TKE release 8.0.

Table 10. Cryptographic adapter identification

Cryptographic adapter	Prefix for FMIDs prior to HCR77B0	Prefix for FMID HCR77B0 and later
Crypto Express2 coprocessor	E	2C
Crypto Express2 accelerator	F	2A
Crypto Express3 coprocessor	G	3C
Crypto Express3 accelerator	H	3A
Crypto Express4 CCA coprocessor	SC	4C
Crypto Express4 EP11 coprocessor	SP	4P
Crypto Express4 accelerator	SA	4A

Note: All newer cryptographic adapters use the convention where *n* is the number in the adapter name. For regional cryptographic servers, *n* represents the generation number of the server:

nA

Crypto Express*n* accelerators.

nC

Crypto Express*n* CCA coprocessors.

nP

Crypto Express*n* EP11 coprocessors.

nR

Regional cryptographic servers. Note: 2R requires ICSF FMID HCR77B1 with PTF OA49069 or later.

Ensure the expected P11 master key support is available

ICSF introduced support for the Enterprise PKCS #11 (EP11) coprocessor and its associated P11 master key with FMID HCR77A0. ICSF uses the master key validation pattern (MKVP) in the header record of the TKDS to determine which EP11 coprocessors to make active. In FMID HCR77A0, an EP11 coprocessor was considered "active" if the MKVP in the current master key register matched the MKVP in the header record of the TKDS. If the MKVP did not match, or if the TKDS was never initialized, the EP11 coprocessor was considered "online", usable only for a limited number of non-secure key PKCS #11 services.

Starting with FMID HCR77A1, the online status no longer exists. Coprocessors are either active or in some error state. If the TKDS has been initialized, then any EP11 coprocessor that does not have a current master key register MKVP that matches the TKDS is not made active and, thus, not usable. Note, however, if the the TKDS has not been initialized, then all EP11 coprocessors will be made active even though they would only be usable for non-secure key PKCS #11 services.

Key store policy

KGUP

ICSF enhanced KGUP to enforce key store policy for duplicate key tokens in the CKDS. When the SAF XFACILIT resource CSF.CKDS.TOKEN.NODUPLICATES is enabled, KGUP checks for duplicate encrypted tokens in the CKDS for ADD and UPDATE control statements. When a duplicate token is found, the processing of that control statement is terminated.

This change might cause KGUP to fail if your ICSF administrator has enabled the CSF.CKDS.TOKEN.NODUPLICATES resource. If you are generating keys with random key values and the job fails because it is a duplicate key token, you should be able to rerun the job to generate a different key

value. If you are adding keys with a specific key value and the job fails, you should contact your ICSF administrator to determine what action to take.

Key material archiving

ICSF implemented a way to archive records in the key data sets. The record remains in the data set, but the key material in the record cannot be used. Any attempt to use the key material will fail unless the optional key archive use control (a SAF XFACILIT resource) is enabled, which allows the request to complete. An SMF record is logged in both cases. An optional joblog message is issued for the first successful reference if the key archive message control (KEYARCHMSG) is enabled. For more information, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

To use this function, the key data sets must be in the common record format (KDSR), introduced in FMID HCR77A1. Existing data sets can be converted to the KDSR format by using the Coordinated KDS Administration callable service. For more information, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

Key material validity

ICSF implemented a way to specify a period when the key material of a key data set record is active. The ICSF administrator can specify the start and end dates when the key material is active and ICSF allows only the key material to be used by applications within those dates. For more information, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

To use this function, the key data sets must be in the common record format (KDSR), introduced in FMID HCR77A1. Existing data sets can be converted to the KDSR format by using the Coordinated KDS Administration callable service. For more information, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

ICSF key data sets

Record metadata

ICSF implemented additional metadata for key data sets records. The metadata include key material validity dates, last referenced, archive and recall dates, and IBM and installation metadata blocks. This metadata can be used as search criteria for the Key Data Set List callable service. The metadata can be read by using the Key Data Set Metadata Read service. Some of the metadata can be added, deleted, and changed by using the Key Data Set Metadata Write service. For more information, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

To use this function, the key data sets must be in the common record format (KDSR), introduced in FMID HCR77A1. Existing data sets can be converted to the KDSR format by using the Coordinated KDS Administration callable service. For more information, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

CKDS

There are three formats of the CKDS:

- A fixed length record format with LRECL=252 (supported by all releases of ICSF). Sample is CSFCKDS.
- A variable length record format with LRECL=1024 (supported by HCR7780 and later releases). Sample is CSFCKD2.
- The common record format (KDSR) which is common to all key data sets with LRECL=2048 (supported by ICSF FMID HCR77A1 and later releases). Sample is CSFCKD3.

The variable length record format is only required if variable-length key tokens are to be stored in the CKDS. All fixed-length and variable-length symmetric key tokens can be stored in the variable-length record format CKDS. See [“Migrating to the variable length CKDS” on page 68](#) for more information.

In addition to supporting all symmetric key tokens, the KDSR format CKDS provides support for metadata for each record including tracking usage of the records. See [“Migrating to the common record format \(KDSR\) key data set” on page 69](#) for more information.

When new key types are added to the CKDS, the following consideration applies when sharing the CKDS:

- When clear DES or AES keys are added to the CKDS, RACF-protect all clear DES and AES keys by label name on all systems sharing the CKDS.

If you have no coprocessor, you can initialize the CKDS for use with clear AES and DES data keys. This CKDS cannot be used on a system with cryptographic coprocessors.

Note: The CKDS exits (single-record, read-write and retrieval) are not enabled for either variable-length record format of the CKDS. See [Chapter 5, “Installation exits,” on page 131](#) for more information.

Migrating to the variable length CKDS

If variable-length symmetric key tokens are to be stored in the CKDS, any existing CKDS must be converted to a variable length record format.

To convert to the LRECL = 1024 format, ICSF provides the conversion utility program, CSFCNV2, that converts a CKDS to the variable length format. See [Chapter 7, “Converting a CKDS from fixed length to variable length record format,” on page 195](#) for more information.

To convert to the KDSR format (ICSF FMID HCR77A1 or later), see [“Migrating to the common record format \(KDSR\) key data set” on page 69](#) for more information.

There is no reason to migrate a variable length record CKDS if your applications are not using AES or HMAC keys in variable-length tokens. You can migrate to the variable length record at any time.

Note: All systems that will share a CKDS with the variable length record format must be running ICSF FMID HCR7780 or later. Those with KDSR format must be running ICSF HCR77A1 or later.

To migrate to a variable length CKDS (LRECL=1024):

1. Install the HC7780 or later release of ICSF on all systems that will share the CKDS.
2. Allocate a new CKDS with the variable length record format. The new CKDS should be large enough to hold all key in the current CKDS.
3. Disable dynamic CKDS updates on all systems.
4. Run the CKDS Conversion2 utility to convert the existing CKDS records to the new record format
5. Refresh the new CKDS on all systems that are sharing the CKDS
6. Enable dynamic CKDS updates on all systems

PKDS

There are two formats of the PKDS: original and KDSR. Both formats use the same LRECL. The KDSR format provides support for metadata for each record including tracking usage of the record. To convert the original format PKDS to common record (KDSR) format, see [“Migrating to the common record format \(KDSR\) key data set” on page 69](#).

The process of re-enciphering the PKDS is different for IBM zEnterprise 196 or newer servers.

TKDS

There are two formats of the TKDS: original and KDSR. Both formats use the same LRECL. The KDSR format provides support for metadata for each record including tracking usage of the record. To convert the original format TKDS to common record (KDSR) format, see [“Migrating to the common record format \(KDSR\) key data set” on page 69](#).

For secure PKCS #11 support (either Enterprise PKCS #11 or regional cryptographic services), the TKDS must be initialized with the appropriate master key. This is the PKCS #11 master key (P11-MK) for Enterprise PKCS #11 services or the regional cryptographic services master key (RCS-MK) for regional cryptographic services. For P11-MK, support to INITIALIZE TKDS and UPDATE TKDS is available in the Master Key Management Panels. For RCS-MK, TKDS initialization implicitly happens the first time a regional cryptographic server is connected.

For information on managing and sharing the TKDS in a sysplex environment, see [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

Access authorization of the new callable services will be determined via SAF calls. No support will be provided for invocation of an installation security exit for these new services. The CSFSERV class controls access to the ICSF PKCS #11 callable services.

Migrating to the common record format (KDSR) key data set

All key data sets can be converted to KDSR format. Any system that will share the KDSR format key data set must be running ICSF FMID HCR77A1 or later.

The conversion is done with the active key data set, and a new key data set with the proper attributes for the KDSR format must be allocated.

The conversion can be done by either calling the CSFCRC callable service or by using the ICSF panels. While the conversion is happening, all updates to the key data set being converted are suspended. At the end of the conversion, all systems in the sysplex sharing the key data set will be using the KDSR format key data set as the active key data set. All new updates are made to the KDSR format key data set.

Converting to KDSR format using the CSFCRC callable service

A application must be written to invoke the CSFCRC callable service in order to convert a key data set to KDSR format. See [*z/OS Cryptographic Services ICSF Application Programmer's Guide*](#) for details about the CSFCRC callable service.

Converting to KDSR format using the ICSF panels

To convert a key data set to KDSR format using the ICSF panels, do the following:

1. On the ICSF Primary Menu panel, select option 2, KDS MANAGEMENT, and press ENTER.
2. When the ICSF Key Data Set Management panel appears, select the type of key data set you want to convert to KDSR format and press ENTER.
3. On the next panel, select the COORDINATED xKDS CONVERSION option and press ENTER.
4. When the ICSF Coordinated KDS conversion panel appears, fill in the required fields and press ENTER.

Changing the RSA master key

The process to reencipher the PKDS and change the RSA master key is different for IBM zEnterprise 196 and newer servers. For these systems, RSA master key change will be processed in the same manner as master key change for the DES, AES and ECC master keys.

This is the original procedure for changing the RSA master key for systems without CEX3C or newer coprocessors and at least the Sep. 2011 LIC, this procedure has not changed.

1. Disable dynamic PKDS updates control (recommended)
2. Disable PKA callable services control
3. Load the new RSA master key
 - TKE: load and set RSA master key
 - ICSF panels: loading the final key part causes the current master key to be set
4. Reencipher the PKDS (old to current master key)
5. Refresh the reenciphered PKDS
6. Enable PKA callable services control
7. Enable dynamic PKDS updates control

For systems with CEX3C or newer coprocessors and at least the Sep. 2011 LIC with the RSA master key loaded, this is the procedure for changing the RSA master key. See [*z/OS Cryptographic Services ICSF Administrator's Guide*](#) for more information.

1. Disable dynamic PKDS updates control (recommended)

2. Load the new RSA master key (TKE or ICSF panels)
3. Reencipher the PKDS (current to new master key)
4. Change the RSA master key (the current master key is set and the reenciphered PKDS becomes active PKDS)
5. Enable dynamic PKDS updates control

Note: When the new RSA master key change process is used:

- The PKA callable services control will not appear on the Administrator Control Functions panel.
- The availability of callable services that required the RSA master key is controlled by the state of the RSA master key. When the RSA master key is active (the master key verification pattern in the PKDS matches the verification pattern of the current RSA master key), RSA callable service are available. Message CSFM130I will be issued.
- The RSA master key cannot be set from the TKE workstation.

Migrating to 24-byte DES master key

ICSF and TKE accept a 16-byte key value for the DES master key. CCA coprocessors with the September 2012 licensed internal code (LIC) or later installed on a CEX3C or later will support both a 16- and 24-byte key value. ICSF and TKE will support loading both key value lengths.

To load a 24-byte DES master key, the **DES master key – 24-byte key** access control point must be enabled in the ICSF role in all CCA coprocessors for the domain where you wish to use a 24-byte DES master key. If the **DES master key – 24-byte key** access control point is not enabled consistently for all coprocessors available to a instance of ICSF, the DES new master key register cannot be loaded. The master key entry utility will fail. A TKE workstation is required to enable the access control point.

It is not possible to share a CKDS between systems with both 16- and 24-byte DES master keys. The master key verification pattern algorithm for the 24-byte DES master key is different from the algorithm for the 16-byte master key. The algorithms are described in the [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

The CKDS Reencipher and Symmetric Change Master Key utilities support both length key values. The coordinated CKDS administration functions support both length key values. The Passphrase KDS Initialization utility will load a 24-byte DES master key if the **DES master key – 24-byte key** access control point is enabled.

Warning: Due to control block changes required to support the 24-byte DES master key, after a 24-byte DES master key has been loaded, the LIC cannot be changed to an earlier version that does not support the 24-byte DES master key. If a change to an earlier LIC is required, all DES master keys must be changed back to 16-byte keys. This can be done using symmetric change master key.

Installation options data set

- AUDITKEYLIFECKDS – Controls auditing of key lifecycle events for CCA symmetric tokens.
- AUDITKEYLIFEPKDS – Controls auditing of key lifecycle events for CCA asymmetric tokens.
- AUDITKEYLIFETKDS – Controls auditing of key lifecycle events for PKCS #11 objects.
- AUDITKEYUSGCKDS – Controls auditing of key usage events for CCA symmetric tokens.
- AUDITKEYUSGPKDS – Controls auditing of key usage events for CCA asymmetric tokens.
- AUDITPKCS11USG – Controls auditing of usage events for PKCS #11 services.
- CKTAUTH - This option has been deprecated. If this option is specified, it is ignored and produces a CSFO0212 message.
- CTRACE - Specifies the CTUCSFxx ICSF CTRACE configuration data set to use from PARMLIB. CTICSF00 is the default ICSF CTRACE configuration data set that is installed with ICSF FMID HCR77A1 and later releases. CTICSF00 may be copied to create new PARMLIB members using the naming convention of CTUCSFxx, where xx is a unique value specified by the user.

This parameter is optional. If the specified PARMLIB member is incorrect or absent, ICSF CTRACE will attempt to use the default CTICSF00 PARMLIB member. If the CTICSF00 PARMLIB member is incorrect or absent, ICSF CTRACE will perform tracing using an internal default set of trace options. By default, ICSF CTRACE support will trace with the KdsIO, CardIO, and SysCall filters using a 2M buffer. For more information refer to [“Creating an ICSF CTRACE configuration data set”](#) on page 23.

- HDRDATE - This option has been deprecated. If this option is specified, it is ignored and produces a CSFO0212 message.
- KEYARCHMSG - Controls whether a joblog message is issued when an application successfully references a key data set record that has been archived.
- KEYAUTH - This option has been deprecated. If this option is specified, it is ignored and produces a CSFO0212 message.
- MASTERKCVLEN - Control the number of hexadecimal digits displayed for the CCA master keys on the ICSF Hardware Status panel.
- RNGCACHE - Controls whether ICSF maintains a cache of random numbers to be used by services that require them.
- TRACEENTRY - This option has been deprecated. If this option is specified, it is ignored and produces a CSFO0212 message. See the description of CTRACE for more information.

Function restrictions

Retained keys are RSA private keys that are stored in a cryptographic coprocessor instead of in the public key storage data set. This change does not affect retained keys that you are currently using, that is, keys that are stored on the cryptographic coprocessor. However, the ICSF services do not allow you to store in a cryptographic coprocessor RSA keys intended for key management use. Your applications can continue to store in the cryptographic coprocessor RSA private keys intended for signature usage. The modulus length of these private keys is limited to 2048-bits.

The 2048-bit RSA keys may have a public exponent, e , in the range of $1 < e < 2^{2048}$, and e must be odd. The RSA public key exponents for 2049-bit to 4096-bit RSA keys are restricted to the values 3 and 65537. The public exponent may be 5, 17, or 257 on a z13, z13s, or later server with the October 2016 or later licensed internal code.

CICS attachment facility

If you have the CICS Attachment Facility installed and you specify your own CICS wait list data set, you need to modify the wait list data set to include the new callable services.

Modify and include:

For FMID HCR77B1

CSFPTRE

For FMID HCR77B0:

CSFFLD, CSFFLE, CSFFPED, CSFFPEE, CSFFPET, CSFKDMW, CSFKDSL, CSFMPS.

For FMID HCR77A1:

CSFAPG, CSFPFO, CSFSXD.

For FMID HCR77A0:

CSFCTT2, CSFCTT3, CSFDCM, CSFDDPG, CSFDKG2, CSFDMP, CSFDPC, CSFDPCG, CSFDPMT, CSFDPNU, CSFDPT, CSFDPV, CSFDRP, CSFDRPG, CSFDSK, CSFEAC, CSFESC, CSFEVF, CSFGIM, CSFKET, CSFMGN2, CSFMGN3, CSFMVR2, CSFMVR3, CSF1PD2, CSF1PE2, CSF1PS2, CSF1PV2, CSFUDK.

Note: If no Wait List is specified, the default wait list will be used. See sample CSFWTL01 for the contents of the default wait list.

Dynamic LPA load

ICSF uses dynamic LPA to load the pre-PC routines, CICS related routines, and other modules which must reside in common storage into above-the-line ECSA. The dynamic LPA load will occur the first time that ICSF is started within an IPL, and the modules will persist across subsequent restarts of ICSF.

Special secure mode

Use of some ICSF services requires that ICSF be in special secure mode: CSNBPGN, CSNBSKI, CSNBSKI2, and CSNBSKM.

Resource Manager Interface (RMF)

Support to enable RMF to provide performance measurements on these selected ICSF services and functions. The measurements refer to these services processing on cryptographic coprocessors except for one-way hash. One-way hash is processed on CPACF.

- Decipher (CSNBDEC)
- Digital Signature Generate (CSNDDSG)
- Digital Signature Verify (CSNDDSV)
- Encipher (CSNBENC)
- Encrypted PIN Translate (CSNBPTR)
- Encrypted PIN Translate Enhanced (CSNBPTRE)
- FPE Decipher (CSNBPFED)
- FPE Encipher (CSNBPFEE)
- FPE Translate (CSNBPFET)
- MAC Generate (CSNBMGN)
- MAC Generate2 (CSNBMGN2)
- MAC Verify (CSNBMVR)
- MAC Verify2 (CSNBMVR2)
- One-Way Hash (CSNBOWH)
- PIN Verify (CSNBPVR)
- Symmetric Algorithm Decipher (CSNBSAD)
- Symmetric Algorithm Encipher (CSNBSAE)

System abend codes

A complete list of the reason codes for the ICSF abend (X'18F') is contained in *z/OS MVS System Codes*, which is published on release boundaries. As a migration aid for FMID HCR77C0, which is not on a release boundary, new and changed codes for FMID HCR77C0 are listed here. Release codes introduced in the previous web deliverables, FMIDs HCR77B1, HCR77B0, HCR77A1, HCR77A0, and HCR7790 are also listed.

An 18F code indicates an abend from ICSF.

FMID HCR77C0 reason codes are as follows:

Code Hex (Dec)

Meaning

484 (1156)

An error occurred during ATTACH of the CSFMKUT task.

487 (1159)

An error occurred during ATTACH of the CSFMIAKT task.

48C (1164)

ASCRES failed during early ICSF processing. This abend results in a wait state (X'040').

FMID HCR77B1 reason codes are as follows:

Code Hex (Dec)**Meaning****47C (1148)**

The regional cryptographic server returned a response parameter that has a length error.

47D (1149)

Error attaching CSFMICST.

47E (1150)

Received OPIN larger than should be possible.

480 (1152)

The regional cryptographic server configuration subtask ended.

481 (1153)

The regional cryptographic server request subtask ended.

482 (1154)

A regional cryptographic server encountered a toxic request.

483 (1155)

The regional cryptographic server returned an unsupported OID.

485 (1157)

Cleanup error: A Crypto block has been double freed.

486 (1158)

Cleanup error: Caller release.

FMID HCR77B0 reason codes are as follows:

Code Hex (Dec)**Meaning****477 (1143)**

Crypto processor encountered a toxic request (APAR OA43012).

478 (1144)

Damaged variable metadata detected.

479 (1145)

An error occurred during ATTACH of the CSFKSMPT task.

47A (1146)

ISGQUERY returned unexpected results.

47B (1147)

KDS bound to multiple KDS tokens.

47F (1151)

Received CC1 from protected-key operation.

The following reason codes are no longer issued as of FMID HCR77B0:

- 2 (2)
- 5 (5)
- A (10)
- 2C (44)
- 50 (80)
- A2 (162)
- 187 (391)
- 40A (1034)

- 40B (1035)
- 40C (1036)
- 40D (1037)
- 416 (1046)
- 429 (1065)
- 42A (1066)
- 475 (1141)

FMID HCR77A1 reason codes are as follows:

Code Hex (Dec)
Meaning

7E (126)

CSFMIRDT subtask cannot be restarted.

18F(44F)

Error while loading an ICSF module on startup. The load abend and reason codes are in registers 2 and 3. The most likely problem is the loading of signed module CSFINPV2. A signature verification failure would be indicated by R2 = 00000360, R3 = 00000040, which should be accompanied by security manager messages detailing the problem.

447 (1095)

Malformed request caused processor recovery.

46F (1135)

PKCS #11 Services detected an error in DER encoded data returned from the Enterprise PKCS #11 coprocessor.

473 (1139)

Data returned from XCP in invalid BER format.

474 (1140)

Data returned from XCP has unexpected attrs.

475 (1141)

XCF problems in the sysplex.

476 (1142)

PKCS #11 Services detected an error from the Enterprise PKCS #11 coprocessor on a derived key decryption request.

The following reason codes are no longer issued as of FMID HCR77A1:

- 470 (1136)
- 471 (1137)

For FMID HCR77A0, the descriptions of the following reason codes have changed:

Code Hex (Dec)
Meaning

1E (30)

Entry code to CSFKSTRE routine not valid.

76 (118)

Error from ATTACH in sysplex subtask control routine.

DB (219)

Error in ISGENQ during KDS sysplex serialization.

DC (220)

Error in IXCMGO during KDS sysplex serialization.

DD (221)

Error in IEAVPSE2 during KDS sysplex serialization.

EA (234)

Error establishing ESTAE for KDS sysplex subtask.

ED (237)

Error initializing KDS sysplex subtask.

FO (240)

Error in IXCJOIN for ICSF KDS sysplex group.

45D (1117)

CSFPLCMD/CSFPLMRT pause failure (XCF notify exit).

Reason codes for application services routines introduced in FMID HCR77A0 are:

Code Hex (Dec)**Meaning****467 (1127)**

Bad XCP message payload length detected.

468 (1128)

Bad XCP message encoding length detected.

469 (1129)

Bad XCP responses length detected.

46A (1130)

CSFSMBTI inconsistent internal control information.

46B (1131)

Terminate stuck I/O subtask.

46D (1133)

Message response too large.

46E (1134)

Reencipher of TKDS failed due to an error on coprocessor.

46F (1135)

PKCS #11 Services detected an error in DER encoded data returned from the Enterprise PKCS #11 coprocessor

470 (1136)

CSFNCCMK error in internal ICSF service call parms

471 (1137)

CSFNCRNC error in internal ICSF service call parms

472 (1138)

CSFKSCS2 detects unusable KDS

473 (1139)

CSFNCUWK data returned from XCP invalid BER format

474 (1140)

CSFNCUWK data returned from XCP has unexpected attrs

475 (1141)

CSFPLXNU XCF problems in the sysplex

476 (1142)

PKCS #11 Services detected an error from the Enterprise PKCS #11 coprocessor on a derived key decryption request.

Reason codes for application services routines introduced in FMID HCR7790 are:

Code Hex (Dec)**Meaning****2D (45)**

A cryptographic coprocessor returned a bad condition code. The coprocessor has been fenced off.

6F (111)

Error from ATTACH of the CSFPLMWT task.

F3 (243)

Error from IXCQUERY request in CSFPLCMD module.

F4 (244)

Error from IXCMSGO request in CSFPLMSR module.

F5 (245)

Error from IEAVPSE2 request in CSFPLMSR module.

FA (250)

Error determining sysplex KDS cluster member list in CSFPLCMD.

19A (410)

XCF message is too big for internal ICSF buffer.

19B (411)

An IXCMSGI request failure occurred receiving an XCF message.

45A (1114)

The CSFPLMWT task received a bad RC from IEAVPSE2.

45B (1115)

The CSFPLMRT task received inconsistent internal control information.

45C (1116)

The CSFSMBTM module received inconsistent internal control information.

45D (1117)

A pause failure occurred in the CSFPLPSC module.

45E (1118)

An IXCMSGO request failure occurred sending an XCF message.

45F (1119)

An XCF message exit error occurred during ICSF sysplex processing.

460 (1120)

A sequence number error occurred during ICSF sysplex processing.

461 (1121)

A cache update failure occurred during ICSF sysplex processing.

462 (1122)

The CSFKSIDC module was called with an incorrect function code.

463 (1123)

The CSFKSIIO module was called with an invalid IO function request.

464 (1124)

The CSFKSIIO module ran out of dynamic storage.

465 (1125)

ICSF forced termination.

466 (1126)

An error occurred during ATTACH of the CSFMISTP task.

SMF records

SMF record information for ICSF is documented in [Appendix B, “ICSF SMF records,” on page 339](#). Refer there for information on SMF records.

Subtypes 19, and 20 are written periodically to record processing times for requests being processed on a cryptographic coprocessor or accelerator. Subtype 19 is written for the PCIXCC. Subtype 20 is written for all supported processors.

TKE workstation

The Trusted Key Entry (TKE) workstation provided secure management of master and operational keys and management of access control points. Refer to [*z/OS Cryptographic Services ICSF TKE Workstation User's Guide*](#) for more information.

Access to callable services

Access to services that are executed on cryptographic coprocessors is through access control points in the Domain Role. To execute callable services on the coprocessor, access control points must be enabled for each service in the Role. For systems that do not use the optional TKE Workstation, all access control points (current and new) are enabled in the role with the appropriate microcode level on the cryptographic coprocessor.

For TKE users who have modified the Domain Role, all new access control points must be enabled using the TKE workstation. For non-TKE users, all new access control points are enabled.

Note: Some access control points are disabled by default in the Coprocessor Role. See the ICSF Application Programmer's Guide and [*z/OS Cryptographic Services ICSF Administrator's Guide*](#) for these access control points. A TKE Workstation is required to enable these access control points

TKE enablement from the support element

You must enable TKE commands on each cryptographic coprocessor from the support element. This is true for new TKE users and those upgrading their level of LIC. See Support Element Operations Guide and [*z/OS Cryptographic Services ICSF TKE Workstation User's Guide*](#) for more information.

Enabling access control points for PKCS #11 coprocessor firmware

A new or a zeroized Enterprise PKCS #11 coprocessor (or domain) comes with an initial set of Access Control Points (ACPs) that are enabled by default. All other ACPs, representing potential future support, are left disabled. When a firmware upgrade is applied to an existing Enterprise PKCS #11 coprocessor, the upgrade may introduce new ACPs. The firmware upgrade does not retroactively enable these ACPs, so they are disabled by default. These ACPs must be enabled via the TKE (or subsequent zeroize) in order to utilize the new support they govern. See Table 28. PKCS #11 Access Control Points in Writing PKCS #11 Applications for a complete description of the Access Control Points.

Table 11. Mapping of Enterprise PKCS #11 ACPs to firmware levels

Enterprise PKCS #11 firmware level	ACPs supported at this level	ACPs that need to be enabled when this code level is obtained via firmware upgrade
Initial release	<p>Control Point Management</p> <p>Allow addition (activation) of Control Points(0)</p> <p>Allow removal (deactivation) of Control Points(1)</p> <p>Cryptographic Operations</p> <p>Sign with private keys(2)</p> <p>Sign with HMAC or CMAC(3)</p> <p>Verify with HMAC or CMAC(4)</p> <p>Encrypt with symmetric keys(5)</p> <p>Decrypt with private keys(6)</p> <p>Decrypt with private keys(7)</p> <p>Key export with public keys(8)</p> <p>Key export with symmetric keys(9)</p> <p>Key import with private keys(10)</p> <p>Key import with symmetric keys(11)</p> <p>Generate asymmetric key pairs(12)</p> <p>Generate symmetric keys(13)</p> <p>Cryptographic Algorithms</p> <p>RSA private-key use(30)</p> <p>DSA private-key use(31)</p> <p>EC private-key use(32)</p> <p>Brainpool (E.U.) EC curves(33)</p> <p>NIST/SECG EC curves(34)</p> <p>Allow non-BSI algorithms (as of 2009) (21)</p> <p>Allow non-FIPS-approved algorithms (as of 2011) (35)</p> <p>Allow non-BSI algorithms (as of 2011) (36)</p> <p>Key Size</p> <p>Allow 80 to 111-bit algorithms(24)</p> <p>Allow 112 to 127-bit algorithms(25)</p> <p>Allow 128 to 191-bit algorithms(26)</p> <p>Allow 192 to 255-bit algorithms(27)</p> <p>Allow 256-bit algorithms(28)</p> <p>Allow RSA public exponents below 0x10001(29)</p> <p>Miscellaneous</p> <p>Allow backend to save semi-retained keys not applicable(14)</p> <p>Allow keywrap without attribute-binding(16)</p> <p>Allow changes to key objects (usage flags only) (17)</p> <p>Allow mixing external seed to RNG not applicable(18)</p> <p>Allow non-administrators to mark key objects TRUSTED(37)</p> <p>Do not double-check sign/decrypt operations(38)</p> <p>Allow dual-function keys - key wrapping and data encryption(39)</p> <p>Allow dual-function keys - digital signature and data encryption(40)</p> <p>Allow dual-function keys - key wrapping and digital signature(41)</p> <p>Allow non-administrators to mark public key objects ATTRBOUND(42)</p> <p>Allow clear passphrases for password-based-encryption(43)</p> <p>Allow wrapping of stronger keys by weaker keys(44)</p> <p>Allow clear public keys as non-attribute bound wrapping keys(45)</p>	None - all default ACPs enabled in the initial release.
Version 2 Sept. 2013 or later licensed internal code (LIC)	<p>Set for initial release plus</p> <p>Cryptographic Operations</p> <p>Allow key derivation (47)</p> <p>Cryptographic Algorithms</p> <p>DH Private Key Use (46)</p>	<p>Cryptographic Operations</p> <p>Allow key derivation (47)</p> <p>Cryptographic Algorithms</p> <p>DH Private Key Use (46)</p>

Migrating from the IBM eServer zSeries 900

This topic discusses migration from the IBM eServer zSeries 900.

Migrating a CKDS and PKDS between a CCF system and a non-CCF system

The Cryptographic Coprocessor Feature (CCF) systems are the z900 and z800. The PCI Cryptographic Coprocessor (PCICC) is an optional feature.

The following systems will be referred to as non-CCF systems in this section. A cryptographic feature is required on the non-CCF systems.

- z9 EC and z9 BC with the optional Crypto Express2 Coprocessor (CEX2C).
- z10 EC and z10 BC with the optional Crypto Express2 Coprocessor (CEX2C) and Crypto Express3 Coprocessor (CEX3C).
- z114 and z196 with the optional Crypto Express3 Coprocessor (CEX3C).
- zBC12 and zEC12 with the optional Crypto Express3 Coprocessor (CEX3C) and Crypto Express4 Coprocessor (CEX4C).
- z13 and z13s with the optional Crypto Express5 Coprocessor (CEX5C).

The processing of the RSA-MK on a non-CCF system depends on the cryptographic features on your system. See “Changing the RSA master key” to determine which processing must be done to load and set the RSA-MK. The PKA Callable Services control is not active on all systems.

CCF only system

SMK equal to KMMK

- Using Master Key Entry
 1. Start ICSF on a non-CCF system, pointing to the initialized CKDS/PKDS.

You will see one or more of these messages depending on your system's cryptographic features:
CSFM124I MASTER KEY xxx ON CRYPTO EXPRESSn COPROCESSOR xxnn, SERIAL NUMBER
nnnnnnnn, NOT INITIALIZED.
 2. Using Master Key Entry, load the value of the CCF DES master key into the new DES-MK register. Load the value of the CCF SMK/KMMK master key into the new RSA-MK register. You will need the checksums for each of these values.
 3. If the non-CCF system has coprocessors (CEX3C or later) with the September, 2011 LIC or later, set the DES and RSA master keys using the SET MK utility.
 4. If the non-CCF system has coprocessors (CEX3C or earlier) without the September, 2011 LIC, do the following steps.
 - Set the DES master key using the SET MK utility.
 - The ASYM-MK will have already been set when the last master key value was entered.
 - Enable the Dynamic PKDS Access control and the PKA Callable Services control.
- Using Pass Phrase Initialization
 1. Start ICSF on a non-CCF system, specifying the initialized CKDS and PKDS in the options data set.
 2. Using PPINIT, type in the same pass phrase used to initialize CCF system, select the Reinitialize system option and type in the CKDS and PKDS names.

SMK not equal to KMMK

Without a PCICC, the PKDS reencipher must run on any CCA Cryptographic coprocessor. If it is not, the non-CCF system will not be able to use the tokens encrypted under the KMMK. This procedure requires that you switch between your CCF and non-CCF TSO sessions.

- Using Master Key Entry

If the non-CCF system has coprocessors (CEX3C or later) with the September, 2011 LIC or later, you must reencipher to the KMMK. On older systems, it does not matter whether you reencipher to the KMMK or the SMK.

This procedure reenciphers to the KMMK.

1. Start ICSF on a non-CCF system, pointing to the initialized CKDS and PKDS.
2. Define an empty PKDS.
3. Load the value of the CCF DES master key into the new DES-MK register. You will need the checksum.
4. Set the DES master key using the SET MK utility.
5. Load the value of the CCF SMK master key into the new RSA-MK register. You will need the checksum.

If the non-CCF system has coprocessors (CEX3C or later) with the September, 2011 LIC or later, do the following steps:

- Set the RSA-MK using the SET MK utility
- Load the value of the CCF KMMK master key into the new RSA-MK register. You will need the checksum.
- Reencipher the active PKDS to the empty PKDS.
- Change the RSA-MK using the CHANGE ASYM MK utility.

If the non-CCF system has coprocessors (CEX3C or earlier) without the September, 2011 LIC, do the following steps:

- Load the value of the CCF KMMK master key into the new RSA-MK register. You will need the checksum. The RSA-MK will be set automatically when the last key part is loaded.
- Reencipher the active PKDS to the empty PKDS.
- Refresh the new PKDS. Enable PKA Callable Services and Dynamic PKDS Access control.

6. Update options data set to point to the new PKDS.
 7. On CCF system, disable PKA Callable Services.
 8. Reset the SMK register.
 9. Load the value of the CCF KMMK master key into the SMK register.
 10. Activate the new PKDS.
 11. Enable PKA Callable Services and Dynamic PKDS Access controls.
 12. Update options data set to point to the new PKDS.
- Using Pass Phrase Initialization
 1. On a CCF system, use PPKEYS utility to get the clear key values of the SMK and KMMK from a pass phrase. You will need the checksum for each of these values.
 2. On a non-CCF system, start ICSF pointing to initialized CKDS and PKDS.
 3. Define an empty PKDS.

If the non-CCF system has coprocessors (CEX3C or later) with the September, 2011 LIC or later, do the following steps:

- a. Using PPINIT, type in the same pass phrase used to initialize CCF system, select the Reinitialize system option and type in the CKDS and PKDS names.
- b. Using Master Key Entry, load the value of the CCF KMMK master key into the new RSA-MK register. You will need the checksum. Load a final key part of zeroes.
- c. Reencipher the PKDS to the empty PKDS.
- d. Change the RSA-MK using the CHANGE ASYM MK utility
- e. Update the options data set to point to the new PKDS.
- f. On a CCF system, disable PKA Callable Services.
- g. Using Master Key Entry, reset the SMK register.

- h. Load the value of the KMMK into the SMK register. You can get the clear key value of the KMMK using the PPKEYS utility. You will need the KMMK checksum.
- i. Activate the new PKDS.
- j. Enable PKA Callable Services/Dynamic PKDS Access.
- k. Update the options data set to point to the new PKDS.

If the non-CCF system has coprocessors (CEX3C or earlier) without the September, 2011 LIC, do the following steps:

- a. Using Master Key Entry, load the value of the CCF KMMK master key into the new RSA-MK register. You will need the checksum. Load a final key part of zeroes. The RSA-MK is automatically set when the final key part is loaded.
- b. Using PPINIT, type in the same pass phrase used to initialize CCF system, select the Reinitialize system option and type in the CKDS and PKDS names.
- c. Reencipher the PKDS to the empty PKDS.
- d. Refresh the new PKDS.
- e. Update the options data set to point to the new PKDS.
- f. On a CCF system, disable PKA Callable Services.
- g. Using Master Key Entry, reset the KMMK register.
- h. Load the value of the SMK into the KMMK register. You can get the clear key value of the SMK using the PPKEYS utility. You will need the SMK checksum.
- i. Activate the new PKDS.
- j. Enable PKA Callable Services/Dynamic PKDS Access.
- k. Update the options data set to point to the new PKDS.

CCF with PCICCs

SMK equal to KMMK

- Using Master Key Entry
 1. Start ICSF on a non-CCF system, pointing to the initialized CKDS/PKDS.
You will see one or more of these messages depending on your system's cryptographic features:
CSFM124I MASTER KEY xxx ON CRYPTO EXPRESSn COPROCESSOR xxnn, SERIAL NUMBER
nnnnnnnn, NOT INITIALIZED.
 2. Using Master Key Entry, load the value of the CCF DES master key into the new DES-MK register. Load the value of the CCF SMK/KMMK master key into the new RSA-MK register. You will need the checksums for each of these values.
 3. If the non-CCF system has coprocessors (CEX3C or later) with the September, 2011 LIC or later, set the DES-MK and RSA-MK using the SET MK utility.
 4. If the non-CCF system has coprocessors (CEX3C or earlier) without the September, 2011 LIC, do the following steps:
 - Set the DES-MK using the SET MK utility.
 - The RSA-MK will have already been set when the last master key value was entered.

SMK not equal to KMMK

Make the SMK equal to KMMK prior to sharing the CKDS and PKDS on a non-CCF system.

- Using Master Key Entry
 1. Define an empty PKDS.
 2. On the CCF system, disable the PKA Callable Services control.

3. Using Master Key Entry, reset ALL-PKA registers. Load the value of the CCF KMMK master key into the SMK/KMMK/ASYM-MK registers on all CCF and PCICC coprocessors. You will need the checksum. The ASYM-MK is automatically set when the final key part is loaded.
4. Reencipher the PKDS to the empty PKDS.
5. Activate the new PKDS.
6. Enable the PKA Callable Services and Dynamic PKDS Access controls.
7. Update the options data set to point to the new PKDS.
8. Start ICSF on the non-CCF system pointing to initialized CKDS and PKDS.
9. Load the value of the CCF DES master key into the new DES-MK register.
10. Set the DES-MK using the SET MK utility.

If the non-CCF system has coprocessors (CEX3C or later) with the September, 2011 LIC or later, do the following steps:

- Load the value of the CCF KMMK master key into the new RSA-MK register. You will need the checksum.
- Set the RSA-MK using the SET MK utility.

If the non-CCF system has coprocessors (CEX3C or earlier) without the September, 2011 LIC, do the following steps:

- Load the value of the CCF KMMK master key into the new RSA-MK register. You will need the checksum. The RSA-MK is automatically set when the final key part is loaded.
- Enable the PKA Callable Services and Dynamic PKDS Access controls. The current RSA-MK now has the same value as the SMK/KMMK on the CCF.

- Using Pass Phrase Initialization

1. On the CCF system, use PPKEYS to get the clear key values of the SMK and KMMK from a pass phrase. You will also need the checksum for each of these values.
2. Define an empty PKDS. Disable PKA Callable Services.
3. Using Master Key Entry, load the value of the CCF KMMK master key into the new ASYM-MK register on the PCICC or PCICCs. You will need the checksum. Load a final key part of zeroes. The ASYM-MK is automatically set when the final key part is loaded. The current ASYM-MK is now the same as the KMMK value.
4. Load the value of the CCF SMK into the new ASYM-MK register on the PCICC or PCICCs. You will need the checksum. Load a final key part of zeroes. The ASYM-MK is automatically set when the final key part is loaded. The current ASYM-MK is now the same as the SMK value. The KMMK value is now in the old ASYM-MK register.
5. Reset the KMMK register on the CCFs. Load the SMK value into the KMMK register. Now the KMMK = SMK.
6. Reencipher the PKDS to the empty PKDS.
7. Activate the new PKDS.
8. Enable the PKA Callable Services and Dynamic PKDS Access controls.
9. Update options data set to point to the new PKDS.
10. Start ICSF on a non-CCF system, pointing to the initialized CKDS and PKDS (the one just reenciphered previously).
11. Using PPINIT, type in the same pass phrase used to initialize CCF system, select the Reinitialize system option and type in the CKDS and PKDS names.

Callable services

These services were only available on the IBM eServer zSeries 900. These services are not supported on newer servers.

- ANSI X9.17 EDC Generate (CSNAEGN)
- ANSI X9.17 Key Export (CSNAKEX)
- ANSI X9.17 Key Import (CSNAKIM)
- ANSI X9.17 Key Translate (CSNAKTR)
- ANSI X9.17 Transport Key Partial Notarize (CSNAKTR)
- Ciphertext Translate (CSNBCTT)
- PKSC Interface Service (CSFPKSC)
- Transform CDMF Key (CSNBTK)
- User Derived Key (CSFUDK)

A migration check, ICSFMIG_DEPRECATED_SERV_WARNINGS, was provided to detect the use of these services. You must migrate away from the use of these services, because support is removed. You should investigate applications using these services, and determine the appropriate actions to remove or replace them.

Functions not supported

This topic lists functions not supported without a CCF installed.

1. There is no KMMK (key management master key).
2. The Commercial Data Masking Facility (CDMF) is no longer supported. The CDMF keyword on KGUP control statements and panels are no longer supported.
3. The Public Key Algorithm Digital Signature Standard is not supported. This affects callable services CSNDPKG, CSNDPKI, CSNDDSG, and CSNDDSV.
4. The PBVC keyword is not supported. This affects callable services Clear PIN Generate Alternate (CSNBCPA), PIN Translate (CSNBPTR) and PIN Verify (CSNBPVR).

Setup considerations

This topic lists setup changes that should be considered when migrating from a IBM eServer zSeries 900.

Consideration should be given to:

1. CICS wait list should be updated for services now executing on PCIXCCs/CEX2Cs. The sample CICS wait list, CSFWTL01, supplied by IBM includes these services and can be used as a reference.
2. PKDS initialization is required.
3. Options data set keywords have changed. See [“Parameters in the installation options data set” on page 30.](#)
4. If sharing a PKDS with a PCICC and PCIXCC/CEX2C, delete the PKDS records for labelnames of retained keys on PCICCs no longer in use.
5. Customers who run CSFEUTIL to setup ICSF for automated electronic delivery process no longer need to execute CSFEUTIL on a newer servers. SHA-1 is available without entering ICSF master keys.

Programming considerations

This topic lists setup changes that should be considered when migrating from a IBM eServer zSeries 900.

Consideration should be given to:

1. The DATAC key type cannot be used on the newer servers.
2. The PIN block format checking on the new cryptographic coprocessors is more rigorous than with a CCF.

For CSNBPVR, CSNBPTR and CSNBCPA services, the input PIN block must have the correct format as specified in the PIN Profile parameter. On a CCF system, the PIN block format checking is incomplete.

For example, the REFORMAT processing mode of PIN Translate (CSNBPTR) may now fail when it was previously successful on a CCF. On a CCF, if input to the PIN verify service (CSNBPVR) is a malformed encrypted PIN block, the service will fail with return code 4, reason code 3028 (verification failed); on newer servers, the service may fail with return code 8 and some appropriate reason code for invalid PIN format.

3. 512 to 2048 bit modulus for RSA keys is supported in all PKA services except SET services (Set Block Compose and Set Block Decompose).
4. All CCF functions are now executed on the coprocessors. This may cause some impact on the performance of customer applications.
5. Reason codes from the new servers may be different from previous cryptographic hardware.
6. On new servers, the requirement that caller must be in supervisor state to use NOCV tokens is lifted for the CKDS Key Record Write (CSNBKRW) service.
7. The z/OS SCHEDULE and IEAMSCHD macros are used to schedule SRBs. On the newer servers, since there are no CCFs on the system, applications should delete FEATURE=CRYPTO on the SCHEDULE and IEAMSCHD macros or the SRB being scheduled will not run.
8. External tokens that are export prohibited are imported differently on z990 and later servers with PCIXCC or CCA Crypto Express coprocessors. The imported internal token will have the same control vector as the external token with export prohibited. These tokens will only be usable on z990 and later servers with a PCIXCC/CEX2C or on CCF systems with PCICCs. On previous hardware (CCF systems) the imported internal token had a control vector that allowed export, and export prohibition was enforced by the export flag in the token.
9. Prohibit Export service can now be used for MAC and MACVER keys.
10. A RACF check is added to the Key Generation Utility (CSFKGUP).
11. The CSFKGUP utility exit control block has been changed for AES. See [Chapter 5, "Installation exits," on page 131](#) for the new format.

Chapter 4. Operating ICSF

You use certain commands to operate ICSF. Also, there are different conditions for operating ICSF that you should consider. This topic describes the ICSF operating tasks.

Starting and stopping ICSF

To start ICSF, issue the operator START command. You must issue the START command after each IPL. You can start ICSF only as a started task.

ICSF should be started as early in initialization as possible as one of first commands in COMMNDxx, rather than later automation. ICSF should be started with SUB=MSTR to eliminate any need to wait for JES. This also allows ICSF to be shut down after JES.

This example shows the format of the START command to start ICSF, assuming that CSF is the name of the start procedure:

```
START CSF, SUB=MSTR
```

To reuse ASIDs, the REUSASID parameter can be added to the START comment:

```
START CSF, SUB=MSTR, REUSASID=YES
```

To stop ICSF, issue the operator STOP command. After you issue the STOP command, all ICSF processing stops. If ICSF stops successfully, a message that states that ICSF is stopped appears on the console.

During shutdown, ideally ICSF is shut down after OMVS and JES are taken down. This allows any final updates to encrypted file systems to be successfully processed. By shutting down ICSF gracefully, it allows ICSF to complete all processing for updates to the key data sets.

This example shows the format of the STOP command to stop ICSF, assuming that CSF is the name of the started procedure:

```
STOP CSF
```

If ICSF is unresponsive to the STOP command, be aware that you are not able to use the CANCEL command to stop ICSF processing. Instead, use the force command:

```
FORCE csfproc, arm
```

Master key validation

When ICSF is started, the master keys are checked against the key data sets.

For CCA, master key verification patterns (MKVP) stored in the cryptographic key data set (CKDS) and the public key data set (PKDS) are compared to the current master keys. A CCA coprocessor becomes active if the current master keys match the MKVPs found in the CKDS and PKDS. If there is any mismatch, the coprocessor does not become active. When an MKVP is not in the CKDS or PKDS, the master key is ignored.

For an Enterprise PKCS #11 (EP11) coprocessor, ICSF uses the master key validation pattern (MKVP) in the header record of the TKDS to determine which EP11 coprocessors to make active. An EP11 coprocessor is active if the MKVP in the current master key register matched the MKVP in the header record of the TKDS or the TKDS has not been initialized.

When ICSF successfully starts, a message indicating that initialization is complete appears on the console.

Note:

1. If a problem is detected with a cryptographic coprocessor or with an accelerator during initialization, message CSFM540I is generated and the device is bypassed.
2. The ICSF_COPROCESSOR_STATE_NEGCHANGE health check monitors the state of the coprocessors and accelerators daily to detect a negative change in state. For more information about this health check, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).
3. The ICSF_MASTER_KEY_CONSISTENCY health check evaluates the master key states of the coprocessors to detect potential master key problems. For more information about this health check, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Starting ICSF during IPL-time

In addition to starting ICSF manually, ICSF can be started automatically during IPL-time. Starting ICSF during IPL-time allows callers of ICSF to take advantage of ICSF functionality during IPL-time. This functionality is available on ICSF FMID HCR77C0 and later running on z/OS V2R3, with PTF for APAR OA55378 applied, and later.

Both the ICSFPROC and ICSF system parameters must be specified in order to start ICSF automatically during IPL-time. You can specify the values of the ICSFPROC and ICSF system parameters in one or more of the following places:

- The IEASYSxx parmlib member.
- By the operator, in response to message IEA101A SPECIFY SYSTEM PARAMETERS.

If you define the values in only the IEASYSxx parmlib member, the system uses that definition. Otherwise, the system determines the ICSFPROC and ICSF system parameters using the values specified via the operator response to message IEA101A SPECIFY SYSTEM PARAMETERS.

To configure ICSF to start during IPL-time:

1. Configure the ICSFPROC system parameter. The ICSFPROC system parameter specifies the ICSF startup procedure to be used during early ICSF initialization. ICSFPROC can be omitted or 'NONE' can be specified to prevent ICSF from starting early. If 'NONE' is specified, ICSF must be started manually. The procedure must reside in a SYS1.PROCLIB data set or an equivalent that is specified by the IEFPSI DD card specification of the MSTJCLxx PARMLIB member. If the procedure is not in this location, ICSF will not start. For information about MSTJCL, see [z/OS MVS Initialization and Tuning Reference](#).

```
ICSFPROC=CSF2
```

```
ICSFPROC=NONE
```

2. Configure the ICSF system parameter. The ICSF system parameter specifies the xx value of the CSFPRMxx member containing the installation options data set. For example, a value of 00 would correspond to the CSFPRM00 member. ICSF can be omitted or 'NONE' can be specified to prevent ICSF from starting early. If 'NONE' is specified, ICSF must be started manually.

```
ICSF=00
```

```
ICSF=NONE
```

3. Modify the ICSF startup procedure. The ICSF startup procedure must be modified to accept the PRM procedure variable. The PRM procedure variable must be set to the xx value of the CSFPRMxx member containing the installation options data set. The following example shows how this would look using the CSFPARM DD statement:

```
//CSF PROC PRM=00
//CSF EXEC PGM=CSFINIT,REGION=0M,TIME=1440,MEMLIMIT=NOLIMIT
//CSFPARM DD DSN=USER.PARMLIB(CSFPRM&PRM),DISP=SHR
```

4. IPL the system. If both the ICSFPROC and ICSF system parameters are configured correctly and the ICSF startup procedure exists and is coded correctly, ICSF starts during IPL-time.

Notes:

- For information on the syntax of the ICSFPROC and ICSF system parameters, see IEASYSxx (system parameter list) in *MVS Initialization and Tuning Reference*.
- For information on how to setup the ICSF startup procedure, see [“Steps to create the ICSF startup procedure”](#) on page 25.
- It is recommended that you set up an AUTOR policy to auto reply to the BCF005A and BCF006A messages after a specified amount of time has passed. In the example below, ICSF must be started manually if the auto reply is NONE after 60 seconds.

```
MSGID(BCF005A) DELAY(60S) REPLY(NONE)
```

```
MSGID(BCF006A) DELAY(60S) REPLY(NONE)
```

- You should remove any existing invocations that start ICSF and rely on ICSF startup at IPL-time. For example, look for any commands that start ICSF in the COMMNDxx parmlib member. After the system brings up ICSF automatically, the system rejects any attempt to bring up a second instance of ICSF. The system issues the following warning message and terminates the second instance of ICSF:

```
CSFM004A ICSF TERMINATING. ICSF ALREADY ACTIVE.
```

- ICSF, when started during IPL-time, is started as a system address space. Any processing (including automation) that relies on ICSF being started as a job (started task) might need to make changes. For example, ICSF would not be included in the output of the DISPLAY JOBS,LIST or DISPLAY A,LIST system commands.

Note: ICSF address space is still included in the output of the DISPLAY JOBS,ALL and DISPLAY A,ALL system commands.

Modifying ICSF

When you issue the MODIFY command, ICSF gives control to the installation exit CSFEXIT5, if it exists. Your installation can write an exit routine for CSFEXIT5 that changes ICSF operations. For example, you might have the installation exit change the CHECKAUTH installation option without having to stop and restart ICSF. See [Chapter 5, “Installation exits,”](#) on page 131 for a description of the installation exits.

If your installation does not write an exit routine for CSFEXIT5, no action occurs when you enter the MODIFY command.

Command syntax notation

You must follow certain syntactical rules when you code the ICSF commands.

How to read syntax diagrams

This section describes how to read syntax diagrams. It defines syntax diagram symbols, items that may be contained within the diagrams (keywords, variables, delimiters, operators, fragment references, operands) and provides syntax examples that contain these items.

Syntax diagrams pictorially display the order and parts (options and arguments) that comprise a command statement. They are read from left to right and from top to bottom, following the main path of the horizontal line.

For users accessing IBM Knowledge Center using a screen reader, syntax diagrams are provided in dotted decimal format.

Symbols

The following symbols may be displayed in syntax diagrams:

Symbol

Definition



Indicates the beginning of the syntax diagram.



Indicates that the syntax diagram is continued to the next line.



Indicates that the syntax is continued from the previous line.



Indicates the end of the syntax diagram.

Syntax items

Syntax diagrams contain many different items. Syntax items include:

- Keywords - a command name or any other literal information.
- Variables - variables are italicized, appear in lowercase, and represent the name of values you can supply.
- Delimiters - delimiters indicate the start or end of keywords, variables, or operators. For example, a left parenthesis is a delimiter.
- Operators - operators include add (+), subtract (-), multiply (*), divide (/), equal (=), and other mathematical operations that may need to be performed.
- Fragment references - a part of a syntax diagram, separated from the diagram to show greater detail.
- Separators - a separator separates keywords, variables or operators. For example, a comma (,) is a separator.

Note: If a syntax diagram shows a character that is not alphanumeric (for example, parentheses, periods, commas, equal signs, a blank space), enter the character as part of the syntax.

Keywords, variables, and operators may be displayed as required, optional, or default. Fragments, separators, and delimiters may be displayed as required or optional.

Item type

Definition

Required

Required items are displayed on the main path of the horizontal line.

Optional

Optional items are displayed below the main path of the horizontal line.

Default

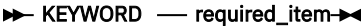
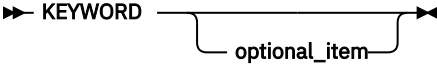
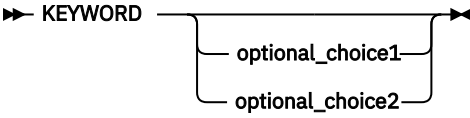
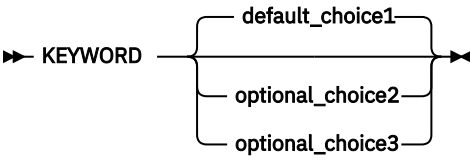
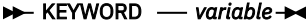
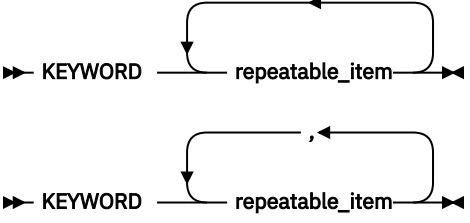
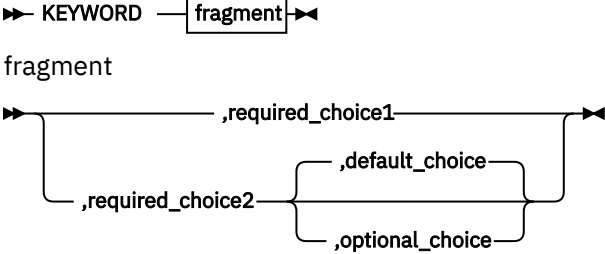
Default items are displayed above the main path of the horizontal line.

Syntax examples

The following table provides syntax examples.

Table 12. Syntax examples	
Item	Syntax example
Required choice. A required choice (two or more items) appears in a vertical stack on the main path of the horizontal line. You must choose one of the items in the stack.	

Table 12. Syntax examples (continued)

Item	Syntax example
<p>Required item.</p> <p>Required items appear on the main path of the horizontal line. You must specify these items.</p>	
<p>Optional item.</p> <p>Optional items appear below the main path of the horizontal line.</p>	
<p>Optional choice.</p> <p>An optional choice (two or more items) appears in a vertical stack below the main path of the horizontal line. You may choose one of the items in the stack.</p>	
<p>Default.</p> <p>Default items appear above the main path of the horizontal line. The remaining items (required or optional) appear on (required) or below (optional) the main path of the horizontal line. The following example displays a default with optional items.</p>	
<p>Variable.</p> <p>Variables appear in lowercase italics. They represent names or values.</p>	
<p>Repeatable item.</p> <p>An arrow returning to the left above the main path of the horizontal line indicates an item that can be repeated.</p> <p>A character within the arrow means you must separate repeated items with that character.</p> <p>An arrow returning to the left above a group of repeatable items indicates that one of the items can be selected, or a single item can be repeated.</p>	
<p>Fragment.</p> <p>The fragment symbol indicates that a labelled group is described below the main syntax diagram. Syntax is occasionally broken into fragments if the inclusion of the fragment would overly complicate the main syntax diagram.</p>	

ICSF operator commands

Beginning with ICSF FMID HCR77B1 and later, ICSF provides support for the following operator commands:

“Display ICSF” on page 90

Displays information about ICSF.

“SETICSF” on page 95

Used to perform specific administration functions.

Note: Installation options modified by the SETICSF command are in effect only until ICSF is stopped or restarted. When ICSF is restarted, the installation options will be re-initialized from the ICSF installation options data set. If you want to make the changes permanent, the installation options data set must be manually updated as needed.

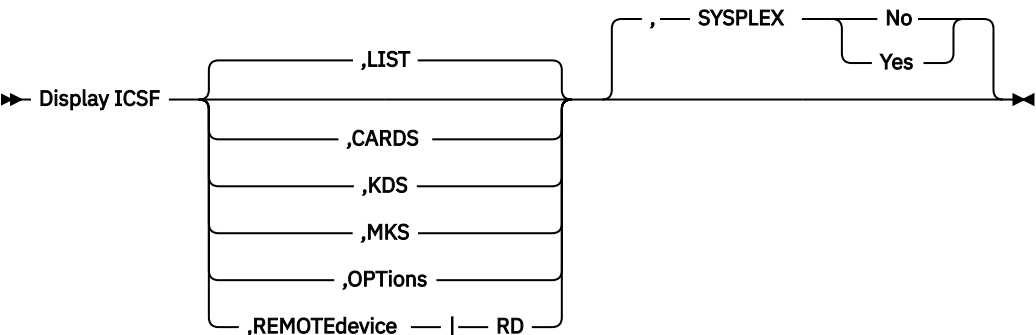
To see how to use RACF profiles to restrict the use of the ICSF operator commands, see ['Controlling the use of operator commands'](#) in *z/OS Security Server RACF Security Administrator's Guide*. The ICSF operator commands are a subset of the MVS system operator commands. Therefore, the subsystem name used in defining the RACF profiles is 'MVS' and the complete profiles names are 'MVS.DISPLAY.ICSF' and 'MVS.SETICSF.ICSF', as defined in ['MVS commands, RACF access authorities, and resource names'](#) in *z/OS MVS System Commands*.

Display ICSF

On systems running ICSF FMID HCR77B1 or later, and running z/OS V1R13 (with the PTF for APAR OA47380 installed) or later, use the Display ICSF command to:

- Display the status for available cryptographic devices.
- Display certain ICSF options.
- Display key lifecycle auditing options.
- Display key usage auditing options.
- Display information about regional cryptographic servers (remote devices).
- Display information pertaining to active key data sets (KDS).
- Display the status of the master key registers for the available cryptographic devices.
- List the systems that are available to participate in commands with a SYSPLEX scope.

Syntax



Parameters

CARDS

The system displays the following (message CSFM668I) information about the cryptographic devices available on the system or sysplex:

- The active domain.

- For each available device:
 - The device type (for example, CRYPTO EXPRESS5 COPROCESSOR).
 - The device index (for example, 5C36).
 - The device status (for example, Active).
 - The device serial number (for example, 99EA6059).
 - The firmware level of the device (for example, 5.0.45).
 - The number of requests both active and in the work queue for the device.

For example:

```
D ICSF,CARDS

CSFM668I 09.35.33 ICSF CARDS 292
ACTIVE DOMAIN = 003
CRYPTO EXPRESS5 COPROCESSOR 5C36
  STATUS=Active          SERIAL#=99EA6059 LEVEL=5.0.45
  REQUESTS ACTIVE=0002
CRYPTO EXPRESS5 COPROCESSOR 5P40
  STATUS=Active          SERIAL#=97006099 LEVEL=02.09 CLIC=0742
  REQUESTS ACTIVE=0001
CRYPTO EXPRESS5 ACCELERATOR 5A42
  STATUS=Active
  REQUESTS ACTIVE=0000
```

KDS

The system displays (message CSFM668I) information about the active key data sets (KDS) on the system or sysplex:

- The dataset name for each active KDS (CKDS, PKDS, and TKDS).
- The format of the KDS (for example, KDSR):
 - Possible values are KDSR, FIXED, and VARIABLE.
- The communication level in place for the KDS (for example, 3). This is only displayed in a sysplex environment.
- Whether the KDS is being shared in a sysplex group (for example, Y).
- The MKVPs initialized in the KDS (for example, DES AES).
 - The possible values are:
 - DES, AES, or both for CKDS.
 - RSA, ECC, or both for PKDS.
 - P11, RCS, or both for TKDS.

For example:

```
SYSA D ICSF,KDS

SYSA CSFM668I 14.38.31 ICSF KDS 040
CKDS RACFDRVR.SHERID.CKDSPLX
  FORMAT=KDSR          COMM LVL=3  SYSPLEX=Y  MKVPs=DES AES
PKDS RACFDRVR.SHERID.PKDSPLX
  FORMAT=KDSR          COMM LVL=3  SYSPLEX=Y  MKVPs=RSA ECC
TKDS RACFDRVR.SHERID.TKDSPLX
  FORMAT=KDSR          COMM LVL=3  SYSPLEX=Y  MKVPs=P11 RCS
```

MKS

The system displays (message CSFM668I) master key information:

- The name of the system (for example, SYSA).
- The active domain (for example, 003).
- For each device on the system:
 - The device index (for example, 5C38).

- The device serial number (for example, 99EA6059).
- The status of the device.
- A status indicator for each possible master key.

For more information on the possible display values, see the Displaying Coprocessor or Accelerator Status topic in [z/OS Cryptographic Services ICSF Administrator's Guide](#).

For example:

```
SYSA  D ICSF,MKS

SYSA  CSFM668I 09.45.18 ICSF MKS 852
      SYSNAME: SYSA   DOMAIN: 003   CPC Name: PR2827A
      FEATURE SERIAL# STATUS      AES DES ECC RSA P11
      5C38   99EA6059 Active      A  A  A  A
      5P39   97006054 Active                      A
```

LIST

The system displays (message CSFM668I) members of a sysplex who are eligible to participate in Display ICSF and SETICSF commands. LIST is the default option.

For example:

```
SYSA  D ICSF,LIST

SYSA  CSFM668I 14.57.29 ICSF LIST 984
      Systems supporting SETICSF and Display ICSF commands:
      SYSA      HCR77B1  DOMAIN = 003
      SYSB      HCR77B1  DOMAIN = 003
```

OPTions

The system displays (message CSFM668I information):

- The name of the system (for example, SYSA).
- The ICSF release that is active (for example, HCR77B1).
- The most recent build date of ICSF executable code (for example, 01/09/16 or the latest ICSF code change).
- How much time must elapse between key references before a *refdate* change is recorded in the KDS record (*refdate* update interval).
- How often KDS *refdate* updates are hardened to the KDS dataset (*refdate* update period).
- The number of master key verification pattern digits.

For example:

```
SYSA      D ICSF,OPTIONS
SYSA      CSFM668I 10.23.21 ICSF OPTIONS 833
      SYSNAME = SYSA      ICSF LEVEL = HCR77C0
      LATEST ICSF CODE CHANGE = 08/22/16
      Refdate update interval in Days/HH.MM.SS = 030/00.00.00
      Refdate update period   in Days/HH.MM.SS = 000/01.00.00
      MASTERKCVLEN = display 3 digits
      AUDITKEYLIFECKDS: Audit CCA symmetric key lifecycle events
      SYSNAME LABEL TOKEN
      SYSA     Yes   Yes
      AUDITKEYLIFEPKDS: Audit CCA asymmetric key lifecycle events
      SYSNAME LABEL TOKEN
      SYSA     Yes   Yes
      AUDITKEYLIFETKDS: Audit PKCS #11 key lifecycle events
      SYSNAME TOKOBJ SESSOBJ
      SYSA     Yes   Yes
      AUDITKEYUSGCKDS: Audit CCA symmetric key usage events
      SYSNAME LABEL TOKEN Interval Days/HH.MM.SS
      SYSA     Yes   Yes   000/01.00.00
      AUDITKEYUSGPKDS: Audit CCA asymmetric key usage events
      SYSNAME LABEL TOKEN Interval Days/HH.MM.SS
      SYSA     Yes   Yes   000/01.00.00
      AUDITPKCS11USG: Audit PKCS #11 usage events
      SYSNAME TOKOBJ SESSOBJ NOKEY Interval Days/HH.MM.SS
      SYSA     Yes   Yes   Yes   000/01.00.00
```

REMOTEdevice|RD

Displays information about regional cryptographic servers (remote devices) on either the local system or if SYSPLEX=YES, all systems in the sysplex.

Notes:

- At least one REMOTEDEVICE option must have been specified in the ICSF installation options data set prior to ICSF being started in order for the Display ICSF,REMOTEDEVICE command to be operational.
- In addition, the current machine type must be an IBM zEnterprise EC12 or later machine.
- If ICSF is started without any REMOTEDEVICE entries specified in the ICSF installation options data set or while running on a machine type other than an IBM zEnterprise EC12 or later machine, the Display ICSF,REMOTEdevice command fails, and ICSF issues message CSFM669I.

The results of the command are displayed through message CSFM668I:

- The dataset name for the active TKDS (for example, CSF.TKDS2).
- The first three hexadecimal bytes of the regional cryptographic server master key verification pattern from the TKDS (for example, AB1122).
- For each device on the system:
 - The device serial number (for example, 87651130).
 - The device port number (for example, 8001).
 - The level indicating the generation of card code (for example, LEVEL=01.00).
 - The HOST/IP of the device (for example, HOST/IP@=123.45.34.100).
 - The remote device identifier (REGIONAL CRYPTO SRV); for example, 1R09, where:
 - 1 = Generation of the device.
 - R = Remote regional cryptographic server.
 - 09 = Index as defined in the options dataset.
 - The status of the device (for example, Active).
 - The current number of socket connections / the maximum number of socket connections as defined in the options dataset (for example, 7/8).

Note: If the current number of sockets = the maximum number of sockets defined, only one number is displayed (as with the second example showing Sockets=8).

 - The current number of active cryptographic requests on the device (In this example, 5 for the first remote device (serial number 87651130) and 6 for the second remote device (serial number 87661276).
 - Optional new master key information: The first three hexadecimal bytes of the regional cryptographic server new master key verification pattern and the state of the new master key (for example, FULL COMMITTED).

Note: During heavy workloads or when SYSPLEX=YES is specified, the display command may be unable to retrieve a recently updated new master key value. If the new master key verification pattern that is displayed does not match the new master key loaded from the RCS utility, wait 10 minutes for an implicit RCS check and then reissue the display command. Otherwise, issue the SETICSF RESTART command for each RCS device.

 - Optional diagnostic information: Displays the device MKVP when the regional cryptographic server master key does not match that in the TKDS.

For example, when SYSPLEX=NO is specified or used by default from SYSA with 2 remote devices:

```
SYSA  D ICSF,RD
SYSA  CSFM668I 04.47.06 ICSF RD 424
      TKDS = CSF.TKDS2
      RCS MKVP FROM TKDS = AB1122 ...
      SERIAL NUMBER=87651130 PORT=8001 LEVEL=01.00
```

```

HOST/IP@=123.45.34.100
REGIONAL CRYPTO SRV 1R06
SYSA      Active          Sockets=7/8
REQUESTS ACTIVE=0005
SERIAL NUMBER=87661276 PORT=8001 LEVEL=01.00
HOST/IP@=123.45.34.101
REGIONAL CRYPTO SRV 1R09
SYSB      Active          Sockets=8
REQUESTS ACTIVE=0006

```

When SYSPLEX=YES is specified, ICSF collects the remote device information from all the systems in the sysplex for display through message CSFM668I. The output of message CSFM668I is sorted and grouped using the sort keys:

- TKDS
- SERIAL NUMBER
- PORT

For example, when SYSPLEX=YES is specified:

```

SYSA  D ICSF,RD,SYSPLEX=Y
SYSA  CSFM668I 05.54.31 ICSF RD 502
      TKDS = CSF.TKDS2
      RCS MKVP FROM TKDS = AB1122 ...
      SERIAL NUMBER=87651130 PORT=8001 LEVEL=01.00
      HOST/IP@=123.45.34.100
      REGIONAL CRYPTO SRV 1R06
      SYSB      Active          Sockets=8
      REQUESTS ACTIVE=0000
      SERIAL NUMBER=87651130 PORT=8002 LEVEL=01.00
      HOST/IP@=123.45.34.100
      REGIONAL CRYPTO SRV 1R06
      SYSC      Active          Sockets=8
      REQUESTS ACTIVE=0000
      SERIAL NUMBER=87651130 PORT=8003 LEVEL=01.00
      HOST/IP@=123.45.34.100
      REGIONAL CRYPTO SRV 1R06
      SYSC      Active          Sockets=8
      REQUESTS ACTIVE=0000
      SERIAL NUMBER=87661062 PORT=8003 LEVEL=01.00
      HOST/IP@=123.45.34.103
      REGIONAL CRYPTO SRV 1R16
      SYSC      Active          Sockets=8
      REQUESTS ACTIVE=0000
      SERIAL NUMBER=87661276 PORT=8001 LEVEL=01.00
      HOST/IP@=123.45.34.101
      REGIONAL CRYPTO SRV 1R09
      SYSB      Active          Sockets=8
      REQUESTS ACTIVE=0000
      SERIAL NUMBER=87661276 PORT=8002 LEVEL=01.00
      HOST/IP@=123.45.34.101
      REGIONAL CRYPTO SRV 1R09
      SYSC      Active          Sockets=8
      REQUESTS ACTIVE=0000
      SERIAL NUMBER=87661276 PORT=8003 LEVEL=01.00
      HOST/IP@=123.45.34.101
      REGIONAL CRYPTO SRV 1R09
      SYSC      Active          Sockets=8
      REQUESTS ACTIVE=0000
      SERIAL NUMBER=87671176 PORT=8003 LEVEL=01.00
      HOST/IP@=123.45.34.102
      REGIONAL CRYPTO SRV 1R13
      SYSC      Active          Sockets=8
      REQUESTS ACTIVE=0000

```

SYSPLEX(YES or NO)

The SYSPLEX keyword increases the scope of the Display ICSF command to all participating members of the sysplex. The Display ICSF output is grouped according to CPC Name and shows the results of the Display ICSF command as it was executed on each member. Specify SYSPLEX=Yes to execute the command on all systems. Otherwise, specify SYSPLEX=No to execute the command only on the local (initiating) system. SYSPLEX=No is the default.

For example:

```
D ICSF,CARDS,SYSPLEX=Y

CSFM668I 11.49.49 ICSF CARDS 919
CPC Name = R01          CPC Sequence# = 0000000000042E08
CRYPTO EXPRESS5 COPROCESSOR 5C57 SERIAL#=99EA6003 LEVEL=5.1.4
SYSA    DOMAIN=000 Active      REQUESTS=0000
SYSB    DOMAIN=002 Active      REQUESTS=0000
SYSC    DOMAIN=008 Active      REQUESTS=0000
CRYPTO EXPRESS5 COPROCESSOR 5P58 SERIAL#=97006035 LEVEL=02.09
SYSA    DOMAIN=000 Active      REQUESTS=0000
SYSB    DOMAIN=002 Active      REQUESTS=0000
SYSC    DOMAIN=008 Active      REQUESTS=0000
CPC Name = R02          CPC Sequence# = 0000000000042E09
CRYPTO EXPRESS5 COPROCESSOR 5P59 SERIAL#=97006102 LEVEL=02.09
SYSA    DOMAIN=000 Active      REQUESTS=0000
CRYPTO EXPRESS5 ACCELERATOR 5P60
SYSC    DOMAIN=008 Active      REQUESTS=0000
```

```
SYSA D ICSF,OPT,SYSPLEX=Y

SYSA CSFM668I 11.36.35 ICSF OPTIONS 995
SYSNAME = SYSA          ICSF LEVEL = HCR77B1
LATEST ICSF CODE CHANGE = 01/09/15
Refdte update interval in Days/HH.MM.SS = 030/00.00.00
Refdte update period in Days/HH.MM.SS = 000/01.00.00
MASTERKCVLEN = display 3 digits
SYSNAME = SYSB          ICSF LEVEL = HCR77B1
LATEST ICSF CODE CHANGE = 01/09/15
Refdte update interval in Days/HH.MM.SS = 005/00.00.00
Refdte update period in Days/HH.MM.SS = 000/01.00.00
MASTERKCVLEN = display 3 digits
```

Usage Notes

For information on how to limit the use of MVS console commands to a specific set of users, see the System Operations topic in [z/OS MVS System Commands](#).

SETICSF

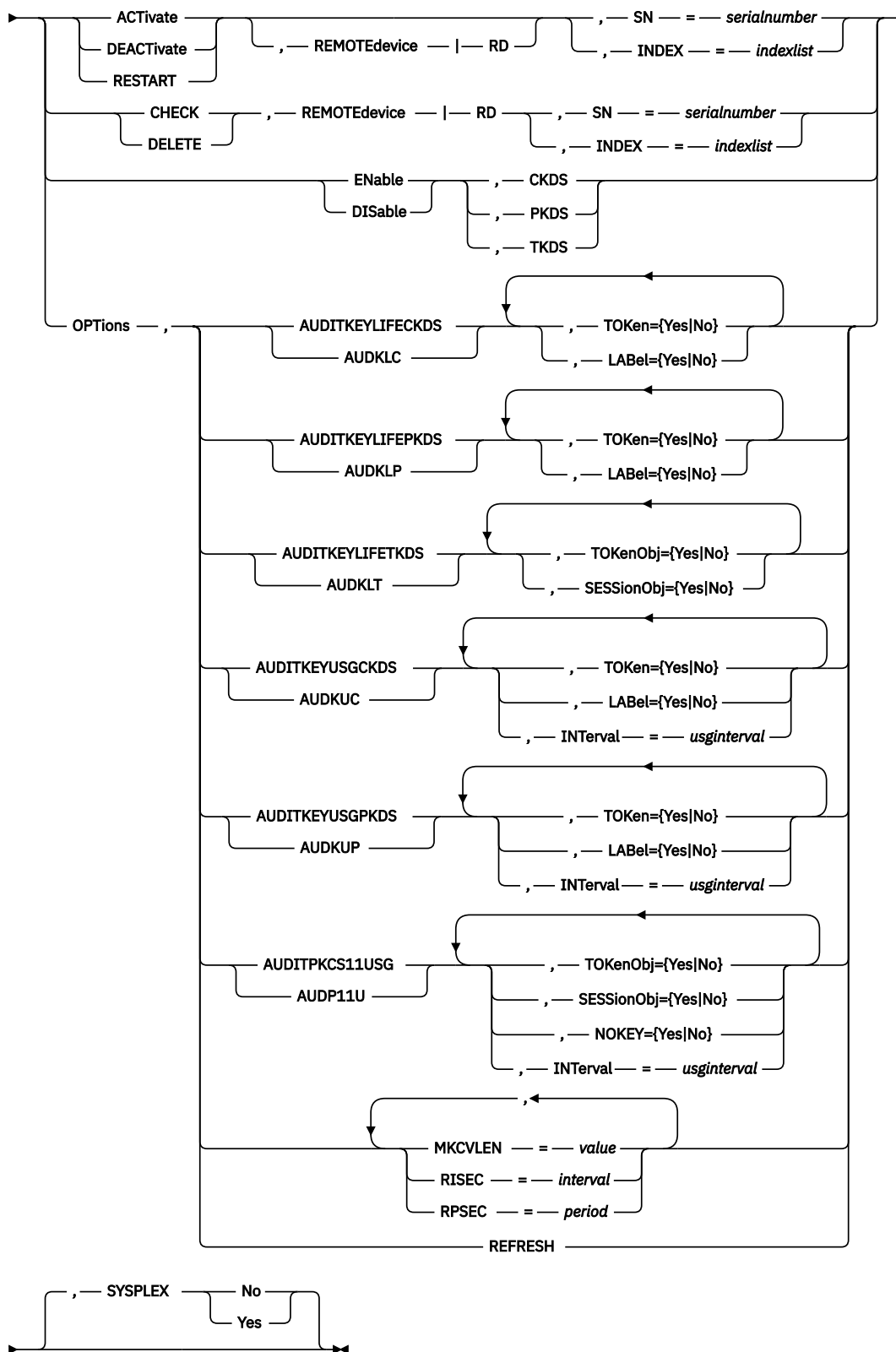
On systems running ICSF FMID HCR77B1 or later, and running z/OS V1R13 (with the PTF for APAR OA47380 installed) or later, the SETICSF command is used to perform the following specific administration functions:

- Activate, deactivate, or restart a cryptographic device.
- Add, check, or delete a regional cryptographic device.
- Attempt to reopen sockets that were not previously opened.
- Change a subset of ICSF's installation options.
- Enable or disable updates to a key data set (KDS).
- Change key lifecycle auditing options.
- Change key usage auditing options.
- Refresh some options in the installation options data set.

Note: For additional information on these administrative functions and their impact on ICSF and cryptographic devices, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Syntax

➤ SETICSF ➤



Add keyword


```

➡ ADD — , — REMOTEdevice — | — RD — , — INDEX — = — indexlist — ➡
      |
      | — , — IP — = — ip-addr-or-hostname —
      | — , — PORT — = — port-number —
      | — , — SOCK — = — number-sockets —

```

Parameters

ACTivate

Activates the specified cryptographic device or devices. The valid device specifications are:

REMOTEdevice

The regional cryptographic server or servers (remote device or devices). REMOTEdevice is optional.

Notes:

- At least one REMOTEDEVICE option must have been specified in the ICSF installation options data set prior to ICSF being started in order for the SETICSF ACTivate,REMOTEdevice command to be operational.
- In addition, the current machine type must be an IBM zEnterprise EC12 or later machine.
- If ICSF is started without any REMOTEDEVICE entries specified in the ICSF installation options data set or while running on a machine type other than an IBM zEnterprise EC12 or later machine, the command fails, and ICSF issues message CSFM670I.

SN=*serialnumber*

Specify the serial number or numbers of the device or devices to be activated. The *serialnumber* value can be a single serial number or a list of serial numbers separated by commas. When more than one value is provided, the set of values must be enclosed in parentheses. For example:

```

SN=99AE6012
SN=(99AE6012,99AE6013,99AE6014)

```

INDEX=*indexlist*

Specify the index or indexes of the device or devices to be activated. The valid range is 0 to 63, or 1-16 when REMOTEdevice is specified. The *indexlist* value can be a single device index, a range of indexes separated by a colon, or a combination of the two separated by commas. When more than one value is provided, the set of values must be enclosed in parentheses. For example:

```

INDEX=01
INDEX=(02:08)
INDEX=(02,04:07,09)

```

Note: To understand how the use of the INDEX value with the SYSPLEX parameter can result in devices with different serial numbers being modified on other systems sharing the KDS, see the explanation of the SYSPLEX parameter.

ADD

Adds a regional cryptographic server (single system only).

Notes:

- At least one REMOTEDEVICE option must have been specified in the ICSF installation options data set prior to ICSF being started in order for the SETICSF ADD,REMOTEdevice command to be operational.
- In addition, the current machine type must be an IBM zEnterprise EC12 or later machine.
- If ICSF is started without any REMOTEDEVICE entries specified in the ICSF installation options data set or while running on a machine type other than an IBM zEnterprise EC12 or later machine, the ADD command fails and ICSF issues message CSFM670I.
- SYSPLEX=YES is not supported for the SETICSF ADD,REMOTEdevice command.

REMOTEdevice|RD

The regional cryptographic server or servers (remote device or devices). All of the following operands must be specified:

INDEX=index-number

Specify the index of the device to be added. Specify a number between 1 and 16, inclusive. Each operational REMOTEDEVICE must have a unique number so SETICSF ADDing a index that already exists will fail. For indexes that are repeated, ICSF will only save the last one specified. Additionally, if remote devices or ports are shared between sysplex members, it is strongly recommended that the same index number is used for each member.

IP=ip-addr-or-hostname

Specify the dotted-decimal Internet protocol (IP) version 4 address or the hostname of the remote device. Each *ip-addr-or-hostname* must locate a single device with fixed serial number. Reverse proxy arrangements where one *ip-addr-or-hostname* is backed by multiple devices (with different serial numbers) is not supported. The opposite arrangement (one serial number assigned to multiple *ip-addr-or-hostnames*) is supported, but not recommended.

Notes:

- Hostnames are not case-sensitive and are stored and displayed by ICSF in lowercase.

PORT=port-number

Specify the port number to be used in conjunction with the IP address or hostname when connecting.

Note: No two ICSF instances may share the same port on a regional cryptographic server. Additionally, it is expected that different workloads (for example, ICSF instances using different token data sets) sharing a regional cryptographic server would use different master keys (RCS-MKs) and that the required RCS-MK for the TKDS would be assigned on a per port basis.

SOCK=number-sockets

Specify the maximum number of sockets ICSF is to open for connections with the remote device. This is a value between 1 and 8, inclusive. Multiple sockets are required in order for ICSF to process multiple simultaneous requests. Consult the remote device's documentation to determine this value. There is an ICSF limit of 8 sockets per REMOTEDEVICE entry. If you desire more than 8 socket connections to a single server, then configure multiple ports on the server and define multiple REMOTEDEVICE entries, one per port. Note that the index value must be unique for each entry.

CHECK

Attempts to reopen sockets that were not previously opened.

REMOTEdevice

The regional cryptographic server or servers (remote device or devices).

Notes:

- At least one REMOTEDEVICE option must have been specified in the ICSF installation options data set prior to ICSF being started in order for the SETICSF CHECK,REMOTEdevice command to be operational.
- In addition, the current machine type must be an IBM zEnterprise EC12 or later machine.
- If ICSF is started without any REMOTEDEVICE entries specified in the ICSF installation options data set or while running on a machine type other than an IBM zEnterprise EC12 or later machine, the command fails, and ICSF issues message CSFM670I.

SN=serialnumber

Specify the serial number or numbers of the device or devices to be checked. The *serialnumber* value can be a single serial number or a list of serial numbers separated by commas. When more than one value is provided, the set of values must be enclosed in parentheses. For example:

```
SN=99AE6012
SN=(99AE6012,99AE6013,99AE6014)
```

INDEX=indexlist

Specify the index or indexes of the device or devices to be checked. Specify a number between 1 and 16, inclusive. The *indexlist* value can be a single device index, a range of indexes separated by a colon, or a combination of the two separated by commas. When more than one value is provided, the set of values must be enclosed in parentheses. For example:

```
INDEX=01
INDEX=(02:08)
INDEX=(02,04:07,09)
```

DEACTivate

Deactivates specified cryptographic devices. The valid device specification are:

REMOTEdevice

The regional cryptographic server or servers (remote device or devices). REMOTEdevice is optional.

Notes:

- At least one REMOTEDEVICE option must have been specified in the ICSF installation options data set prior to ICSF being started in order for the SETICSF DEACTivate, REMOTEdevice command to be operational.
- In addition, the current machine type must be an IBM zEnterprise EC12 or later machine.
- If ICSF is started without any REMOTEDEVICE entries specified in the ICSF installation options data set or while running on a machine type other than an IBM zEnterprise EC12 or later machine, the command fails, and ICSF issues message CSFM670I.

SN=serialnumber

Specify the serial number or numbers of the device or devices to be deactivated. The *serialnumber* value can be a single serial number or a list of serial numbers separated by commas. When more than one value is provided, the set of values must be enclosed in parentheses. For example:

```
SN=99AE6012
SN=(99AE6012,99AE6013,99AE6014)
```

INDEX=indexlist

Specify the index or indexes of the device or devices to be deactivated. The valid range is 0 to 63, or 1-16 when REMOTEdevice is specified. The *indexlist* value can be a single device index, a range of indexes separated by a colon, or a combination of the two separated by commas. When more than one value is provided, the set of values must be enclosed in parentheses. For example:

```
INDEX=01
INDEX=(02:08)
INDEX=(02,04:07,09)
```

Note: To understand how the use of the INDEX value with the SYSPLEX parameter can result in devices with different serial numbers being modified on other systems sharing the KDS, see the explanation of the SYSPLEX parameter.

DELETE

Removes a regional cryptographic server from a system or systems.

REMOTEdevice

The regional cryptographic server or servers (remote device or devices).

Notes:

- At least one REMOTEDEVICE option must have been specified in the ICSF installation options data set prior to ICSF being started in order for the SETICSF DELETE,REMOTEdevice command to be operational.
- In addition, the current machine type must be an IBM zEnterprise EC12 or later machine.
- If ICSF is started without any REMOTEDEVICE entries specified in the ICSF installation options data set or while running on a machine type other than an IBM zEnterprise EC12 or later machine, the command fails, and ICSF issues message CSFM670I.

SN=serialnumber

Specify the serial number or numbers of the device or devices to be deleted. The *serialnumber* value can be a single serial number or a list of serial numbers separated by commas. When more than one value is provided, the set of values must be enclosed in parentheses. For example:

```
SN=99AE6012
SN=(99AE6012,99AE6013,99AE6014)
```

INDEX=indexlist

Specify the index or indexes of the device or devices to be deleted. Specify a number between 1 and 16, inclusive. The *indexlist* value can be a single device index, a range of indexes separated by a colon, or a combination of the two separated by commas. When more than one value is provided, the set of values must be enclosed in parentheses. For example:

```
INDEX=01
INDEX=(02:08)
INDEX=(02,04:07,09)
```

DISable

Disables updates for the specified key data set. The valid KDS specifications are:

- CKDS
- PKDS
- TKDS

ENable

Enables updates for the specified key data set. The valid KDS specifications are:

- CKDS
- PKDS
- TKDS

OPTions

Changes the value of an ICSF option. The supported options are:

AUDITKEYLIFECKDS,AUDKLC

Changes one or more options related to lifecycle auditing of CKDS labels and tokens.

LABEL,LAB = YES|NO

YES

Enables key lifecycle auditing of CKDS labels.

NO

Disables key lifecycle auditing of CKDS labels.

TOKEN,TOK = YES|NO

YES

Enables key lifecycle auditing of CKDS tokens.

NO

Disables key lifecycle auditing of CKDS tokens.

Example:

```
AUDITKEYLIFECKDS,LABEL=YES,TOKEN=NO
```

AUDITKEYLIFEPKDS,AUDKLP

Changes one or more options related to lifecycle auditing of PKDS labels and tokens.

LABEL,LAB = YES|NO

YES

Enables key lifecycle auditing of PKDS labels.

NO

Disables key lifecycle auditing of PKDS labels.

TOKEN,TOK = YES|NO

YES

Enables key lifecycle auditing of PKDS tokens.

NO

Disables key lifecycle auditing of PKDS tokens.

Example:

```
AUDKLP, TOK=NO, LABEL=YES
```

AUDITKEYLIFETKDS,AUDKLT

Changes one or more options related to lifecycle auditing of TKDS token objects and session objects.

TOKENOBJ,TOKO = YES|NO

YES

Enables key lifecycle auditing of TKDS token objects.

NO

Disables key lifecycle auditing of TKDS token objects.

SESSIONOBJ,SESSO = YES|NO

YES

Enables key lifecycle auditing of TKDS token objects.

NO

Disables key lifecycle auditing of TKDS token objects.

Example:

```
AUDKLT, TOKO=YES  
AUDKLT, TOKO=YES, SESSO=YES
```

AUDITKEYUSGCKDS,AUDKUC

Changes one or more options related to key usage auditing of CKDS labels and tokens.

LABEL,LAB = YES|NO

YES

Enables key usage auditing of CKDS labels.

NO

Disables key usage auditing of CKDS labels.

TOKEN,TOK = YES|NO

YES

Enables key usage auditing of CKDS tokens.

NO

Disables key usage auditing of CKDS tokens.

INTERVAL,INT = *usginterval*[H|M|S]

The interval over which key usage records are aggregated before being written out to SMF. The time unit may be specified as H – hours, M – minutes, or S – seconds. If the time unit is not specified, the default is S - seconds. The minimum value of *usginterval* is 1 second. The maximum value is 24 hours.

Example:

```
AUDKUC, LABEL=YES, TOK=YES  
AUDKUC, INT=8H
```

AUDITKEYUSGPKDS,AUDKUP

Changes one or more options related to key usage auditing of PKDS labels and tokens.

LABEL,LAB = YES|NO

YES

Enables key usage auditing of PKDS labels.

NO

Disables key usage auditing of PKDS labels.

TOKEN,TOK = YES|NO

YES

Enables key usage auditing of PKDS tokens.

NO

Disables key usage auditing of PKDS tokens.

INTERVAL,INT = *usginterval*[H|M|S]

The interval over which key usage records are aggregated before being written out to SMF. The time unit may be specified as H – hours, M – minutes, or S – seconds. If the time unit is not specified, the default is S - seconds. The minimum value of *usginterval* is 1 second. The maximum value is 24 hours.

Example:

```
AUDITKEYUSGPKDS, LAB=YES, TOKEN=NO  
AUDKUP, LAB=YES, TOKEN=NO, INT=3600
```

AUDITPKCS11USG,AUDP11U

Changes one or more options related to usage auditing of PKCS #11 services.

TOKENOBJ,TOKO = YES|NO

YES

Enables key usage auditing of PKCS #11 token objects.

NO

Disables key usage auditing of PKCS #11 token objects.

SESSIONOBJ,SESSO = YES|NO

YES

Enables key usage auditing of PKCS #11 session objects.

NO

Disables key usage auditing of PKCS #11 session objects.

NOKEY = YES|NO

YES

Enables usage auditing of PKCS #11 services which do not involve an object.

NO

Disables usage auditing of PKCS #11 services which do not involve an object.

INTERVAL,INT = *usginterval*[H|M|S]

The interval over which key usage records are aggregated before being written out to SMF. The time unit may be specified as H – hours, M – minutes, or S – seconds. If the time unit is not specified, the default is S - seconds. The minimum value of *usginterval* is 1 second. The maximum value is 24 hours.

Example:

```
AUDP11U,TOKO=YES,SESSIONOBJ=NO  
AUDP11U,TOKO=YES,SESSIONOBJ=NO,NOKEY=YES,INTERVAL=1440M
```

MKCVLEN = value

Specifies the number of hexadecimal digits to display on the ICSF Coprocessor Hardware Status panel (CSFCMP40) for the verification and hash patterns for the master keys. The patterns are also referred to as key check values. The value may be 2, 3, 4, 5, 6, or ALL. When an integer value is specified, that number of digits will be displayed. When ALL is specified, all digits will be displayed.

This option can be used to be in compliance with the ISO11568 standard for display of the key check values for master keys.

Notes:

- This option corresponds to the MASTERKCVLEN option in the ICSF installation options data set. Be aware that when ICSF is restarted, the value will revert to the value specified by the MASTERKCVLEN option in the ICSF installation options data set.
- This option has no effect on the output of the DISPLAY ICSF,MKS command.

REFRESH

Refreshes supported option parameters whose values have been updated in the current installation options data set listed in the ICSF startup procedure on the CSFPARM DD statement.

Refreshable option parameters are AUDITKEYLIFECKDS, AUDITKEYLIFEPKDS, AUDITKEYLIFETKDS, AUDITKEYUSGCKDS, AUDITKEYUSGPKDS, AUDITPKCS11USG, BEGIN, CHECKAUTH, DEFAULTWRAP, END, FIPSMODE, KEYARCHMSG, KDSREFDAYS, MASTERKCVLEN, MAXSESSOBJECTS, RNGCACHE, SSM, USERPARM, and WAITLIST.

RISEC = interval

Specifies, in seconds, how often a record should be written for a reference date/time change. The values must be between 0 (write a record for every reference) and 2592000 (30 days) seconds. For example:

```
RISEC=300
```

Note: OPTIONS,RISEC corresponds to the KDSREFDAYS option in the ICSF options data set, which can only be specified in full days. When the RISEC option has been used to change the *refdate* interval, the value for KDSREFDAYS on the Installation Options Display panel is set to SETICSF to indicate that the current value has been modified from the value that is set in the installation options dataset.

RPSEC = period

Specifies how often in seconds ICSF hardens *refdate* updates to the appropriate key data set. The value must be between 10 and 3600. For example:

```
RPSEC=30
```

Note: There is no corresponding keyword in the ICSF options data set for the RPSEC option. The value can only be changed using the SETICSF command.

Installation options modified by the SETICSF command are in effect only until ICSF is stopped or restarted. When ICSF is restarted, the installation options will be re-initialized from the ICSF installation options data set. If you want to make the changes permanent, the installation options data set must be manually updated as needed.

RESTART

Restarts specified cryptographic devices. For the specified devices, the work queues are cleared and ICSF runs through normal configuration processing in an attempt to return a device that is in an error state to an active state. This is most appropriate for a device that has had an error such as CARD BUSY. The valid device specification are:

REMOTEdevice

The regional cryptographic server or servers (remote device or devices). REMOTEdevice is optional.

Notes:

- At least one REMOTEDEVICE option must have been specified in the ICSF installation options data set prior to ICSF being started in order for the SETICSF RESTART, REMOTEdevice command to be operational.
- In addition, the current machine type must be an IBM zEnterprise EC12 or later machine.
- If ICSF is started without any REMOTEDEVICE entries specified in the ICSF installation options data set or while running on a machine type other than an IBM zEnterprise EC12 or later machine, the command fails, and ICSF issues message CSFM670I.

SN=serialnumber

Specify the serial number or numbers of the device or devices to be restarted. The *serialnumber* value can be a single serial number or a list of serial numbers separated by commas. When more than one value is provided, the set of values must be enclosed in parentheses. For example:

```
SN=99AE6012
SN=(99AE6012,99AE6013,99AE6014)
```

INDEX=indexlist

Specify the index or indexes of the device or devices to be restarted. The valid range is 0 to 63, or 1-16 when REMOTEdevice is specified. The *indexlist* value can be a single device index, a range of indexes separated by a colon, or a combination of the two separated by commas. When more than one value is provided, the set of values must be enclosed in parentheses. For example:

```
INDEX=01
INDEX=(02:08)
INDEX=(02,04:07,09)
```

Note: To understand how the use of the INDEX value with the SYSPLEX parameter can result in devices with different serial numbers being modified on other systems sharing the KDS, see the explanation of the SYSPLEX parameter.

SYSPLEX(YES or NO)

The SYSPLEX keyword increases the scope of the SETICSF command to all participating members of the sysplex. The SETICSF command is executed locally on the initiating system and then again on each participating member of the sysplex. The output indicates which systems were able to process the request as well as those systems that were not able to process the request due to a lack of support or an error.

Specify SYSPLEX=Yes to execute the command on all systems. When SYSPLEX=YES is specified, the command may affect cryptographic devices on all systems within the sysplex as follows:

- When SN is specified, all cryptographic devices that have the specified serial number or numbers are affected. No other filtering criteria is applied.
- When INDEX is specified instead of SN, additional filtering criteria is applied. Cryptographic devices that do not meet this criteria are skipped:
 - The command will only affect those systems within the sysplex that share the same TKDS via the SYSPLEXTKDS(YES,...) ICSF installation option. This includes the originating system.
 - For each such system, both the index or indexes and serial number or numbers must match that of the system where the command was issued. For example:
 - The command SETICSF DEACT,REMOTE,INDEX=1,SYSPLEX=YES would deactivate the remote device at index 01 on the originating system as well as the remote device at index 01 on any system sharing the KDS provided that the remote device at index 01 on that system represents the same regional cryptographic server (same serial number).

- If the REMOTE keyword is not specified, the use of SYSPLEX with INDEX results in the command action being performed on all devices at that index on the originating system as well as the cryptographic device at index 01 on any system that is sharing the KDS.

For example, the command SETICSF DEACT,INDEX=1,SYSPLEX=YES would deactivate the cryptographic device at index 01 on the originating system as well as the cryptographic device at index 01 on any system sharing the KDS. In this case, it is better to use SN rather than INDEX as the SETICSF DEACT command can affect devices that have different serial numbers when INDEX is used with SYSPLEX=YES and the command is issued without the REMOTE keyword.

Specify SYSPLEX=No to execute the command only on the local (initiating) system. When SYSPLEX=NO is specified or defaulted, the command affects only the remote device connections on the system where the command was issued.

SYSPLEX=No is the default.

Usage Notes

Installation options modified by the SETICSF command are in effect only until ICSF is stopped or restarted. When ICSF is restarted, the installation options will be re-initialized from the ICSF installation options data set. If you want to make the changes permanent, the installation options data set must be manually updated as needed.

For information on how to limit the use of MVS console commands to a specific set of users, see the System Operations topic in [z/OS MVS System Commands](#).

Using different configurations

A central processor complex can have multiple cryptographic features of various types. This topic describes some of the different configurations available.

You can divide your processor complex into PR/SM logical partitions. When you create logical partitions on your processor complex, you use the usage domain index on the Support Element Customize Image Profile page only if you have, or plan to add a cryptographic feature.

The DOMAIN parameter is optional. The number that is specified for the usage domain index must correspond to the domain number you specified with the DOMAIN(n) keyword in the installation options data set – if you specified one. The DOMAIN keyword is required if more than one domain is specified as the usage domain on the PR/SM panels.

A cryptographic feature can be configured and shared across multiple partitions.

Note: The domain assigned to the TKE Host LPAR must be unique if TKE is to control all the coprocessor cards in the environment. No other LPAR can use the domain assigned to the TKE Host.

The maximum number of LPARs depends on your server. The maximum number of usage domains matches the maximum number of LPARs available on the server. A usage domain can be configured to be unique to one LPAR or assigned to different LPARs accessing different cryptographic features. This is illustrated by LPAR 1 and LPAR 3 in [Figure 1 on page 106](#). They are both assigned to usage domain 0, but on two different CEXnAs.

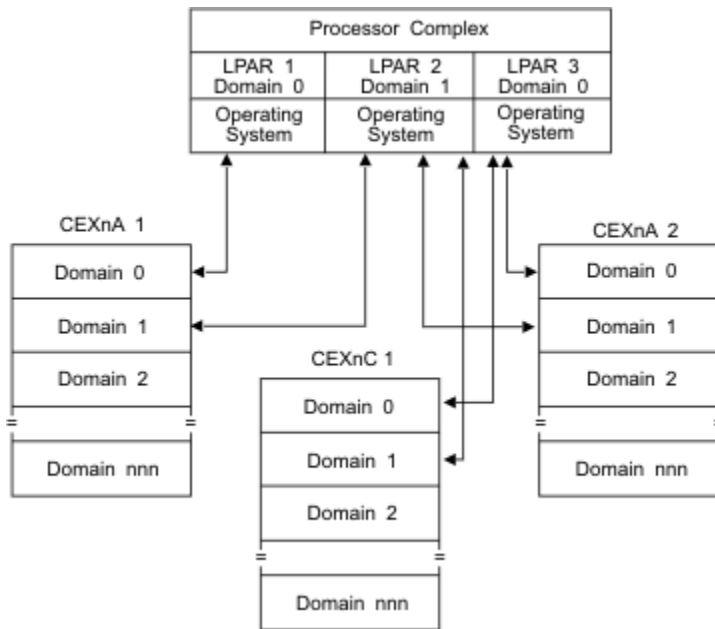


Figure 1. Multiple Crypto coprocessors on a complex

Adding and removing cryptographic coprocessors

It may become necessary for your installation to add or remove cryptographic features. This topic gives you a brief overview of the hardware implications. For more detailed information, refer to the *zSeries PR/SM Planning Guide* and the *zSeries Hardware Management Console Operations Guide (OS/2)*.

There are several terms associated with removing the features. Use the Support Element (SE) panel to configure cryptographic features online and offline (standby). Use the ICSF Coprocessor Management panel from your TSO user ID to activate and deactivate cryptographic features. Use the TKE workstation to enable and disable cryptographic coprocessors.

Adding cryptographic coprocessors

You can dynamically add cryptographic features. You must have feature 3863 installed on your system.

The cryptographic feature number must be in the Candidates list of the LPAR Activation panel. **Configure On** the card. Each feature will display. For coprocessors, once the master keys are entered, they become active. The accelerator will automatically become active.

Note: ALL crypto coprocessors cards must be loaded with the same level of code. Otherwise, unpredictable results can occur. When updating licensed internal code (LIC) on the coprocessors:

- You can migrate to new LIC levels on the coprocessors one at a time without taking an outage, and
- you need to complete the LIC upgrade on all coprocessors before trying to exploit a new function introduced by the new LIC.

Steps for activating/deactivating cryptographic coprocessors

From your TSO userid, select option 1, Coprocessor Mgmt. On the Coprocessor Management panel, you can select the features you want to activate or deactivate.

```

CSFCMP00 ----- ICSF Coprocessor Management ----- Row 1 to 5 of 5
COMMAND ===>                                     SCROLL ==> PAGE

Select the cryptographic features to be processed and press ENTER.
Action characters are: A, D, E, K, R and S. See the help panel for details.

CRYPTO    SERIAL    STATUS    AES    DES    ECC    RSA    P11
FEATURE  NUMBER
-----  -
.  4A00    N/A      Active
.  4P38    97006070 Active
a  4C39    93X06044 Deactivated
d  4C40    93X06077 Master key incorrect  C    C    U    C
.  4C41    93X06071 Active      A    A    A    A
***** Bottom of data *****

```

Figure 2. ICSF coprocessor management

When a coprocessor or accelerator is deactivated through the Coprocessor Management Panel, the card is only deactivated for that one LPAR.

Note: On systems running ICSF FMID HCR77B1 or later, the SETICSF ACTivate and the SETICSF DEACTivate commands can also be used to activate or deactivate coprocessors and accelerators.

Steps to configure on/off cryptographic coprocessors

To configure the cryptographic features online and offline, you must use the support element (SE) panel.

Before configuring a feature offline, it is strongly recommended that you deactivate the feature first from the ICSF Coprocessor Management panel. You need to 'deactivate' the feature in ALL partitions that are using that feature. This allow jobs to complete before the feature is varied offline. You use the **Configure On/Off** service on the Support Element panel to take the feature offline (standby).

After you configure the feature offline from the SE panel, press ENTER on the Coprocessor Management panel to verify that the feature is offline. This configuring is done to remove and replace features or to load new code for the cryptographic features.

To bring a feature back online, use the SE panel again. If a feature was deactivated and then configured offline, you will need to activate it again through the Coprocessor Management panel.

There are no z/OS operator commands to configure the devices online or offline.

Steps for enabling/disabling cryptographic coprocessors

With TKE you can disable/enable coprocessors. When a coprocessor is deactivated through the Coprocessor Management Panel, the coprocessor is only deactivated for that one LPAR. When a coprocessor disabled by TKE, the card is disabled for the entire system, not just the LPAR that issued the disable.

Intrusion latch on the cryptographic coprocessors

Under normal operation, the intrusion latch on a coprocessor is tripped when the feature is removed. This causes all installation data, master keys, retained keys, roles and authorities to be zeroized in the feature when it is reinstalled.

If a situation arises where a coprocessor needs to be removed, for example, you need to remove your feature for service, and you do not want the installation data to be cleared, perform this procedure to disable the coprocessor before removing.

This process will require you to switch between the TKE application, the ICSF Coprocessor Management panel, and the Support Element.

1. Open an Emulator Session on the TKE workstation and logon to your TSO userid on the Host System where the coprocessor will be removed.
2. From the ICSF Primary Option Menu on TSO, select Option 1 for Coprocessor Management.

3. Leave the Coprocessor Management panel displayed during the rest of this procedure. You will be required to press ENTER on the Coprocessor Management panel at different times. DO NOT EXIT this panel.
 4. Open the TKE Host where the coprocessor will be removed. Open the coprocessor. Click on Disable Crypto Module.
 5. After the coprocessor has been disabled from TKE, press ENTER on the Coprocessor Management panel. The status should change to DISABLED.
- Note:** You do not need to deactivate a disabled card.
6. **Configure Off** the coprocessor from the Support Element.
 7. After the card has been taken Offline, press ENTER on the Coprocessor Management panel. The status should change to OFFLINE.
 8. Remove the coprocessor. Perform whatever operation needs to be done. Replace the coprocessor.
 9. **Configure On** the coprocessor from the Support Element.
 10. When the initialization process is complete, press ENTER on the Coprocessor Management panel. The status should change to DISABLED.
 11. From the TKE Workstation Crypto Module General page, click on Enable Crypto Module.
 12. After the coprocessor has been enabled from TKE, press ENTER on the Coprocessor Management panel. The Status should return to its original state. If the Status was ACTIVE in step 2, when the coprocessor is enabled it should return to ACTIVE.

All installation data, master keys, retained keys, roles, and authorities should still be available. The coprocessor data was not cleared with the card removal because it was Disabled first via the TKE workstation.

Adding and removing regional cryptographic servers

Regional cryptographic servers are network-attached, standalone devices or dedicated Linux LPARs that perform geography-specific cryptography. These servers are secure key hardware security modules (HSMs) that operate similar to IBM's PKCS #11 secure coprocessors (CEXnP). They are marketed and serviced by third party vendors. Currently, the only geography-specific cryptography supported by these devices is the Chinese SMx family of algorithms. Secure keys are stored in the TKDS, protected by the Regional Cryptography Server Master Key (RCS-MK).

The network-attached, stand-alone devices require no particular zSeries hardware, but does require communicating with z/OS V1R13 or later and ICSF FMID HCR77B1 or later. ICSF communicates with these devices using TCP/IP, with optional TLS protection. The Linux LPARs require IBM z13 or later hardware. ICSF communicates with the Linux LPARs using TCP/IP, with TLS protection required.

Once configured and online, ICSF makes the algorithms offered by these devices available as PKCS #11 vendor-defined extensions. For information on the algorithms offered, see [*z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*](#) and [*z/OS Cryptographic Services ICSF Application Programmer's Guide*](#).

Steps to add a regional cryptographic server

To enable ICSF to use a regional cryptographic server, you need to complete the following tasks:

1. Follow the vendor's documentation to setup the regional cryptographic server. At a minimum, you must perform the following operations and record the necessary information:

- Establish the TCP/IP hostname or IP address for the device. Record the value: _____
- Configure the device to listen on a specified port. Record the port number: _____

Note: It is important that a unique port be opened for each instance of ICSF that will use the device.

- Determine the optimal number of sockets to be opened by a given instance of ICSF on the port it will be using. Record the number of sockets: _____

- Set the socket inactivity timeout value.

Note: It is important that you configure the device such that the sockets will never time out.

- Set the regional cryptographic server master key (RCS-MK) to be used for the port.

Notes:

- a. All regional cryptographic servers in use by a given instance of ICSF must have the same RCS-MK set on each port used by this particular instance of ICSF.
 - b. All instances of ICSF that share a given TKDS must have the same RCS-MK set on the ports that these ICSF instances will use.
- If TLS protection is desired, configure the device to use TLS. Record whether your device is configured with TLS protection or with no TLS protection: _____

Note: See [“Setup AT-TLS \(optional\)” on page 110](#) if you need to provision the device with an X.509 certificate for use with TLS.

2. Allocate and assign a TKDS to ICSF if one is not already assigned. For additional information, see [“Creating the TKDS” on page 17](#).
3. Configure ICSF to use TCP/IP. See [“Configuring ICSF to use TCP/IP for communications with regional cryptographic servers” on page 109](#) for additional information.

If TLS is desired, optionally generate the certificates necessary using Security Server (RACF) or an equivalent product and provision the regional cryptographic server accordingly.
4. Define the REMOTEDEVICE entry in the ICSF options data set for this regional cryptographic server and restart ICSF.
 - Optionally, you can issue the SETICSF ADD,REMOTEDEVICE console command to dynamically add the regional cryptographic server to ICSF as long as at least one REMOTEDEVICE entry was added to the ICSF options data set prior to the current start of ICSF and your system is running ICSF FMID HCR77B1 or later.

Steps to remove a regional cryptographic server

A regional cryptographic server may be dynamically removed by issuing the SETICSF DELETE,REMOTEDEVICE console command. For additional information on the SETICSF console command, see [“Changing regional cryptographic server status using the SETICSF operator command” on page 112](#).

If the regional cryptographic server being removed was defined to ICSF using a REMOTEDEVICE entry in the ICSF options data set, remove the entry from the options data set to make the removal of this regional cryptographic server permanent.

Configuring ICSF to use TCP/IP for communications with regional cryptographic servers

The Transmission Control Protocol (TCP) and the Internet Protocol (IP) is a protocol suite that allows communications in a network. If you intend to use regional cryptographic servers with ICSF, you must configure ICSF to use TCP/IP. z/OS Communications Server provides the TCP/IP networking protocol on z/OS. It also provides Application Transparent Transport Layer Security (AT-TLS), which allows client and server applications to communicate safely using TCP/IP. While ICSF does not require the use of AT-TLS, it is highly recommended.

For information about configuring TCP/IP, see:

- *z/OS Communications Server: IP Configuration Guide*
- *z/OS Communications Server: IP Configuration Reference*

Note: In an ICSF regional cryptographic server network, ICSF goes outbound to connect to the regional cryptographic server or servers and never listens on a port for incoming connections. Therefore, ICSF always plays the role of a client, not a server, when using TCP/IP.

Steps to configure ICSF to use TCP/IP

You need to perform the following tasks to set up ICSF to use TCP/IP:

- [“Setup the ICSF address space for z/OS UNIX System Services” on page 110](#)
- [“Give the ICSF address space access to the TCP/IP stack” on page 110](#)
- [“Setup AT-TLS \(optional\)” on page 110](#)

Setup the ICSF address space for z/OS UNIX System Services

In order to use TCP/IP, the ICSF started task address space must be DUBBED as a z/OS UNIX System Services process. This requires that the user ID associated with the ICSF started task be assigned a z/OS UNIX UID and that its default group is assigned a GID. There are multiple ways this may be done:

1. Use the BPX.UNIQUE.USER facility to automatically assign permanent OMVS segments (with unique IDs) to user IDs the first time they are needed. This is the preferred way.
2. Use the BPX.NEXT.USER facility in conjunction with the AUTOUID and AUTOGID keywords on the ALTUSER and ALTGROUP commands to add the next available UID and GID to the user ID and group assigned to the ICSF started task.
3. Manually determine and assign the GID and UID to be used and add the appropriate OMVS segment to the user ID assigned to the ICSF started task.

For information about assigning UIDs and GIDs, see [z/OS Security Server RACF Security Administrator's Guide](#).

Give the ICSF address space access to the TCP/IP stack

Stack access control provides a way to allow or disallow users or groups of users to access a TCP/IP stack. The TCP/IP stack to be protected is represented by the resource EZB.STACKACCESS.sysname.tcpname in the SERVAUTH class. If you have a SAF profile protecting this resource, the user ID assigned to the ICSF started task must have access to it.

For more information, see the topic on stack access control in *z/OS Communications Server: IP Configuration Guide*.

Default Key Label Checking

If your configuration has the Default Key Label Checking controls for Key Store Policy enabled and the CHECKAUTH(YES) option is in effect, you need to permit the ICSF started task ID to the CSF-PKDS-DEFAULT profile in the CSFKEYS class. To determine access to tokens that are not stored in the CKDS or PKDS, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Setup AT-TLS (optional)

If you want to encrypt ICSF's communications with a regional cryptographic server or servers, you need to configure the z/OS Communications Server for AT-TLS. You will also need to configure each related regional cryptographic server to perform the server role for TLS.

Note: ICSF is the client. The steps to do this are specific to the regional cryptographic server. See the vendor provided documentation associated with the regional cryptographic server for more information. At a minimum, the regional cryptographic server needs to be provisioned with a TLS server certificate and its associated certificate authority (CA). How you acquire these certificates is your choice. You may use z/OS Security Server (RACF) or equivalent certificate commands or use the z/OS Public Key Infrastructure Services (PKI). You may also choose to purchase your certificates from a commercial certificate authority.

The following are sample z/OS Security Server (RACF) TSO commands to create the CA and server certificates. The samples assume the following:

- The CA is to be labeled 'Regional Server CA'.
- One server certificate is to be created, where the *hostname* is nacc.company.com.
- The user ID assigned to the ICSF started task is ICSFU.
- All certificates are to be valid from January 1, 2015, through December 31, 2024.

- All certificates will have RSA 2048-bit keys.
- For RACDCERT EXPORT, 'hlq' is to be replaced with the desired data set high-level qualifier.

```
/* Create the CA certificate */
RACDCERT CERTAUTH GENCERT SUBJECTSDN(C('CN') O('Company.com'))
OU('Regional Server CA')) WITHLABEL('Regional Server CA') SIZE(2048)
NOTBEFORE(DATE(2015-01-01)) NOTAFTER(DATE(2024-12-31))

/* Create the server certificate */
RACDCERT SITE GENCERT SUBJECTSDN(C('CN') O('Company.com') CN('nacc.company.com'))
WITHLABEL('nacc.company.com') SIZE(2048) NOTBEFORE(DATE(2015-01-01))
NOTAFTER(DATE(2024-12-31)) SIGNWITH(CERTAUTH LABEL('Regional Server CA'))

/* Export the server certificate and private key to be installed on the regional
server.
The password is 'RegionalServer1' */
RACDCERT SITE EXPORT(LABEL('nacc.company.com')) FORMAT(PKCS12DER)
PASSWORD('RegionalServer1') DSN('hlq.NACC.P12')
```

AT-TLS policy

The following is a sample z/OS Communications Server AT-TLS policy agent configuration file that may be used to enable AT-TLS whenever ICSF connects to a regional cryptographic server:

```
###
### AT-TLS Policy Agent Configuration file for ICSF Regional Crypto
###
TTLRule
{
  JobName
  LocalAddr
  RemoteAddr
  RemotePortRange
  Direction
  Priority
  TTLGroupActionRef
  TTLEnvironmentActionRef
}
TTLGroupAction
{
  TTLEnabled
}
TTLEnvironmentAction
{
  HandshakeRole
  EnvironmentUserInstance
  TTLEnvironmentAdvancedParms
  {
    TLSv1
    TLSv1.1
    TLSv1.2
  }
  TTLSKeyringParms
  {
    Keyring
  }
}
```

Notes:

1. Check the JobName rule to ensure it matches the name of the ICSF started procedure.
2. The keyring name may be changed if desired.

AT-TLS Authorization

The ICSF started task userid must have authorization to the CSFPKE and CSFDSV profiles in the CSFSERV class. See *z/OS Communication Server: IP Configuration Guide* for details on encryption algorithms.

The keyring

The client keyring (named CSF.ICSF.KEYRING in the AT-TLS policy topic) must be created and populated with the certificate authority certificates used to issue the server certificates to the regional cryptographic servers.

The following are sample z/OS Security Server (RACF) TSO commands to provision the keyring using the certificate created in the AT-TLS policy topic:

```
/* Create the keyring under user ID ICSFU */
RACDCERT ID(ICSFU) ADDRING(CSF.ICSF.KEYRING)

/* Add the CA certificate */
RACDCERT ID(ICSFU) CONNECT(CERTAUTH LABEL('Regional Server CA'))
RING(CSF.ICSF.KEYRING)
```

For more information on AT-TLS, see the topic, Application Transparent Transport Layer Security data protection, in *z/OS Communications Server: IP Configuration Guide*.

Displaying cryptographic coprocessor status using the DISPLAY ICSF operator command

Use the DISPLAY ICSF operator command to display information about your regional cryptographic servers if your system is running ICSF FMID HCR77B1 and later. For additional information, see the REMOTEDEVICE keyword in [“Display ICSF” on page 90](#).

Adding a regional cryptographic server using the SETICSF operator command

Use the SETICSF operator command to add a regional cryptographic server if your system is running ICSF FMID HCR77B1 and later. For additional information, see the ADD,REMOTEDEVICE keyword in [“SETICSF” on page 95](#).

Changing regional cryptographic server status using the SETICSF operator command

Use the SETICSF operator command to change the status of your regional cryptographic servers if your system is running ICSF FMID HCR77B1 and later. For additional information, see the REMOTEDEVICE keyword in [“SETICSF” on page 95](#).

Performance considerations for using installation options

You specify installation options in the installation options data set. The CHECKAUTH installation option provides additional security checking, but affects performance.

In ICSF, the Security Server (RACF) always checks non-Supervisor State callers. The CHECKAUTH option allows you to specify whether CSF performs access control checking of Supervisor State and System Key callers. Specify CHECKAUTH(NO) if you do not want CSF to check Supervisor State and System Key callers. Specify CHECKAUTH(YES) if you want CSF to check Supervisor State callers. Checking Supervisor State and System Key callers significantly affects performance.

The SYSPLEXCKDS, SYSPLEXPADS and SYSPLEXTKDS options specify whether sysplex-wide data consistency for the CKDS, PKDS, and TKDS is desired. For a description of the subkeywords, see [“Parameters in the installation options data set” on page 30](#).

The RNGCACHE option specifies whether ICSF should maintain a cache of random numbers for services that return or use random numbers. Specifying RNGCACHE(NO) turns off this caching which will decrease performance for services that use random numbers.

Dispatching priority of ICSF

To avoid performance problems, the dispatching priority of ICSF should be set at least as high as that of the highest task using ICSF.

VTAM session-level encryption

ICSF supports VTAM session-level encryption. VTAM session-level encryption provides protection for messages within SNA sessions, that is, between pairs of logical units that support their respective end users. When this method of protection is in effect, data is enciphered by the originating logical unit and deciphered only by the destination logical unit. Thus, the data never appears in the clear while passing through the network.

ICSF places no restrictions on the addressing mode of calling programs. In particular, when VTAM session-level encryption is used with ICSF, VTAM can use storage greater than 16 megabytes.

System SSL encryption

ICSF supports System SSL encryption on all servers. A cryptographic feature is required. For more information, For more information, see *z/OS Cryptographic Services System SSL Programming*.

Access method services cryptographic option

In compatibility mode, ICSF supports the Access Method Services Cryptographic Option. The option enables the user of the Access Method Services REPRO command to use the Data Encryption Algorithm to encipher data.

The Access Method Services user can use REPRO to encipher data that is written to a data set, and then store the enciphered data set offline. When desired, you can bring the enciphered data set back online, and use REPRO to decipher the enciphered data. You can decipher the data either on the host processor on which it was enciphered, or on another host processor that contains the Access Method Services Cryptographic Option and the same cryptographic key that was used to encipher the data. You can either use ICSF to create the cryptographic keys, or use keys that the Access Method Services user supplies.

With the exception of catalogs, all data set organizations that are supported for input by REPRO are eligible as input for enciphering. Similarly, with the exception of catalogs, all data set organizations supported for output by REPRO are eligible as output for deciphering. The resulting enciphered data sets are always sequentially organized (SAM or VSAM entry-sequenced data sets).

See [Appendix E, “Using AMS REPRO encryption,”](#) on page 389 for more information in using this method.

Remote key loading

The process of remote key loading is loading DES keys to automated teller machines (ATMs) from a central administrative site. Because a new ATM has none of the bank's keys installed, getting the first key securely loaded is currently done manually by loading the first key-encrypting key (KEK) in multiple cleartext key parts. A new standard ANSI X9.24-2 defines the acceptable methods of doing this using public key cryptographic techniques, which will allow banks to load the initial KEKs without having to send anything to the ATMS. This method is quicker, more reliable and much less expensive.

Once an ATM is in operation, the bank can install new keys as needed by sending them enciphered under a KEK it installs at a previous time. Cryptographic architecture in the ATMs is not Common Cryptographic Architecture (CCA) and it is difficult to export CCA keys in a form understood by the ATM. Remote key loading will make it easier to export keys to non-CCA systems without compromising security.

In order to use ATM Remote Key Loading, TKE users will have to enable the access control points for these functions:

- Trusted Block Create - API Keyword = Inactive
- Trusted Block Create - API Keyword = Active
- Public Key Import - Source Key Token = Trusted Block
- Public Key Import - Source Key Token = PKA96 Key Token
- Remote Key Export

Event recording

ICSF records certain ICSF events in the System Management Facilities (SMF) data set. ICSF also sends messages that are generated during processing to the ICSF job log and consoles. The SMF recording and messages help you detect problems and track events. This topic describes the events that ICSF records in the SMF record and describes where ICSF sends certain messages.

These records can be used with RACF SMF type 80 record to audit use of the callable services and the keys. The RACF type 80 records are extracted and formatted using the RACF SMF Unload Utility. See [*z/OS Security Server RACF Auditor's Guide*](#) for information on how to use this utility. For information about the formatted SMF records see [*z/OS Security Server RACF Macros and Interfaces*](#).

System Management Facilities (SMF) recording

ICSF uses SMF record type 82 to record certain ICSF events. Record type 82 contains:

- A fixed header / self-defining section: This section contains the common SMF record headers fields and the triplet fields (offset/length/number), if applicable, that locate the other sections on the record.
- A ICSF event specific (subtype) section: Each subtype contains information about the event that caused ICSF to write to the SMF record. For subtypes that log state changes, the SMF record will contain additional auditing sections.
- An auditing header section: This section is present in the record for subtypes that log state changes. It describes the number and overall length of the auditing sections that follow.
- A server user section and, optionally, an end user section: If both sections are present, they can appear in either order.

You can map record type 82 by using the CSFSMF82 macro.

ICSF records information in the SMF data set when these events occur:

- ICSF starts.
- You use the ICSF panels to process an operational key loaded using the TKE workstation.
- The in-storage CKDS is refreshed.
- A dynamic change is made to a record in the CKDS.
- A dynamic change is made to a record in the PKDS.
- You use the ICSF panels to load master keys on a coprocessor.
- An RSA retained key is created or deleted.
- The TKE workstation issues a coprocessor command request or receives a reply response from a coprocessor.
- A cryptographic processor is configured online or offline.
- ICSF records processing times for coprocessors and accelerators.

- ICSF joins or leaves the ICSF sysplex group.
- A trusted block is created or activated.
- A dynamic change is made to a record in the TKDS.
- Duplicate tokens were detected in a key data set.
- The in-storage PKDS is refreshed.
- Key store policy checking detects the unauthorized use of a key token.
- Key store policy PKA key extensions checking detects the unauthorized use of a key.
- A secure symmetric key token is used for CPACF encryption.
- The TKE workstation sends an audit record to ICSF.
- Key store policy checking detects an attempt to use an archived or inactive KDS record.

Each of these events causes ICSF to record information in a separate subtype in the SMF record.

Recording and Formatting type 82 SMF Records in a Report: Sample jobs are available (in SYS1.SAMPLIB) to assist in the recording and formatting of type 82 SMF data:

- CSFSMFJ - JCL that executes the code to dump and format SMF type 82 records for ICSF. Before executing the JCL, you need to make modifications to the JCL (see the prologue in the sample for specific instructions). After the JCL has been modified, terminate SMF recording of the currently active dump dataset (by issuing I SMF) to allow for the unloading of SMF records. After SMF recording has been terminated, execute the JCL. The output goes into the held queue. This is an example of CSFSMFJ.

```
//CSFSMFJ JOB <JOB CARD PARAMETERS>
//*****
//* LICENSED MATERIALS - PROPERTY OF IBM *
//* (C) COPYRIGHT IBM CORP. 2002 *
//* *
//* This JCL reads Type 82 SMF records and formats them in a report.*
//* *
//* CAUTION: This is neither a JCL procedure nor a complete JOB. *
//* Before using this JOB step, you will have to make the following *
//* modifications: *
//* *
//* 1) Add the job parameters to meet your system requirements. *
//* 2) Change the DUMPIN DSN=hlq.smfdata.input to be the name of *
//* the dataset where you currently have SMF data being *
//* recorded. *
//* 3) Change the STEPLIB VOL=SER=ttttt1 and VOL=SER=ttttt2 to *
//* be the volumes where these sort datasets reside. *
//* 4) Change the SYSPROC DSN=hlq.rexx.dataset to be the name of *
//* the dataset where you have placed the CSFSMFR REXX sample. *
//* *
//* Prior to executing this job, you need to terminate SMF *
//* recording of the currently active dump dataset for allow the *
//* unload of SMF records. *
//* *
//*****
//* -----*
//* UNLOAD SMF 82 RECORDS FROM VSAM TO VBS *
//* -----*
//SMFDMP EXEC PGM=IFASMFDP
//DUMPIN DD DISP=SHR,DSN=hlq.smfdata.input
//DUMPOUT DD DISP=(NEW,PASS),DSN=&&VBS,UNIT=3390,
// SPACE=(CYL,(1,1)),DCB=(LRECL=32760,RECFM=VBS,BLKSIZE=4096)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
// INDD(DUMPIN,OPTIONS(DUMP))
// OUTDD(DUMPOUT,TYPE(82))
//*
//* -----*
//* COPY VBS TO SHORTER VB AND SORT ON DATE/TIME *
//* -----*
//COPYSORT EXEC PGM=SORT,REGION=6000K
//STEPLIB DD DISP=SHR,DSN=SYS1.SORTLPA,VOL=SER=ttttt1,UNIT=3390
// DD DISP=SHR,DSN=SYS1.SICELINK,VOL=SER=ttttt2,UNIT=3390
//SYSOUT DD SYSOUT=*
//SORTWK01 DD UNIT=3390,SPACE=(CYL,10)
//SORTIN DD DISP=(OLD,DELETE),DSN=&&VBS
//SORTOUT DD DISP=(NEW,PASS),DSN=&&VB,UNIT=3390,
```

```
//          SPACE=(CYL,(1,1)),DCB=(LRECL=3000,RECFM=VB)
//SYSIN    DD *
SORT FIELDS=(11,4,A,7,4,A),FORMAT=BI,SIZE=E4000
//*
//*-----*
//*    FORMAT TYPE 82 RECORDS                                *
//*-----*
//FMT      EXEC PGM=IKJEFT01,REGION=5128K,DYNAMNBR=100
//SYSPROC  DD DISP=SHR,DSN=hlq.rexx.dataset
//SYSTSPRT DD SYSOUT=*
//INDD     DD DISP=(OLD,DELETE),DSN=&&VB
//OUTDD    DD SYSOUT=*
//SYSTSIN  DD *
          %CSFSMFR
```

- CSFSMFR - An EXEC that formats the SMF type 82 records into a readable report.

ICSF Initialization (Subtype 1)

When ICSF starts, ICSF writes to subtype 1 after initialization is completed. Subtype 1 describes the values of installation options that are specified in the installation options data set.

Subtype 1 contains this information:

- Special secure mode (SSM) option
- Security Server (RACF) checking of Supervisor State and System Key callers (CHECKAUTH) option
- Compatibility mode with CUSP or PCF (COMPAT) option
- Cryptographic domain number (DOMAIN) option
- CKDS name (CKDSN) option
- Maximum length for data in a callable service (MAXLEN) option

Beginning with z/OS V1 R2, the MAXLEN parameter may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647). If a value greater than this is specified, an error will result and ICSF will not start.

- User parameter (USERPARM) option
- PKDS name (PKDSN) option
- TKDS name (TKDSN) option

SMF records for this subtype will also contain a server user audit section.

Operational Key Part Entry (Subtype 7)

ICSF writes to subtype 7 when key parts are entered using the TKE workstation and are processed using the operational key entry ICSF panels. Subtype 7 contains this information:

- The ENC-ZERO verification pattern of the completed key
- A bit indicating whether the verification pattern is valid
- The cryptographic coprocessor domain number
- The cryptographic coprocessor number
- The name of the CKDS that contains the entry with the key part
- The label of the CKDS entry that contains the key part

SMF records for this subtype will also contain server user and end user audit sections.

CKDS Refresh (Subtype 8)

ICSF writes to subtype 8 when the in-storage CKDS is successfully refreshed. ICSF refreshes the in-storage CKDS by reading a disk copy of a CKDS into storage. Subtype 8 contains this information:

- Name of the current in-storage CKDS that ICSF refreshes
- Name of the disk copy of the CKDS that ICSF read into storage to replace the current CKDS

SMF records for this subtype will also contain server user and end user audit sections.

Dynamic CKDS Update (Subtype 9)

ICSF writes to subtype 9 when an application uses the dynamic CKDS update or the KDS metadata write services to write to the CKDS. Subtype 9 contains this information:

- Name of the changed CKDS
- An indication of the operation performed.
- The CKDS entry (which includes the label name and key type) that was changed

SMF records for this subtype will also contain server user and end user audit sections.

Dynamic PKDS Update (Subtype 13)

ICSF writes to subtype 13 when an application uses the dynamic PKDS update or the KDS metadata write services to change the PKDS. Subtype 13 contains this information:

- The name of the changed PKDS
- An indication of the operation performed.
- The name of the changed entry in the PKDS

SMF records for this subtype will also contain server user and end user audit sections.

Cryptographic Coprocessor Clear Master Key Entry (Subtype 14)

ICSF writes to subtype 14 whenever you use ICSF panels to update AES-MK, DES-MK, ECC-MK, or RSA-MK in the new master key register on a coprocessor. Subtype 14 contains this information:

- The master Key valid indicator
- The type of coprocessor
- The new master key verification pattern
- The key part verification pattern
- The cryptographic coprocessor processor number
- The cryptographic coprocessor serial number
- The cryptographic coprocessor domain index

SMF records for this subtype will also contain server user and end user audit sections.

Cryptographic Coprocessor Retained Key Create or Delete (Subtype 15)

ICSF writes to subtype 15 whenever you create or delete a retained private key in a coprocessor. Subtype 15 contains this information:

- The operation performed (created, deleted from coprocessor, deleted from PKDS)
- The type of coprocessor
- The retained key label
- The cryptographic coprocessor processor number
- The cryptographic coprocessor serial number
- The domain index

SMF records for this subtype will also contain server user and end user audit sections.

Cryptographic Coprocessor TKE Command Request or Reply (Subtype 16)

ICSF writes to subtype 16 whenever a TKE workstation either issues a command request to, or receives a reply response from a coprocessor. Subtype 16 contains this information:

- The indicator for request or reply
- The type of coprocessor
- The cryptographic coprocessor processor number

- The cryptographic coprocessor serial number
- The cryptographic coprocessor domain index
- The request command block or reply response block length
- The request command data block or reply response data block length
- The request or reply CPRB
- The length of the fixed audit data
- The number of relocate sections
- The function id
- The function return code
- The function description - describes the function id.

SMF records for this subtype will also contain server user and end user audit sections.

Cryptographic Coprocessor Configuration (Subtype 18)

ICSF writes subtype 18 when a coprocessor or accelerator is brought online or taken offline. Subtype 18 contains this information:

- The operation performed (coprocessor brought online, taken offline)
- The coprocessor number
- The coprocessor serial number, or accelerator number

PCI X Cryptographic Coprocessor Timing (Subtype 19)

ICSF periodically records processing times for PCIXCC operations in subtype 19. Subtype 19 contains this information:

- The time immediately before the operation begins
- The time immediately after the operation ends
- The time immediately after the results of the operation have been communicated to the caller address space
- The number of processes waiting to submit work to the same PCIXCC, domain, and reference slot used by this operation
- The function code for this operation
- The PCIXCC processor number
- The PCIXCC serial number
- The PCIXCC domain
- A reference number that identifies an internal ICSF queue element

Cryptographic Coprocessor Timing (Subtype 20)

ICSF periodically records processing times for coprocessor or accelerator operations in subtype 20. Subtype 20 contains this information:

- The device type
- The time immediately before the operation begins
- The time immediately after the operation ends
- The time immediately after the results of the operation have been communicated to the caller address space
- The number of processes waiting to submit work to the same coprocessor, domain, and reference slot used by this operation
- The function code for this operation
- The coprocessor or accelerator processor number

- The coprocessor or accelerator serial number
- The coprocessor or accelerator domain
- A reference number that identifies an internal ICSF queue element

ICSF Sysplex Group (Subtype 21)

ICSF writes subtype 21 when ICSF joins or leaves the ICSF sysplex group. Subtype 21 contains this information:

- The name of the ICSF sysplex group
- The name of the sysplex member
- An indication of whether the member joined or left the sysplex group
- An indication of whether the join or leave was due to normal initialization/termination processing
- An indication of whether the leave was due to error recovery processing
- The time of the join or leave
- The name of the active CKDS

Trusted Block Create (Subtype 22)

ICSF writes subtype 22 when the Trusted Block Create callable services are invoked. Subtype 22 contains this information:

- Type of call, Active or Inactive
- If a Public Key Section was present in the Trusted Block Token
- ASID of the Caller
- If Input Trusted Block Token is in the PKDS, save it's Label
- If Output Trusted Block Token is in the PKDS, save it's Label
- If the Transport Key Token is in the CKDS, save it's Label

SMF records for this subtype will also contain server user and end user audit sections.

Token Data Set (TKDS) (Subtype 23)

ICSF writes subtype 23 when the Token Data Set (TKDS) record is updated (created, modified, deleted) of PKCS #11 tokens or token objects. Token Data Set callable services are invoked. Subtype 23 contains this information:

- The name of the changed TKDS
- An indication of the operation performed
- The name of the changed entry in the TKDS

SMF records for this subtype will also contain server user and end user audit sections.

Duplicate Key Tokens (Subtype 24)

ICSF writes subtype 24 when the security administrator has indicated that duplicate key tokens must be identified. Subtype 24 contains this information:

- The data set name
- The number of key labels
- The key labels

Key Store Policy Key Token Authorization Checking (Subtype 25)

ICSF writes subtype 25 when a callable service is called and the key token authorization checking detects the key token is not authorized to the caller. The key token is a duplicate of one or more records in the key data set. The check of the CSFKEYS profiles of the record with the key token found the user was unauthorized to use the records. Subtype 25 contains this information:

- Key store and list information.
- The number of key labels.
- The unauthorized duplicate key label and key type.

SMF records for this subtype will also contain server user and end user audit sections.

PKDS Refresh (Subtype 26)

ICSF writes to subtype 26 when the in-storage PKDS is successfully refreshed. ICSF refreshes the in-storage PKDS by reading a disk copy of a PKDS into storage. Subtype 26 contains this information:

- Name of the current in-storage PKDS that ICSF refreshes
- Name of the disk copy of the PKDS that ICSF read into storage to replace the current PKDS

SMF records for this subtype will also contain server user and end user audit sections.

Key Store Policy PKA Key Management Extensions (Subtype 27)

When PKA Key Management Extensions are enabled, ICSF writes to subtype 27 to record operational and error information related to PKA Key Management Extensions. A subtype 27 record is written:

- when a CSF.PKAEXTNS.ENABLE or CSF.PKAEXTNS.ENABLE.WARNONLY profile in the XFACILIT class uses the APPLDATA field to specify a trusted certificate repository, an SMF record is cut to indicate if the trusted certificate repository was successfully changed, or whether there was an error. The APPLDATA field and the repository it specifies will be checked at startup and whenever the XFACILIT class is RACLISTed. ICSF will write a subtype 27 record if the certificate repository is changed, or if there is an error. In this case, subtype 27 will indicate if:
 - the trusted certificate repository was changed
 - the specified trusted certificate repository is empty
 - an error was detected while extracting the APPLDATA
 - the specified repository was not found
 - one or more certificates could not be parsed
- when an application calls a service attempting to use a key in a way that is not allowed by the ICSF segment specifications within the CSFKEYS or XCSFKEY profile that covers the key. The SMF record will be written at the completion of the callable service, which, depending on whether PKA Key Management Extensions had been enabled in warning or fail mode, may or may not allow the requested operation on the key. Subtype 27 contains this information. In this case, subtype 27 will indicate if:
 - an asymmetric key may not be used for the requested function
 - a symmetric key cannot be exported by the provided asymmetric key

SMF records for this subtype will also contain server user and end user audit sections.

High Performance Encrypted Key (Subtype 28)

Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1), Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD and CSNESYD1), Field Level Encipher (CSNBFLE, CSNEFLE), and Field Level Decipher (CSNBFLD, CSNEFLD) callable services exploit CP Assist for Cryptographic Functions (CPACF) for improved key management performance. An encrypted DATA key stored in the CKDS can be used in these services, but only when SYMCPACFWRAP(YES) is specified in the ICSF segment of the CSFKEYS class profile that covers the key. For Field Level Encipher and Field Level Decipher, an encrypted DATA key that is not stored in the CKDS can be used, but only when SYMCPACFWRAP(YES) is specified in the ICSF segment of the CSF-PROTECTED-KEY-TOKEN CSFKEYS class profile. ICSF writes to subtype 28 at the completion of functions that attempt to wrap an encrypted key under the CPACF wrapping key. Subtype 28 will indicate if the rewrapping operation is:

- Permitted for this symmetric key
- Not permitted for this symmetric key

SMF records for this subtype will also contain server user and end user audit sections.

For more information about protected-key CPACF, see [z/OS Cryptographic Services ICSF Overview](#).

TKE Workstation Audit Record (Subtype 29)

If you have the optional TKE Workstation, you can use the TKE Audit Record Upload Configuration Utility to send Trusted Key Entry workstation security audit records to a Z host, where they will be saved in the z/OS System Management Facilities (SMF) dataset. Each TKE security audit record is stored in the SMF dataset as a type 82 subtype 29 record. For more information on the TKE Audit Record Upload Configuration Utility, refer to the [z/OS Cryptographic Services ICSF TKE Workstation User's Guide](#).

Key Store Policy Archived and Inactive Checking (Subtype 30)

ICSF writes subtype 30 when a callable service attempts to use an archived record. ICSF writes subtype 30 when a callable service attempts to use an inactive (outside the key material validity dates) record. Subtype 30 contains this information:

- The reference activity.
- Key data set name.
- The entry that was referenced.

SMF records for this subtype will also contain server user and end user audit sections.

CCA symmetric key lifecycle event (Subtype 40)

ICSF writes subtype 40 whenever a CCA symmetric key undergoes a lifecycle event. A lifecycle event is any event that changes a key, the key's metadata, or the key's state. Examples of lifecycle events include generating a key, updating a key, and a key becoming active. Subtype 40 contains information about the event, information identifying the key, metadata about the key, and information identifying the user. See the AUDITKEYLIFECKDS option in [“Parameters in the installation options data set” on page 30](#) for more details about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

Note: When using the CKDS KEYS utility, key life cycle events are only recorded for ADD functions.

CCA asymmetric key lifecycle event (Subtype 41)

ICSF writes subtype 41 whenever a CCA asymmetric key undergoes a lifecycle event. A lifecycle event is any event which changes a key, the key's metadata, or the key's state. Examples of lifecycle events include generating a key, updating a key, and a key becoming active. Subtype 41 contains information about the event, information identifying the key, metadata about the key, and information identifying the user. See the AUDITKEYLIFECPKDS option in [“Parameters in the installation options data set” on page 30](#) for more details about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

PKCS #11 key lifecycle event (Subtype 42)

ICSF writes subtype 42 whenever a PKCS #11 key undergoes a lifecycle event. A lifecycle event is any event which changes a key, the key's metadata, or the key's state. Examples of lifecycle events include generating a key, updating a key, and a key becoming active. Subtype 42 contains information about the event, information identifying the key, metadata about the key, and information identifying the user. See the AUDITKEYLIFETKDS option in [“Parameters in the installation options data set” on page 30](#) for more details about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

Regional cryptographic server configuration (Subtype 43)

ICSF writes to subtype 43 when a regional cryptographic server is configured online or offline. Subtype 43 contains this information:

- The regional cryptographic server index.
- The regional cryptographic server serial number.

- The regional cryptographic server port number.
- The length of the regional cryptographic server host name.
- The regional cryptographic server host name.

SMF records for this subtype will also contain server user and end user audit sections.

CCA symmetric key usage event (Subtype 44)

ICSF writes subtype 44 whenever a CCA symmetric key is used. Subtype 44 contains information about the event, information identifying the key, metadata about the key, and information identifying the user. See the AUDITKEYUSGCKDS option in [“Parameters in the installation options data set” on page 30](#) for more details about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

CCA asymmetric key usage event (Subtype 45)

ICSF writes subtype 45 whenever a CCA asymmetric key is used. Subtype 45 contains information about the event, information identifying the key, metadata about the key, and information identifying the user. See the AUDITKEYUSGPKDS option in [“Parameters in the installation options data set” on page 30](#) for more details about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

PKCS #11 key usage event (Subtype 46)

ICSF writes subtype 46 whenever a PKCS #11 key is used. Subtype 46 contains information about the event, information identifying the key, metadata about the key, and information identifying the user. See the AUDITPKCS11USG option in [“Parameters in the installation options data set” on page 30](#) for more details about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

PKCS #11 no key usage event (Subtype 47)

ICSF writes subtype 47 whenever a supported PKCS #11 event does not involve a key or object. Subtype 47 contains information about the event and information identifying the user. See the AUDITPKCS11USG option in [“Parameters in the installation options data set” on page 30](#) for more details about enabling these events.

SMF records for this subtype will also contain server user and end user audit sections.

Message recording

ICSF writes messages to the job log, and to the security console and the operator console.

ICSF writes most of its messages to the job log. Messages that demand action from the master console operator will display on the operator console, and messages related to system security will display on the security console. Some of these console messages will appear only on the console, and some will also be written to the job log. Messages that are not displayed on either the operator or security console are written to the job log.

For a description of each ICSF message, see [z/OS Cryptographic Services ICSF Messages](#).

Security considerations

You can provide enhanced security on ICSF by controlling access to resources and changing the values of your keys periodically. This topic describes these aspects of security:

- Controlling access to utility programs - KGUP, CSFDUTIL
- Controlling access to the callable services
- Controlling access to cryptographic keys

- Controlling access to CCA key tokens
- Scheduling changes for cryptographic keys
- Controlling access to panel functions
- Controlling access to RACF SMF log records

Controlling the program environment

Some programs or applications, which use ICSF, require that the environment is program controlled. In a program controlled environment, programs within the address space are defined to the Security Server (RACF). Defining a program to RACF requires the program name and the name of the data set that contains the program.

If there is not already an * or ** profile in PROGRAM class, you must define one using RDEFINE instead of RALTER for the first command.

```
RALTER PROGRAM ** ADDMEM('CSF.SCSFMODE0'/VOLSER/NOPADCHK)
RALTER PROGRAM ** ADDMEM('CSF.SCSFMODE1'/VOLSER/NOPADCHK)
RALTER PROGRAM ** ADDMEM('CSF.SCSFSTUB'/VOLSER/NOPADCHK)
RDEFINE PROGRAM CSF* ADDMEM('SYS1.SIEALNKE'/VOLSER/NOPADCHK)
RDEFINE PROGRAM CSN* ADDMEM('SYS1.SIEALNKE'/VOLSER/NOPADCHK)
```

The VOLSER specification is optional.

For more information, see [z/OS Security Server RACF Security Administrator's Guide](#).

Controlling access to KGUP

Anyone running the key generator utility program can read and alter an unprotected cryptographic key data set (CKDS). Therefore, only authorized users should have access to the key generator utility program. To make it difficult for an unauthorized person to execute the key generator utility program, store the program in an APF-authorized library that is protected by the Security Server (RACF). Additionally, a security administrator can define a CSFKGUP profile in the CSFSERV class and permit or deny users access to the utility.

Controlling access to CSFDUTIL

CSFDUTIL reads through a CKDS or PKDS and generates a report for duplicate secure key tokens. Only authorized users should have permission to access the CKDS or PKDS datasets directly.

Controlling access to the callable services

Unauthorized persons should not perform the cryptographic or key management functions that the callable services provide. The security administrator should be the only one able to access some services like those used in managing keys. The security administrator can give access to some services, such as enciphering and deciphering data, to persons who are authorized on the system.

You can use the Security Server (RACF) to control which users can use ICSF callable services. For example, you can use the key export service to export any type of key. Your installation may want only the security administrator to be able to use the key export function.

ICSF provides security exit points that you can use to control access to a callable service instead of Security Server (RACF). For information about the security exit points, see [“Security installation exits” on page 168](#).

Your installation may want other users to just be able to export data keys, because sending encrypted data between systems is a common function. The data key export callable service permits the export of data keys only. Your security administrator can have access to the key export service and can use the Security Server (RACF) to give other users access to the data key export service. For more information on controlling who can use ICSF callable services, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Access control points for specific functions may be enabled/disabled through the TKE workstation. See the [z/OS Cryptographic Services ICSF TKE Workstation User's Guide](#) for additional information.

Controlling access to cryptographic keys

Besides the key generator utility program and services, your installation should also control access to the cryptographic keys. First, it is highly recommended that you store cryptographic keys in data sets that are protected by RACF or an equivalent product. You should limit access to authorized persons or applications. Second, you can use RACF to control access to keys in the in-storage cryptographic key data set. For more information on protecting cryptographic keys, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

When clear DES or AES keys are added to the CKDS, RACF-protect all clear keys by label name on all systems sharing the CKDS.

ICSF also provides security exit points that you can use to control access to keys in the in-storage CKDS and in the PKDS. For information about the security exit points, see [“Security installation exits” on page 168](#).

Controlling access to secure key tokens

You and your installation have the option of controlling access to a secure tokens that have the same token value and different key labels. To do this, define a key store policy. Key store policy are a system wide setting, using RACF profiles to define the policy. Because key store policy makes use of additional RACF checks, careful planning should occur before implementing the support.

For details on key store policy, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Scheduling changes for cryptographic keys

You should periodically change the value of cryptographic keys to reduce the possibility of exposing a key value. It is recommended that you change the master keys at least every 12 months.

The security administrator can use the key generator utility program (KGUP) to change the cryptographic keys. KGUP updates keys in the disk copy of the cryptographic key data set while the callable services access keys in the in-storage copy of the cryptographic key data set. Therefore, you can change the keys without affecting cryptographic operations. For more information on using KGUP, refer to [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Controlling access to administrative panel functions

You can perform many ICSF administration functions by using the TSO panels. RACF can protect access to these functions. The functions include:

- Refreshing the CKDS or PKDS
- Setting the master keys
- Changing the master keys
- Clear key entry (access can also be controlled through the TKE workstation, domain controls)
- Pass phrase MK/KDS initialization
- Administrative control functions (enabling and disabling dynamic CKDS access, PKA callable services, and dynamic PKDS access)

These functions are treated the same way as callable services. To view and change system status, see [z/OS Cryptographic Services ICSF Administrator's Guide](#) for more information.

Obtaining RACF SMF log records

For information on how to capture SMF log records for RACF access events, see [z/OS Security Server RACF Auditor's Guide](#) and [z/OS Security Server RACF Command Language Reference](#).

You can extract RACF log records from the SMF data set that can be correlated to the ICSF log records. For more information on how to obtain RACF log records from the SMF data set, see [z/OS Security Server RACF Auditor's Guide](#).

Debugging aids

This topic contains information you can use when diagnosing problems on ICSF. This topic describes:

- Component trace
- Abnormal endings
- Using the IPCS formatting routine
- Detecting ICSF serialization contention conditions

Component trace

ICSF Component Trace is configured using a PARMLIB member. A default PARMLIB member, CTICSF00, is shipped and installed with ICSF starting at the ICSF FMID HCR77A1 release level. This PARMLIB member may be specified with the CTRACE option within the ICSF options data set.

Optionally, this PARMLIB member may be copied and customized to a CTICSFxx PARMLIB data set, where xx is a value used to make a copy. The new CTICSFxx PARMLIB member may then be specified at ICSF startup time using the CTRACE option within the ICSF options data set.

Refer to section “Creating an ICSF CTRACE Configuration Data Set” in Chapter 2 for more information on creating a CTICSFxx PARMLIB member.

The TRACEENTRY option in the ICSF Options Data Set has been deprecated. If this option is specified, it will be ignored and will produce a CSFO0212 message.

ICSF Component Trace may also be dynamically updated using the TRACE CT command. A CTICSFxx PARMLIB member may be passed to the TRACE CT command. Specific ICSF Component Trace options may also be specified via replies to the TRACE CT command on the operator console.

Following are examples of how to use the TRACE CT command to specify a CTICSFxx PARMLIB member and individual command options.

To configure ICSF CTRACE to use minimal tracing, use this TRACE OFF command:

```
TRACE CT,OFF,COMP=CSF
```

To specify a new CTICSFxx PARMLIB member, issue this command:

```
TRACE CT,ON,COMP=CSF,PARM=CTICSFxx
```

To specify that you want to trace ASID 0042, issue this command:

```
TRACE CT,ON,COMP=CSF
```

Follow the TRACE ON command with this reply:

```
R nn,ASID=(0042),END
```

To specify that you want to trace JOBNAME MYJOB, issue this command:

```
TRACE CT,ON,COMP=CSF
```

Follow the TRACE ON command with this reply:

```
R nn,JOBNAME=(MYJOB),END
```

Security Considerations

To specify that you want to change the trace buffer size to 250K, issue this command:

```
TRACE CT,250K,COMP=CSF
```

Follow the TRACE command with this reply:

```
R nn,END
```

To specify that you want to change the trace filtering to CARDIO, issue this command:

```
TRACE CT,ON,COMP=CSF
```

Follow the TRACE ON command with this reply:

```
R nn,OPTIONS=(CARDIO),END
```

To display the current active trace options, issue this command:

```
DISPLAY TRACE,COMP=CSF
```

Abnormal endings

ICSF has an abnormal ending in these cases only:

- When an error occurs during ICSF initialization
- When you specify FAIL(ICSF) in the callable service exit installation option
- When the setting of a cryptographic domain index fails

If an abnormal end occurs in any other cases, your application or unit of work ends; however, ICSF is still available.

ICSF has an abnormal end code unique to ICSF. Errors specific to ICSF result in an abnormal end code of X'18F' and a unique reason code. In general, all abnormal ends occurring within ICSF result in an appropriate system dump, user dump, or LOGREC recording.

Review the reason code to see if the abnormal end was an installation or user error. For a list of the reason codes for abnormal end code X'18F', refer to [z/OS MVS System Codes](#). If you cannot resolve the problem, save the dump and contact the IBM Support Center.

IPCS formatting routine

There is a CTrace filter exit for ICSF. You can now issue these IPCS commands:

```
CTRACE COMP(CSF) OPTIONS((COUNTS,FAILURES))  
CTRACE COMP(CSF) OPTIONS((COUNTS))  
CTRACE COMP(CSF) OPTIONS((FAILURES))
```

COUNTS

Produces a list of services called and how often they were called.

FAILURES

Produces output for each failed ICSF service trace entry.

There is a formatter for ICSF called CSFDATA. It is an IPCS VERBEXIT. To run it, enter:

```
VERBX CSFDATA 'options'
```

The supported options are:

- CELL
- CCPA
- CCPS
- CACB

- CCPD

If no options are specified you get VERBX CSFDATA Output:

```
No valid options were specified on VERBX CSFDATA.
Valid options are CELL,CCPA,CCPS,CACB,CCPD
```

You can use the Interactive Problem Control System (IPCS) to format and display the certain ICSF control blocks. The IPCS CBFORMAT command displays the control block's eye-catcher name, its location in the address space, and its field names with their offsets. You specify a symbol with the command to identify the control block. [Table 13 on page 127](#) lists the control blocks you can display, the symbol IPCS recognizes for each control block, and a reference for the control block format.

<i>Table 13. IPCS symbols and format references for the ICSF Control Blocks</i>		
Control Block	Symbol	Format Reference
Installation-defined Service Table	CSFMGST	Varies for each installation.
CSF Exit Name Table	CSFENT	See Table 20 on page 141 .
Cryptographic Communication Vector Table	CSFCCVT	See Table 113 on page 329 .
Cryptographic Communication Vector Table Extension	CSFCCVE	See Table 114 on page 331 .
Secondary Parameter Block	CSFASPB	See Table 24 on page 154 .

For example, to format and display the ICSF Exit Name table issue this command:

```
CBFORMAT CSFENT
```

Instead of using a symbol to identify the control block, you can provide an address. Find and specify the address of the control block in the address space at the time of the dump. When you specify an address, you must also specify the STRUCTURE keyword with the control block symbol.

Note: To format the secondary parameter block, you must provide an address to identify the control block.

For example, if the address of the secondary parameter block is F632D0, issue this command to format the secondary parameter block.

```
CBFORMAT F632D0. STRUCTURE(CSFASPB)
```

In the example, the secondary parameter block is located at address F632D0 in the address space at the time of the dump. On the command, you must put a period after the address. With this control block, you also specify the structure keyword with the symbol CSFASPB.

For more information about using the CBFORMAT command, see [z/OS MVS IPCS User's Guide](#).

Detecting ICSF serialization contention conditions

If a user task or address space holds an ENQ or latch for an extended period of time, it is likely hung and needs to be cancelled so that other work can obtain the ENQ or latch. Some applications might provide controls or document procedures for addressing situations in which the application appears to be gating the rest of the workload. The ICSF system programmer should consult the application's system programmer or administrator regarding actions to take for or against the application. Such action could include stopping or canceling the application.

ICSF requires Global Resource Serialization (GRS) ENQ resources to manage concurrent operations involving the key data sets (CKDS, PKDS and TKDS), and the ICSF ENQ scheme has ICSF itself obtaining any necessary data set ENQ, in a proxy fashion, on behalf of an application unit of work driving an ICSF

API request requiring an ENQ. ICSF also manages any set of additional, different application requests that may be waiting for that same ENQ resource. For this reason, GRS always perceives only ICSF as a key data set ENQ resource owner or waiter, and a DISPLAY GRS,CONTENTION command would not illustrate key data set ENQ contention between two or more competing application requests within a single system scope. For sysplex scope ENQ contention, DISPLAY GRS,CONTENTION would, without any internal assistance, illustrate only ICSF itself as an ENQ holder or waiter, and would not reflect any client application identity or information associated with ICSF's ENQ resource usage.

ICSF provides an internal capability to embellish the DISPLAY GRS command output to illustrate the ICSF client applications for which ICSF is holding an ENQ resource, and on the general conditions involving client waiters for an ENQ resource. This enhanced capability is transparently provided and requires no additional ICSF or GRS installation or configuration action. The ICSF support to enhance the DISPLAY GRS output is relevant on a DISPLAY GRS,CONTENTION command only if GRS can detect contention, which is not the case when two or more ICSF client application requests are competing for the same ENQ resource within a single system scope. The ICSF support is relevant on a DISPLAY GRS,RES=(*qname-rname*) command whenever the ENQ resource specified in the *qname-rname* option is currently held, regardless of whether or not contention exists. For this reason, the DISPLAY GRS,RES=() command version is recommended as the reliable technique for obtaining information about ICSF key data set ENQ serialization conditions. The DISPLAY GRS command syntax for the various ICSF key data set ENQ resources can be summarized as follows:

Table 14. DISPLAY GRS command syntax ICSF key data set ENQ resources	
This command:	Displays ENQ information for the:
DISPLAY GRS,RES=(SYSZCKT.*)	CKDS
DISPLAY GRS,RES=(SYSZPKT.*)	PKDS
DISPLAY GRS,RES=(SYSZTKT.*)	TKDS

Here is sample command output for the DISPLAY GRS,RES=(SYSZCKT.*) command:

```
ISG343I 12.01.33 GRS STATUS 360
S=SYSTEM SYSZCKT SYSZCKT
SYSNAME      JOBNAME      ASID      TCBADDR      EXC/SHR
SY1          CSFJM70 /APPL107    0040/0045    007D8E88 EXCLUSIVE
ADDITIONAL RESOURCE INFORMATION FROM:  ICSF Managed ENQ
Owner: APPL107  TTOKEN: 000001200000000300000003007FF050 Waiters: 005
```

In this example, the display command result illustrates that ICSF on system SY1 started under jobname CSFJM70 and executing in ASID 40, has obtained the CKDS ENQ resource exclusively on behalf of the client application running with a jobname of APPL107 and executing in ASID 45. Furthermore, the APPL107 application unit of work that caused ICSF to obtain this ENQ was the task identified by task token 000001200000000300000003007FF050, and there are five additional application requests on system SY1 that are awaiting access to this ENQ resource.

The DISPLAY GRS,RES=() command must be executed on (or routed to) all of the systems within the scope of a sysplex to obtain the comprehensive understanding of an ICSF key data set ENQ resource.

ICSF also exploits Global Resource Serialization (GRS) latches for serializing resources that are managed within the scope of a single system. In the case of ICSF latches, whenever a client application request requires an ICSF latch for serialization, the latch is obtained under the application's unit of work (not proxied like the ENQ), and therefore the DISPLAY GRS,CONTENTION command will always illustrate the application information for the current latch owner or owners.

The following operational steps are recommended when ICSF serialization contention is suspected as a cause for a workload slowdown or hang:

1. Issue the DISPLAY GRS,CONTENTION command to illustrate sysplex scope contention on ICSF ENQ serialization resources, or system level contention on ICSF latch serialization resources. If the command result demonstrates latch contention, go to step 3. If the command result demonstrates

ICSF key data set ENQ contention and discloses the ENQ owner client application information, go to step 3. If the command result does not demonstrate contention, or does not disclose the ENQ owner client application information, proceed to the next step.

2. Issue the following commands as needed (depending on the key data sets you are using):

```
DISPLAY GRS,RES=(SYSZCKT.*)
DISPLAY GRS,RES=(SYSZPKT.*) Issue this command only if you are utilizing a PKDS
DISPLAY GRS,RES=(SYSZTKT.*) Issue this command only if you are utilizing a TKDS
```

The commands need to be executed either on all systems within a sysplex, or on the local system where the ENQ resource is known to be owned. The command result should disclose the ENQ owner client application information.

3. Initiate an action for or against the client application to end the unit of work on behalf of which ICSF has obtained the ENQ resource. Such action could include stopping or canceling the application.

ENF signals

ICSF sends an ENF signal to listeners in the following situations:

- Whenever ICSF is started.
- Whenever ICSF is terminating.
- Whenever a master key is changed.

Listener exit routines are invoked synchronously. Listeners of these signals should follow the guidelines documented in [*z/OS MVS Programming: Authorized Assembler Services Guide*](#) on coding listener exit routines. In particular, avoid any processing that may take an extended period of time to complete.

Table 15. ICSF ENF codes

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
19	ICSF has encountered a change.	None	<p>A four-byte parameter area.</p> <p>X'00000002' ICSF has started and is ready for requests.</p> <p>X'00000003' ICSF is terminating and will no longer accept requests.</p> <p>X'nn000004' One or more master keys (MKs) have been changed. Byte <i>nn</i> indicates the MKs that have been changed as follows.</p> <p>Bit Meaning when set:</p> <p>0 DES MK changed.</p> <p>1 AES MK changed.</p> <p>2 RSA MK changed.</p> <p>3 ECC MK changed.</p> <p>4 P11 MK changed.</p> <p>5 RCS MK changed.</p> <p>6-7 Reserved.</p>	EXIT or SRBEXIT / NO

Chapter 5. Installation exits

Your installation can define exit routines to supplement the Integrated Cryptographic Service Facility (ICSF), the key generator utility program (KGUP), and the PCF conversion program. Exit routines are programs that programmers at your installation write to allow you to “customize” an application. Your installation may need to perform specific functions with the data that your cryptographic application manipulates. At various points in processing, ICSF, KGUP, and the PCF conversion program release control to an exit routine.

Some common uses for installation exits include:

- Identifying and verifying users
- Accessing alternate data sets
- Manipulating input commands
- Manipulating output data

This topic describes the various types of exit points in ICSF and the functions that your exits can perform.



Attention: Only an experienced system programmer should use the ICSF installation exits. Writing an exit routine and installing a new exit are tasks that require a thorough knowledge of system programming in an OS/390 and z/OS environment. An unknowledgeable programmer who attempts to write exit routines or to install new exit points, runs the risk of seriously degrading the performance of your system and causing complete system failure.

Types of exits

ICSF provides several types of exit points:

- Exits that are called during initialization, stopping, and modification of ICSF itself, which are known as the mainline exits
- Exits that are called from the services
- An exit called when a record is read from or written to a fixed length record CKDS.
- An exit called when you update the CKDS with a key that is entered through the key entry hardware or during conversion program processing
- An exit called when records are retrieved from the in-storage CKDS
- Security exits that are called during initialization and stopping of ICSF, during a call to a service, and when accessing a CKDS entry
- An exit called at various points during KGUP processing

These topics briefly describe the different types of exits available in ICSF.

Note: Although IBM no longer supplies security exit routines, the exit points still remain.

Mainline exits

You can supply three exits that are called during ICSF initialization. You can also define an exit routine to run after an operator issues the STOP command and another exit to run after the MODIFY command. Thus, mainline exits can run at these five different points:

- Initialization points
 - Before ICSF initialization
 - After ICSF reads and interprets the installation options
 - Before the completion of ICSF initialization

- When an operator issues a STOP ICSF command
- When an operator issues a MODIFY ICSF command

You can use a mainline exit to alter values in the Cryptographic Communication Vector Table, to end ICSF, or to change ICSF installation options. For more information about the mainline exits, see [“Mainline installation exits” on page 136](#).

Exits for the services

Each of the services in ICSF calls an exit before and after processing. [z/OS Cryptographic Services ICSF Application Programmer's Guide](#) describes the services in greater detail.

You can use a service exit to change, augment, or replace processing or to bypass the IBM-supplied processing for the service entirely. [“Services installation exits” on page 144](#) gives further details about exits for the services.

The PCF CKDS conversion program exit

The PCF conversion program changes a CKDS from PCF to ICSF CKDS format. See [Chapter 8, “Migration from PCF to z/OS ICSF,” on page 199](#) for more information about the conversion program.

ICSF provides three exit points for the same exit routine:

- During the initialization of the conversion program
- While the conversion program is processing individual records
- During the ending of the conversion program

See [“PCF conversion program installation exit” on page 161](#) for more information about the conversion program installation exit (CSFCONVX).

The single-record, read-write exit

Certain ICSF processes read records from or write records to the CKDS. These processes include running a conversion program, refreshing and reenciphering the CKDS, and using the key entry hardware to enter a key. When these processes read or write CKDS records, they call the exit. You can customize the processing of a CKDS record read-write with the single-record, read-write exit (CSFSRRW). See [“Single-record, Read-write installation exit” on page 164](#) for more information about the single-record, read-write exit.

Note: This exit is given control only for a fixed-length record CKDS. The exit is not given control for the variable-length record format or KDSR format of the CKDS.

The cryptographic key data set entry retrieval exit

You can use certain services to manage keys on ICSF. A service can access a key in the in-storage CKDS by specifying a key label. For more information about the services, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

When a service requests a record from the in-storage CKDS by label, ICSF calls the CKDS entry retrieval exit. For instance, you can use this exit to perform a specific search of the installation data field in the record. See [“Cryptographic key data set entry retrieval installation exit” on page 159](#) for more information about the CKDS entry retrieval exit.

Note: This exit is given control only for a fixed-length record CKDS. The exit does not work with the variable-length record format of the CKDS.

Security exits

You can supply four different exits to control access to resources on ICSF. ICSF calls the security exits at these points:

- During CSF initialization

- During CSF termination
- When an application calls an ICSF service
- When an entry in the in-storage CKDS is accessed

See [“Security installation exits” on page 168](#) for more information about the security exits.

The KGUP exit

You use KGUP to generate and maintain keys in the CKDS. KGUP creates key values that systems can use in key exchanges. The ICSF administrator uses job control language to start KGUP and specifies information to KGUP through the use of a control statement.

As opposed to the five different mainline exits, ICSF provides one exit for KGUP processing that is called at four different points. ICSF calls the KGUP exits at these points:

- During KGUP initialization
- Before KGUP processes a key that is identified by a control statement
- Before KGUP updates the CKDS
- During KGUP termination

The KGUP exit receives a parameter that identifies the exit's calling point. Thus, the installation exit can perform different functions at each of the calls.

You can use the KGUP exit to change key values, make a copy of a CKDS entry, or end KGUP. [“Key generator utility program installation exit” on page 172](#) gives a more detailed description of the KGUP exit.

Entry and return specifications

All of the exits described in [“Types of exits” on page 131](#) use standard linkage conventions on entry and return from the exits.

Registers at entry

The mainline exits have these register contents on entry:

Register

Contents

0

Address of the exit parameter block (EXPB)

1

Address of a parameter list

2–12

Not applicable

13

Address of register save area

14

Return address

15

Entry point address

The service exits have these register contents on entry:

Register

Contents

0

Address of the exit parameter block (EXPB)

1
Address of a parameter list

2–13
Not applicable

14
Return address

15
Entry point address

The CKDS entry retrieval installation exit has these register contents on entry:

**Register
Contents**

0
Not applicable

1
Address of a parameter list

2–12
Not applicable

13
Address of register save area

14
Return address

15
Entry point address

The conversion program, single-record, read-write, and KGUP exits have these register contents on entry:

**Register
Contents**

0
Not applicable

1
Address of a control block (CVXP, RWXP, or KGXP, depending on the exit)

2–12
Not applicable

13
Address of register save area

14
Return address

15
Entry point address

The particular control blocks that are passed through register 0 or register 1 are described with each exit.

Registers at return

Registers for all exits must contain the original contents on entry with the exception of register 15 which must contain a valid return code. See each exit for a list of valid return codes. The registers should contain this information on return.

**Register
Contents**

0–14
Same as entry contents

Exits environment

ICSF calls different types of exits in distinct environments. The exits differ regarding the mode in which they run and how they address data.

Mainline exits

ICSF mainline exits run in task mode in the ICSF address space. All the passed storage pointers specify addresses in the ICSF address space and are not ALET qualified. There are essentially no restrictions on the use of z/OS services for these exits.

Service exits

ICSF calls the service exits in cross memory mode after a space switch PC. The exits run in the ICSF address space, which is the primary address space. The exits need to address parameters in the caller's address space, which is the secondary address space. In general, user-passed parameters, including the parameter list itself, are in the secondary address space. An exit that is running in access register (AR) mode using an ALET of 1 can access these parameters. For information about cross memory mode and AR mode, see [z/OS MVS Programming: Extended Addressability Guide](#).

CKDS entry retrieval exit

The exit runs in cross memory mode. The addresses of the CKDS records that are used by the exit are ALET-qualified. The exit receives both the current CKDS record address and the record's associated ALET as parameters in the exit parameter list. The exit must run in AR mode, and must use the information passed in the exit parameter list to access CKDS entries. For information about cross memory mode and AR mode, see [z/OS MVS Programming: Extended Addressability Guide](#).

KGUP, Conversion Programs, and Single-record, Read-write exits

The exits run in task mode in the caller's home address space. The exits do not run in cross memory mode and are not passed ALET-qualified storage pointers. There are essentially no restrictions on the use of z/OS services for these exits.

Security exits

The initialization and termination security exits run in task mode in the ICSF address space. The passed storage pointers specify an address in the ICSF address space and are not ALET-qualified. There are essentially no restrictions on the use of z/OS services for these exits.

ICSF calls the security service exit and the security keys exit in cross memory mode after a space switch PC. The security service exit runs in the ICSF address space, which is the primary address space. The security key exit runs in cross memory and AR mode.

Exit recovery

An ESTAE routine provides recovery for the mainline exits; the single-record, read-write exit; and the security initialization and termination exits. If an exit ends abnormally, the ESTAE routine intercepts the abnormal ending code and schedules a system dump. If the conversion program exit ends abnormally, the conversion program ends abnormally. If the KGUP exit ends abnormally, KGUP also ends abnormally. ESTAE routines provide recovery for the conversion program and KGUP.

The ICSF Functional Recovery Routine (FRR) provides recovery for the service exits, the CKDS entry retrieval exit, and the security service and key exits. If an exit ends abnormally, the FRR intercepts the abnormal ending code and schedules a system dump.

There are times during ICSF processing that ICSF suppresses dumps. For example, ICSF does not schedule dumps when integrity checking user data. This action avoids the possibility of user errors that can severely affect system performance. However, ICSF does write a record to SYS1.LOGREC if the error occurs.

When writing exits, you may also want to suppress dumps under certain circumstances. You can suppress dumps by setting a bit on in the SPB. This bit, the SPBTERM bit, is the third bit of the flag byte at offset 18 in the SPB. An exit might want to suppress dumps whenever the exit writes user storage. The exit can turn the bit on before the WRITE instruction and turn the bit off again after the instruction.

Mainline installation exits

ICSF begins when an operator issues a START command from the operator console. When ICSF issues this command, the initialization process begins.

After ICSF starts, operators can issue the MODIFY or STOP commands. You can define installation exits to customize ICSF at the initialization, stopping, and modification points.

Purpose and use of the exits

ICSF calls the mainline exits during the startup, modification, and shutdown stages. The exits allow your installation to change the initialization options, issue special messages, and bypass operator commands. This is a description of each point at which ICSF calls mainline exit routines.

CSFEXIT1

ICSF calls this exit after an operator issues a START command, but before any processing takes place. You can use this exit to change the allocation of the installation options data set.

ICSF always calls the exit. If this exit does not exist, ICSF continues normal processing. If this exit exists, ICSF starts it.

CSFEXIT2

ICSF calls this exit during the initialization process after the installation options data set is read and interpreted. You can use this exit to change certain installation options.

CSFEXIT3

ICSF calls this exit just before ICSF initialization is complete. You can use this exit to issue commands to start other cryptographic work.

CSFEXIT4

ICSF calls this exit when an operator issues a STOP command. You can use this exit to decide to allow or disallow the STOP command.

CSFEXIT5

CSFEXIT5 receives the command input block (the string that is entered by the operator), so you can customize CSFEXIT5 to perform any processing you require. ICSF calls this exit when an operator issues a MODIFY command. ICSF provides the MODIFY command exit to allow each installation the flexibility of defining its own command. ICSF does no processing when an operator uses the MODIFY command. The MODIFY command is simply a call to CSFEXIT5.

Environment of the exits

The exits receive control with these characteristics:

- Supervisor state
- Key 0
- APF-authorized

- TCB mode
- Address Space Control mode=access register mode
- AMODE(31) or AMODE(64)

The exit receives control in AMODE(64) if the service was invoked in AMODE(64); otherwise the exit receives control in AMODE(31). If you have a callable service exit for a service which supports invocation by an AMODE(64) caller, once HCR7720 is installed, you should recode your exit to be sure it can handle being invoked in AMODE(64).

- RMODE(ANY)

The exits can change the characteristics during their processing. However, the exits must return to ICSF with the same characteristics as on entry.

Installing the exits

Because ICSF calls CSFEXIT1 before any initialization occurs, the exit is not defined in the same way as the other exits. For all the mainline exits, install the load module that contains the exit into an APF-authorized library. ICSF uses this normal z/OS search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

You must define CSFEXIT2, CSFEXIT3, CSFEXIT4, and CSFEXIT5 in the installation options data set. However, you *must not* define CSFEXIT1 in the installation options data set, and the load module name for the exit must be CSFEXIT1.

To define the exits in the installation options data set, define the ICSF exit point name and load module name on the EXIT keyword in the installation options data set. For information about the installation options data set, see [“Parameters in the installation options data set” on page 30](#). The EXIT keyword has this syntax:

EXIT (ICSF exit point name, load module name, FAIL (options))

The **ICSF exit point name** portion of the keyword refers to the ICSF name for each exit, CSFEXIT2, CSFEXIT3, CSFEXIT4, and CSFEXIT5. The **load module name** is the name of the load module that contains the exit. The name can be any valid name your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded. The valid FAIL options are:

NONE

Initialization continues even if exits cannot be loaded.

SERVICE

Initialization continues even if exits cannot be loaded.

EXIT

Initialization continues even if exits cannot be loaded.

ICSF

End ICSF if exits cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, abnormally ends, and generates an SVC dump when attempting to load the exit.

Input

All mainline exits receive the address of an exit parameter block (EXPB) passed in register 0. Each exit receives the address of an address list passed in register 1. Each address in the list points to a parameter.

[Figure 3 on page 138](#) illustrates the contents of register 0 and EXPB for the mainline exits.

Register 0

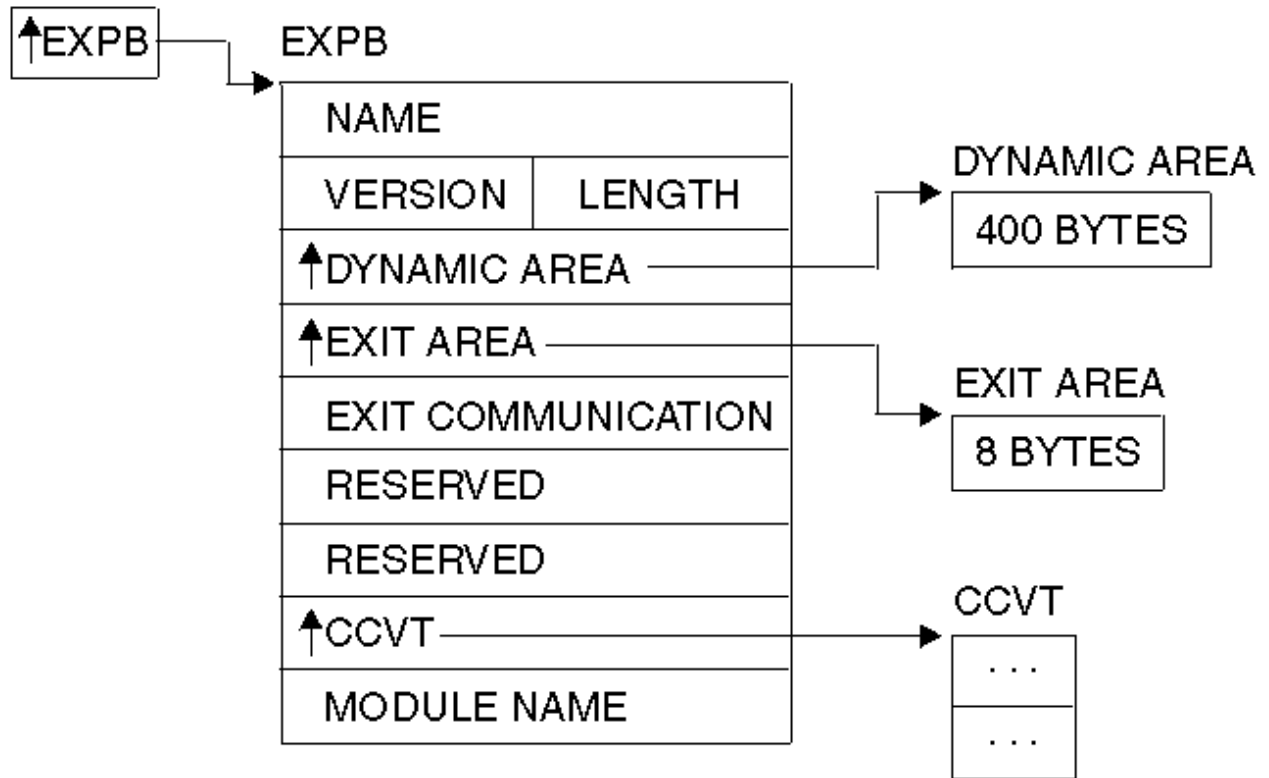


Figure 3. EXPB control block for mainline exits

Both the mainline exits and the services exits receive the address of EXPB in register 0. Some of the fields in EXPB are used only by the service exits and are reserved fields for the mainline exits.

The Exit Parameter Block

Table 16 on page 138 describes the contents of the exit parameter block.

Table 16. EXPB Control Block format for Mainline Exits		
Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. This field contains the character string EXPB.
4	2	Version. The version of the control block. This field contains the character string 01.
6	2	Length. The length of the control block. The value of this field is 40 in decimal.
8	4	Dynamic area address. The address of a 400-byte area that the exit can use as a dynamic area.

Table 16. EXPB Control Block format for Mainline Exits (continued)

Offset (Dec)	Number of Bytes	Description
12	4	Exit area address. The address of an 8-byte area the exits can use to communicate with each other. ICSF does not check or change this field.
16	4	Exit communication area. A character string that can be used for communication between the exits. The field is initialized to zero before CSFEXIT1 is called, and ICSF does not modify this field.
20	4	Flags. Reserved. The flag field is used only by the exits for the services. The field contains binary zeros for the mainline exits.
24	4	Secondary parameter block (SPB) address. Reserved. The SPB is used only by the exits for the services. The field contains binary zeros for the mainline exits.
28	4	CCVT address. Address of the Cryptographic Communication Vector Table (CCVT). “The Cryptographic Communication Vector Table (CCVT)” on page 329 describes the CCVT in greater detail.
32	8	Module name. The installation exit's load module name. The field contains the value of the load module name you specified on the EXIT keyword in the installation options data set. The field is 8 bytes of characters, and the value is left-justified and padded with blanks.

Parameters

All mainline exits receive an address list that uses standard entry linkage. Register 1 points to the address list. Each address in the list points to a parameter. Tables in the next four topics describe the parameters for each of the mainline exits.

CSFEXIT1

This table describes the parameters for CSFEXIT1:

Table 17. CSFEXIT1 parameters

Parameter	Number of Bytes	Description
1	8	The data set name (DDNAME) of the installation options data set.
2	Variable	The command input block for the START command. The command control block is mapped by IEZCIB.

When ICSF calls this, the Cryptographic Communication Vector Table exists, but the table is not yet complete.

CSFEXIT2 and CSFEXIT3

Both CSFEXIT2 and CSFEXIT3 receive the same parameters. [Table 18 on page 140](#) describes these parameters.

Table 18. CSFEXIT2 and CSFEXIT3 parameters		
Parameter	Number of Bytes	Description
1	44	A character string that is the CKDS name specified in the CKDSN installation option.
2	4	A decimal value that is the maximum length permitted for data passed to services specified in the MAXLEN installation option. Beginning with z/OS V1 R2, the MAXLEN parameter may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647). If a value greater than this is specified, an error will result and ICSF will not start.
3	4	ICSF environmental options. Note: Do not change bits 1 - 5. Byte 1: Bit Meaning When Set On 0 Special secure mode enabled. 1 - 5 Reserved. 6 Security Server (RACF) checking required for authorized callers. 7 PCF coexistence. Bytes 2–4: Reserved
4	4	Address of the exit name table. Table 20 on page 141 describes the exit name table.

CSFEXIT4 and CSFEXIT5

Both CSFEXIT4 and CSFEXIT5 receive the same parameters. [Table 19 on page 140](#) describes these parameters.

Table 19. CSFEXIT4 and CSFEXIT5 parameters		
Parameter	Number of Bytes	Description
1	44	A character string that is the CKDS name specified in the CKDSN installation option.

Table 19. CSFEXIT4 and CSFEXIT5 parameters (continued)

Parameter	Number of Bytes	Description
2	4	A decimal value that is the maximum length permitted for data passed to services specified in the MAXLEN installation option. Beginning with z/OS V1 R2, the MAXLEN parameter may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647). If a value greater than this is specified, an error will result and ICSF will not start.
3	4	ICSF environmental options. Note: Do not change bits 1 - 5. Byte 1: Bit Meaning When Set On 0 Special secure mode enabled. 1 - 5 Reserved. 6 Security Server (RACF) checking required for authorized callers. 7 PCF coexistence. Bytes 2–4: Reserved
4	4	Address of the exit name table. Table 20 on page 141 describes the exit name table.
5	Variable	The command input block. You can use the IEZCIB mapping macro to map the control block.

The Exit Name Table

The exit name table contains a list of all of the exits and their load module names. [Table 20 on page 141](#) describes the format of the exit name table.

Table 20. Format of the Exit Name table

Offset (Dec)	Number of Bytes	Description
0	4	Exit name table ID. The value is always the character string ENT.
4	2	Exit name table version. The value is always the character string 01.
6	2	Length of the exit name table. This value is in decimal.
8	4	Number of entries in the array which is the number of exits ICSF supplies. This value is in decimal.
12	4	Subpool that the exit name table is in.
16	4	Reserved.

Table 20. Format of the Exit Name table (continued)

Offset (Dec)	Number of Bytes	Description
20	4	Reserved.
24	4	Reserved.
28	4	Reserved.
32	8	ICSF exit name 1. This value is a character string.
40	8	Installation load module name 1. This value is a character string.
48	4	<p>Flags.</p> <p>Flag bytes. Only the first two bytes are used; bytes 3 and 4 are reserved.</p> <p>Byte 1:</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0</p> <p>Exit has been requested by the installation.</p> <p>1</p> <p>Exit has been loaded.</p> <p>2</p> <p>Exit is active.</p> <p>3</p> <p>If exit fails, end ICSF.</p> <p>4</p> <p>If exit fails, do not call the exit again.</p> <p>5</p> <p>If exit fails, fail the service.</p> <p>6</p> <p>If exit fails, do nothing.</p> <p>7</p> <p>Exit has failed previously.</p> <p>Byte 2:</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0</p> <p>The exit should be called.</p> <p>1</p> <p>The exit is available to the installation.</p> <p>2</p> <p>If the security exit fails, fail the service.</p> <p>3–7</p> <p>Reserved.</p>
52	4	Address of the exit.
56	4	Reserved.
60	4	Reserved.

Table 20. Format of the Exit Name table (continued)

Offset (Dec)	Number of Bytes	Description
64	8	ICSF exit name 2. This value is a character string.
72	8	Installation load module name 2. This value is a character string.
80	4	Flags. See offset +48 for flag byte definitions.
84	4	Address of the exit.
88	4	Reserved.
92	4	Reserved.
x	8	ICSF exit name a.
x+8	8	Installation load module name a.
x+16	4	Flags. See offset +48 for flags.
x+20	4	Address of the exit.
x+24	4	Reserved.
x+28	4	Reserved.

Return Codes

All mainline exits can pass back a return code in register 15. CSFEXIT1, CSFEXIT2, and CSFEXIT3 support these decimal return codes:

Return Code Description

0
Proceed with initialization.

16
End ICSF.

Any return codes other than those listed cause ICSF to end abnormally.

CSFEXIT4 supports these decimal return codes:

Return Code Description

0
Proceed with the STOP command.

4
Do not allow the STOP command to proceed.

Any return codes other than those listed cause processing of the STOP command to end abnormally.

CSFEXIT5 supports these decimal return codes:

Return Code Description

0
Continue processing.

4

End ICSF.

Any return codes other than those listed cause processing of the MODIFY command to end abnormally.

Services installation exits

ICSF provides services that you can use to perform various cryptographic functions. Examples of these functions include enciphering and deciphering data, generating and verifying message authentication codes, generating and verifying PINs, and dynamically updating the CKDS and PKDS. You can define an installation exit for each of the services to customize processing.

Starting with FMID HCR77B0, ICSF provides a single service exit called CSF_SERVICE_EXIT that gets control for all services. The intent of this exit is for statistics generation. For more information, see [“CSF_SERVICE_EXIT - ICSF callable services exit” on page 157](#).

For a detailed description of the services, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

Use this general format to request a service:

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

Table 21 on page 146 lists the ICSF exit names for each of the services. The parameters that the application passes to a service are known as the service parameter list, and the parameters vary from service to service. [“Parameters” on page 156](#) describes the services parameter lists in more detail.

Purpose and use of the exits

Each of the services has an installation exit. Each installation exit for a service has two exit points:

- **The Preprocessing exit point.** This exit point occurs after an application program calls a service, but before the service starts processing. For example, you can use this exit point to check or change the parameters that the application passes on the call, or to end the call. You can also perform additional security checks.
- **The Postprocessing exit point.** This exit point occurs after the service has finished processing, but before the service returns control to the application program. For example, you can use this exit point to check and change the return code from the service or perform cleanup processing.

Environment of the exits

The exits receive control with these characteristics:

- Supervisor state
- Key 0
- APF-authorized
- TCB or SRB mode
- Cross memory mode
- AR mode
- AMODE(31) or AMODE(64)

The exit receives control in AMODE(64) if the callable service was invoked in AMODE(64); otherwise the exit receives control in AMODE(31). If you have a callable service exit for a service which supports invocation by an AMODE(64) caller, you must recode your exit to be sure it can handle being invoked in AMODE(64).

- RMODE(ANY)

The exits can change the characteristics during their processing. However, the exits must return to their caller with the same characteristics as on entry.

You must write the exits in assembler, because you are in AR and cross memory mode and the addresses of some of the parameters you may access are ALET-qualified. In particular, parameters passed into a service are in the user's address space which you can access with an ALET of 1.

For information about cross memory and AR mode, see [z/OS MVS Programming: Extended Addressability Guide](#).

Installing the exits

You install an exit for a service by installing the load module that contains the exit into an APF-authorized library. ICSF uses this normal search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and the load module name as a value on the EXIT keyword in the installation options data set. For more information about the installation options data set, see [“Parameters in the installation options data set”](#) on page 30. The EXIT keyword has this syntax:

EXIT (ICSF name, load module name, FAIL (options))

The **ICSF name** portion of the keyword refers to the ICSF name for each service exit. Note that the ICSF name for each service exit is the same as its name. Table 21 on page 146 lists the ICSF names for each of the service exits. Table 22 on page 150 lists the ICSF names for each of the compatibility service exits. The **load module name** is the name of the load module that contains the exit. The name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded or it ends abnormally. When the exit fails to load, the valid FAIL options mean:

NONE

Initialization continues. The exit is not available to be called.

EXIT

Initialization continues. The exit is not available to be called.

SERVICE

Initialization continues. The exit is not available to be called.

ICSF

ICSF is ended.

When the exit ends abnormally, the valid FAIL options are:

NONE

No action is taken. The exit can be called again and will end abnormally again.

EXIT

The exit is no longer available to be called again.

SERVICE

The service or program that called the exit is no longer available to be called again.

ICSF

ICSF is ended.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit. If the exit ends abnormally, the service call fails regardless of the fail option you specified. Fail options apply only to subsequent requests for the service.

Note: In this table, CSFPCI (PCI interface) is a part of the product-sensitive programming interface.

<i>Table 21. Services and their ICSF names</i>	
Service	ICSF name
Authentication Parameter Generate	CSFAPG
Ciphertext Translate2	CSFCTT2
Ciphertext Translate2 (with ALET)	CSFCTT3
CKDS Key Record Create	CSFKRC
CKDS Key Record Create2	CSFKRC2
CKDS Key Record Delete	CSFKRD
CKDS Key Record Read	CSFKRR
CKDS Key Record Read2	CSFKRR2
CKDS Key Record Write	CSFKRW
CKDS Key Record Write2	CSFKRW2
Clear Key Import	CSFCKI
Clear PIN Encrypt	CSFCPE
Clear PIN Generate	CSFPGN
Clear PIN Generate Alternate	CSFCPA
Control Vector Translate	CSFCVT
Cryptographic Variable Encipher	CSFCVE
CVV Key Combine	CSFCKC
Data Key Export	CSFDKX
Data Key Import	CSFDKM
Decipher	CSFDEC
Decipher (with ALET)	CSFDEC1
Decode	CSFDCO
Derive ICC MK	CSFDCM
Derive Session Key	CSFDSK
Digital Signature Generate	CSFDSG
Digital Signature Verify	CSFDSV
Diversified Key Generate	CSFDKG
Diversified Key Generate2	CSFDKG2
DK Deterministic PIN Generate	CSFDDPG
DK Migrate PIN	CSFDMP

Table 21. Services and their ICSF names (continued)

Service	ICSF name
DK PAN Modify in Transaction	CSFDPMT
DK PAN Translate	CSFDPT
DK PIN Change	CSFDPC
DK PIN Verify	CSFDPV
DK PRW Card Number Update	CSFDPNU
DK PRW CMAC Generate	CSFDPCG
DK Random PIN Generate	CSFDRPG
DK Regenerate PRW	CSFDRP
ECC Diffie-Hellman	CSFEDH
EMV Scripting Service	CSFESC
EMV Transaction Service	CSFEAC
EMV Verification Functions	CSFEVF
Encipher	CSFENC
Encipher (with ALET)	CSFENC1
Encode	CSFECO
Encrypted PIN Generate	CSFEPG
Encrypted PIN Translate	CSFPTR
Encrypted PIN Translate Enhanced	CSFPTRE
Encrypted PIN Verify	CSFPVR
FPE Decipher	CSFFPED
FPE Encipher	CSFFPEE
FPE Translate	CSFFPET
Generate Issuer MK	CSFGIM
HMAC Generate	CSFHMG
HMAC Generate (with ALET)	CSFHMG1
HMAC Verify (with ALET)	CSFHMV1
HMAC Verify	CSFHMV
ICSF Multi-Purpose Service	CSFMPS
Key Data Set List	CSFKDSL
Key Data Set Metadata Read	CSFKDMR
Key Data Set Metadata Write	CSFKDMW
Key Encryption Translate	CSFKET
Key Export	CSFKEX
Key Generate	CSFKGN

Table 21. Services and their ICSF names (continued)

Service	ICSF name
Key Generate2	CSFKGN2
Key Import	CSFKIM
Key Part Import	CSFKPI
Key Part Import2	CSFKPI2
Key Test	CSFKYT
Key Test2	CSFKYT2
Key Test Extended	CSFKYTX
Key Translate	CSFKTR
Key Translate2	CSFKTR2
MAC Generate	CSFMGN
MAC Generate (with ALET)	CSFMGN1
MAC Generate2	CSFMGN2
MAC Generate3	CSFMGN3
MAC Verify	CSFMVR
MAC Verify (with ALET)	CSFMVR1
MAC Verify2	CSFMVR2
MAC Verify3	CSFMVR3
MDC Generate	CSFMDG
MDC Generate (with ALET)	CSFMDG1
Multiple Clear Key Import	CSFCKM
Multiple Secure Key Import	CSFSKM
One Way Hash Generate	CSFOWH
One Way Hash Generate (with ALET)	CSFOWH1
PCI Interface	CSFPCI
PIN change/unblock	CSFPCU
PKA Decrypt	CSFPKD
PKA Encrypt	CSFPKE
PKA Key Generate	CSFPKG
PKA Key Import	CSFPKI
PKA Key Translate	CSFPKT
PKA Key Token Change	CSFPKTC
PKA Public Key Extract	CSFPKX
PKDS Key Record Create	CSFPKRC
PKDS Key Record Delete	CSFPKRD

Table 21. Services and their ICSF names (continued)

Service	ICSF name
PKDS Key Record Read	CSFPKRR
PKDS Key Record Read2	CSFPRR2
PKDS Key Record Write	CSFPKRW
Prohibit Export	CSFPEX
Prohibit Export Extended	CSFPEXX
Random Number Generate	CSFRNG
Random Number Generate Long	CSFRNGL
Recover PIN From Offset	CSFPFO
Remote Key Export	CSFRKX
Restrict Key Attribute	CSFRKA
Retained Key Delete	CSFRKD
Retained Key List	CSFRKL
Secure Key Import	CSFSKI
Secure Key Import2	CSFSKI2
Secure Messaging for Keys	CSFSKY
Secure Messaging for PINs	CSFSPN
SET Block Compose	CSFSBC
SET Block Decompose	CSFSBD
Symmetric Key Export	CSFSYX
Symmetric Key Export with Data	CSFSXD
Symmetric Key Generate	CSFSYG
Symmetric Key Import	CSFSYI
Symmetric Key Import2	CSFSYI2
Symmetric MAC Generate	CSFSMG
Symmetric MAC Generate (with ALET)	CSFSMG1
Symmetric MAC Verify	CSFSMV
Symmetric MAC Verify (with ALET)	CSFSMV1
TR-31 Export	CSFT31X
TR-31 Import	CSFT31I
Transaction Validation	CSFTRV
Trusted Block Create	CSFTBC
Unique Key Derive	CSFUKD
VISA CVV Service Generate	CSFCSG
VISA CVV Service Verify	CSFCSV

Note:

1. The aliases for the PKA services is CSNDxxx or CSNFxxx.
2. The aliases for the symmetric key services are CSNBxxx or CSNExxx.

<i>Table 22. Compatibility services and their ICSF names</i>	
Compatibility Service	ICSF Name
Encipher under Master Key	CSFEMK
Generate a key	CSFGKC
Import a key	CSFRTC
Cipher/Decipher	CSFEDC

Input

The installation exit for each service gets the address of the exit parameter block (EXPB) in register 0. ICSF obtains and initializes an EXP for every service call. [Figure 4 on page 151](#) illustrates the contents of register 0, and [Table 23 on page 151](#) illustrates the EXPB for the service exits.

Register 1 contains the address of an address list. Each address in the list points to a parameter. “Parameters” on [page 156](#) describes the service parameter list. The parameters the exit receives are the same parameters that are passed on the call to the service. For more information about the parameters for each service, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

Register 0

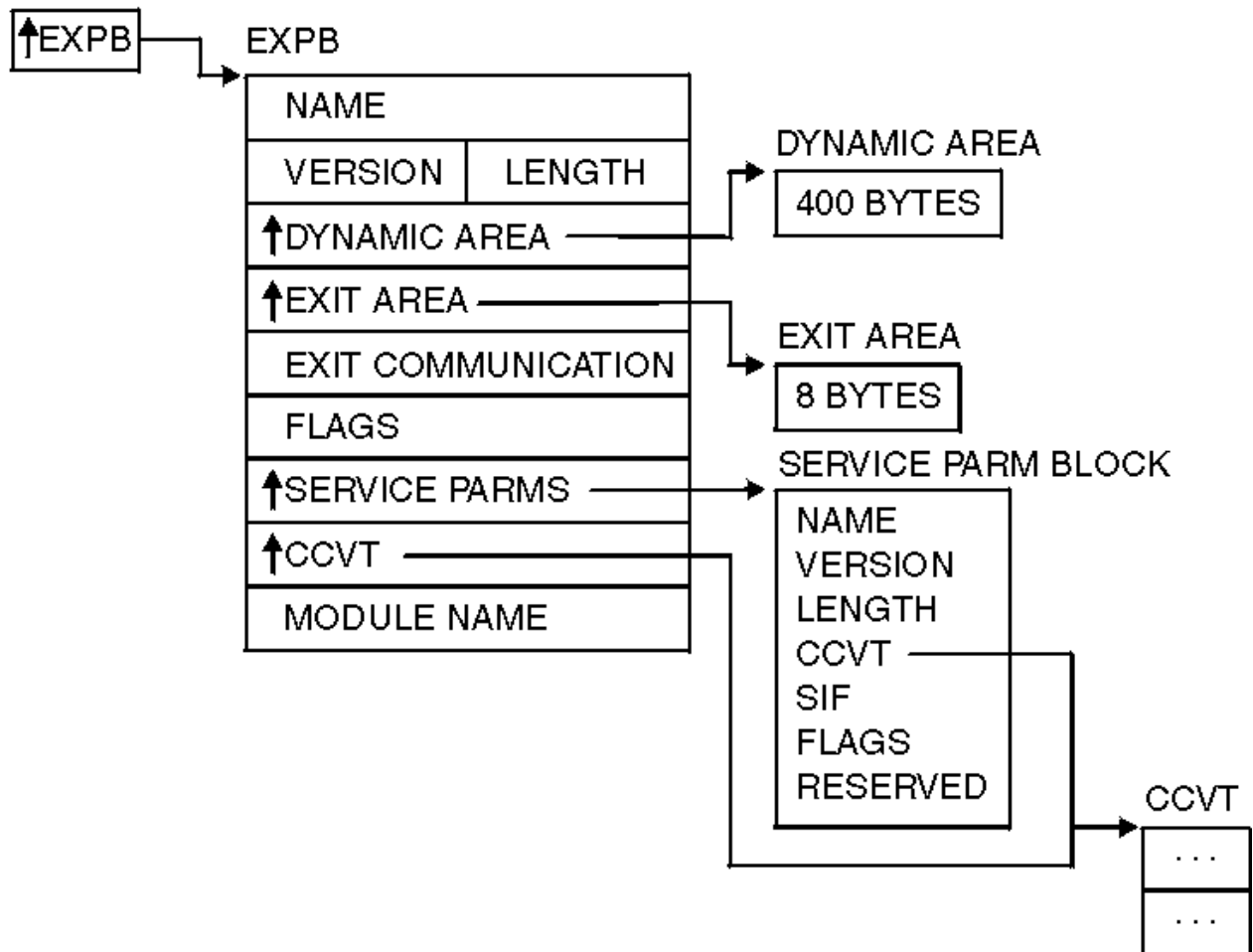


Figure 4. EXPB control block in the service exits

Exit parameter block

Table 23 on page 151 describes the contents of the exit control block.

Table 23. EXPB Control Block Format for Services		
Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. The field contains the character string EXPB.
4	2	Version. The version of the control block. The field contains the character string 01.
6	2	Length. The length of the control block. The value is 40 in decimal.

Table 23. EXPB Control Block Format for Services (continued)

Offset (Dec)	Number of Bytes	Description
8	4	Dynamic area. The address of a 400-byte area that the exit can use as a dynamic area.
12	4	Exit area address. The address of an 8-byte area for the preprocessing and postprocessing invocations of the exit to use for communication. ICSF does not check or change this field.
16	4	Exit communication area. A character string that can be used for communication between preprocessing and postprocessing invocations of a service exit.

Table 23. EXPB Control Block Format for Services (continued)

Offset (Dec)	Number of Bytes	Description
20	4	<p>Flags.</p> <p>A flag byte. Each bit setting (on/off) indicates a particular condition. ICSF sets bit 0 and an exit cannot change that bit. Your exit can set any of the other bits.</p> <p>Bit</p> <p>Meaning When Set On/Off</p> <p>0 Postprocessing invocation./Preprocessing invocation.</p> <p>1 Reserved.</p> <p>2 Use the return and reason code that the exit places in register 0 and register 15 as the service's return code/reason code. Do not use the exit's return code as the service return code in registers 0 and 15.</p> <p>The exit can pass any valid return code in register 15 and any valid reason code in register 0. If this bit is set on, ICSF uses these codes as the service's return and reason codes. See “Return Codes” on page 156 for more information about using exit return codes.</p> <p>3 Do not call the postprocessing invocation of the service exit./Call the postprocessing invocation of the service exit.</p> <p>4 Bypass the service./Run the service.</p> <p>5 Use the return and reason code that the exit places in the service's parameter list./Do not store codes the exit places in the service's parameter list.</p> <p>The exit can pass any valid return and reason code in the first two parameters of the service's parameter list. “Parameters” on page 156 describes the service parameter list.</p> <p>6 CSFSKRC bypass input label parsing./CSFSKRC parse the input label.</p> <p>7–31 Reserved.</p>
24	4	<p>Secondary parameter block.</p> <p>The address of the secondary parameter block. The exit can use the SPB to determine the environmental information of the service. For a description of the SPB, see “Secondary parameter block” on page 154.</p>

Table 23. EXPB Control Block Format for Services (continued)

Offset (Dec)	Number of Bytes	Description
28	4	CCVT. Address of the Cryptographic Control Vector Table (CCVT). For a description of the CCVT, see “The Cryptographic Communication Vector Table (CCVT)” on page 329.
32	8	Module name. The installation exit's load module name. The field contains the value of the load module name you specified on the EXIT keyword in the installation options data set. The field is 8 bytes of characters, and the value is left-justified and padded with blanks.

Secondary parameter block

Offset +24 of EXPB contains the address of the secondary parameter block (SPB). The exit can use the SPB to determine the environmental conditions of the service. [Table 24 on page 154](#) describes the contents of SPB.

Table 24. SPB Control Block Format

Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. The field contains the character string SPB.
4	2	Version. The version of the control block. The field contains the character string 04.
6	2	Length. The length of the control block.
8	4	CCVT. The address of the Cryptographic Communication Vector Table (CCVT). For a description of the CCVT, see “The Cryptographic Communication Vector Table (CCVT)” on page 329.
12	4	Signal Information Word. Bytes 1–2 Reserved. Bytes 3–4 of the field contain the installation-assigned code number for an installation-defined service.

Table 24. SPB Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
16	4	<p>Flags and Indicators. Each byte of this field is either an indicator byte or contains flag bits. The contents of each byte in the field are:</p> <p>Byte 1—PSW key. This byte contains the original caller's program status word key. The first four bits are the key and the remaining four bits are zeros.</p> <p>Byte 2—Caller's state. Each bit in byte 2 indicates a condition of the caller's state.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0</p> <p>ICSF was entered via SVC entry from a PCF compatibility macro.</p> <p>1</p> <p>Original caller in AMODE(31).</p> <p>2</p> <p>Original caller in AR mode.</p> <p>3</p> <p>Original caller in SRB mode.</p> <p>4</p> <p>Original caller in supervisor state or system key.</p> <p>5</p> <p>Original caller in AMODE(64).</p> <p>6–7</p> <p>Reserved.</p> <p>Byte 3—Flag byte 1. The first flag byte. Each bit that is set on indicates a particular condition.</p> <p>Note: These bits are informational. Do not change bits 0 and 1.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0</p> <p>Reserved.</p> <p>1</p> <p>Reserved.</p> <p>2</p> <p>The recovery routine should not retry.</p> <p>3 - 7</p> <p>Reserved.</p> <p>Byte 4—Flag byte 2</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0</p> <p>The service parameter list has a position for a return code.</p> <p>1</p> <p>The service parameter list has a position for a reason code.</p> <p>2</p> <p>Reserved.</p> <p>3</p> <p>The caller has no exit data.</p> <p>4 and 5</p> <p>Reserved.</p> <p>6–7</p> <p>Reserved.</p>
20	4	Reserved.
24	4	Auxiliary SPB Pointer
28	4	EDC buffer pointer.
32	4	EDC buffer length.

Table 24. SPB Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
36	4	Address of XPB.
40	8	ID for latch manager.
48	4	Address for ERPB.
52	8	Original caller's register 1.
60	4	Address of CPRB request storage.
64	4	Length of CPRB request storage.
68	4	Address of CPRB reply storage.
72	4	Length of CPRB reply storage.
76	4	CCPS address.
80	4	Serialization block address.
84	4	Recovery token.
88	8	Recovery footprint for hash tables.
96	4	Reserved.
100	4	Pointer to metal C stack.
104	2	Entry point index of metal C caller.
106	2	Flags and indicators Byte 1 - Reserved. Byte 2 - Saved value of the caller's key.
108	4	ASCB of SPB owner.
112	4	Register 14 from CSFMIREC.
116	4	Reserved.
120	4	ENVR object address.
124	4	ENVR object length.
128	4	Regional cryptographic request block address.
132	20	Reserved.
152	512	CTRACE buffer.

Parameters

Each service has a unique parameter list. Parameters 1–4 are always the return code, reason code, exit data length, and exit data. The other parameters differ with each service. The installation exit gets passed the address of the service parameter list in Register 1. For a description of each service's parameter list, refer to [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

Return Codes

To use a return code and reason code that are set in the postprocessing exit, you must set bit 2 in Offset +20 of EXPB. Setting bit 2 on causes ICSF to return the return code from the exit in register 15 and the reason code in register 0. Even though the application program receives the codes from the exit in the registers, the program still receives the codes from the service in the parameter list. The return code is the first parameter, and the reason code is the second parameter in the list.

Some control languages can access registers more easily than others. For this reason, ICSF allows you to return the return code and the reason code in both the registers and the parameter list. To do this, set bit 5 as well as bit 2 in Offset +20 of EXPB. The application then receives the return code and the reason code from the exit in both the registers and the parameter list.

If you do not set either of or both of the flag bits, the service ignores any return or reason code from the exit. The application program receives the codes from the service in both the registers and the parameter list.

The exit can pass back any valid return code for each service. For a listing of each service's return codes, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

CSF_SERVICE_EXIT - ICSF callable services exit

The ICSF callable services exit CSF_SERVICE_EXIT can be used to generate statistics for all ICSF callable services. This exit point occurs after the callable service finished processing, but before the service returns control to the application program. The intent of this exit is for statistics generation.

Controlling the exit routine through the dynamic exits facility

ICSF defined CSF_SERVICE_EXIT to the dynamic exits facility. You can refer to the exit by the name CSF_SERVICE_EXIT. You can use the EXIT statement of the PROGxx parmlib member, the SETPROG EXIT operator command, or the CSVDYNEX macro to control this exit and its exit routines.

If you do not associate any exit routines with CSF_SERVICE_EXIT in the PROGxx parmlib member, the system defaults to having no exit routine.

To limit the number of times the exit routine abnormally ends before it becomes inactive, use the ADDABENDNUM and ABENDCONSEC parameters on the CSVDYNEX REQUEST=ADD macro or the ABENDNUM and CONSEC parameters of the SETPROG EXIT operator command or of the EXIT statement of the PROGxx parmlib member. An ABEND is counted when both of the following conditions exist:

- The exit routine does not provide recovery or the exit routine does provide recovery, but percolates the error.
- The system allows a retry; that is, the recovery routine is entered with bit SDWACLUP off.

By default, the system disables the exit routine after two consecutive ABENDs.

Exit routine environment

CSF_SERVICE_EXIT receives control in the following environment:

- Enabled for I/O and external interrupts.
- In supervisor state with PSW key 0.
- In AMODE(31) or AMODE(64), depending on the AMODE of the exit routine.
- In AR mode.
- In cross-memory mode with P = ICSF's address space and S = user's address space. Home might or might not equal secondary.
- With no locks held.
- Task or SRB mode.

If the callable service was started in AMODE(64) and an exit routine, which is AMODE(31), needs to access the user's parameters, the exit routine needs to switch to AMODE(64). Regardless of AMODE, the exit routine must not rely on the high 32 bits of any general register having a specific value on entry.

An exit routine can change characteristics (AMODE, ASC mode, locks-held state, cross-memory state, and so on) during its processing. However, the exit routine must return with the same characteristics as on entry. If you plan to access the user's parameters, you must write the exits in a language that can access ALET-qualified variable. This is because you are in AR mode and all the user's parameters, including the parameter list itself, are ALET-qualified. In particular, parameters that are passed into a service are in the user's address space, which you can access with an ALET of 1 (secondary).

Exit recovery

If an exit routine ABENDs and does not have recovery that retries, the system records the error to LOGREC and ICSF calls any exit routines that remain to be called. Whether the exit routine continues to be started depends on the ABEND processing of the dynamic exits facility.

Note: ICSF recommends, for system performance reasons, that the exit not establish recovery unless it modifies critical resources.

Entry specifications

ICSF passes the address of the IXIB (ICSF exit interface block) to exit CSF_SERVICE_EXIT.

The contents of the registers on entry to the exit are as follows.

Register

Contents

R0

N/A

R1

Address of IXIB - ICSF exit interface block.

R2 - R12

N/A

R13

Address of 144-byte save area.

R14

Return address.

R15

Entry point address of exit.

AR0

First 4 bytes of 8-byte PARAM area that is provided by the exit routine owner on CSVDYNEX ADD.

AR1

Second 4 bytes of 8-byte PARAM area that is provided by the exit routine owner on CSVDYNEX ADD.

Parameter list contents: Register 1 contains the address of the ICSF exit interface block (IXIB), which resides in the primary address space (ICSF's address space). The IXIB is mapped by macro CSFZIXIB and the layout is shown in [Table 25 on page 158](#).

Table 25. IXIB control block format		
Offset (Dec)	Number of bytes	Description
0	4	Parmlist with a single entry that points to the IXIB.
4	4	EBCDIC ID.
8	2	Version number of this IXIB.
10	1	Flags X'80' Bit = 1 Caller is AMODE(31). X'40' Bit = 1 Caller is AMODE(64).
11	1	PSW key is in the first 4 bits.
12	4	Address of 2048-byte work area.
16	2	IBM assigned service number.
18	2	Installation service number.

Table 25. IXIB control block format (continued)

Offset (Dec)	Number of bytes	Description
20	4	Reserved.
24	8	Service name.
32	8	Original caller's R1.
40	4	Return code from service.
44	4	Reason code from service.
48	16	STCKE value before service called.
64	16	STCKE value after service called.
80	32	Reserved.

Note on original caller's R1

Each ICSF callable service has a unique parameter list. For a description of each service's parameter list, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

Note on exit behavior

Exit routines that are registered to CSF_SERVICE_EXIT must not change any parameters, including the IXIB or anything it points to.

Cryptographic key data set entry retrieval installation exit

The cryptographic key data set entry retrieval installation exit (CSFCKDS) is called when a service requests an entry from the in-storage cryptographic key data set (CKDS) by label. ICSF calls this exit after it finds the record in the CKDS and before it returns the record to the service.

Note: This exit is given control only for a fixed-length record CKDS. The exit does not work with the variable-length record format of the CKDS.

Purpose and use of the exit

The exit point lists the entry that matches a certain label and type. You can use the exit to check fields in a record and decide whether to use the record. The exit sets a return code that specifies whether to use the record or not. Use the *exit_data* parameter in the service to specify what the exit should use as a search value.

For example, you can use the CKDS entry retrieval exit to perform a specific search of the installation data field. An installation can specify whatever it chooses to in the installation data field. The exit can select a record that matches a certain key label and key type. You can check the record and accept or reject it based on the installation data field.

Note: The cryptographic key data set entry retrieval installation exit will not be given control if SYSPLEXCKDS(YES,FAIL(xxx)) is specified in the ICSF installation options data set.

Environment of the exit

The exit receives control with these characteristics:

- Supervisor state
- Key 0
- APF-authorized
- TCB or SRB mode

- AR mode
- AMODE(31)
- RMODE(ANY)
- Cross memory mode

The exit can change the characteristics during its processing. However, the exit must return to its caller with the same characteristics as on entry.

The exit runs in the cross memory mode in the ICSF address space. The CKDS records are ALET-qualified. ICSF supplies the address and the ALET of a CKDS record as parameters to the CKDS retrieval exit.

For information about cross memory mode and AR mode, see [z/OS MVS Programming: Extended Addressability Guide](#).

Installing the exit

Install the CKDS entry retrieval exit by installing the load module that contains the exit into an APF-authorized library. ICSF uses this normal z/OS search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and the load module name on the EXIT keyword in the installation options data set. “Parameters in the installation options data set” on page 30 describes the installation options data set in further detail. The EXIT keyword has this syntax:

EXIT (ICSF name, load module name, FAIL (options))

The **ICSF name** portion of the keyword refers to the ICSF name for the exit. The ICSF name for the CKDS entry retrieval exit is CSFCKDS. The **load module name** is the name of the load module that contains the exit. The name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded or if it ends abnormally. The valid FAIL options are:

NONE

Do not take any action.

EXIT

Do not call this exit again. The exit will not receive control during subsequent attempts at CKDS retrieval.

SERVICE

Fail the service. All subsequent attempts at CKDS entry retrieval fail.

ICSF

End ICSF.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit. If the exit ends abnormally, the attempt at CKDS entry retrieval fails, regardless of the FAIL option you specified. FAIL options only apply to subsequent attempts at CKDS entry retrieval.

Input

The CKDS entry retrieval exit receives the address of an address list passed in register 1. Each address in the list points to a parameter. The address list exists in the ICSF address space, and register 1 is not ALET-qualified.

[Table 26 on page 161](#) describes the parameters for the CKDS entry retrieval exit.

Table 26. The CKDS Entry Retrieval Exit Parameters

Parameter	Description
1	The address of the current CKDS record. See “Cryptographic Key Data Set (CKDS) formats” on page 215 for a description of the CKDS record format.
2	The address of the ALET of the current CKDS record. This record is a fullword address.
3	The address of the record that matches a certain label and type. This value is a fullword integer. The parameter is in the ICSF address space and the exit can access the parameter using an ALET of 0.
4	The address of the record chosen. This value is a fullword integer. The parameter is in the ICSF address space and the exit can access the parameter using an ALET of 0.
5	The address of the exit data length. This value is a fullword integer. The parameter is in the caller's address space, which is the secondary address space, and the exit can access the parameter using an ALET of 1.
6	The address of the exit data. For a description of exit data, see z/OS Cryptographic Services ICSF Application Programmer's Guide . The parameter is in the caller's address space, which is the secondary address space, and the exit can access the parameter using an ALET of 1.
7	The address of the secondary parameter block. See “Secondary parameter block” on page 154 for a description of the secondary parameter block. The parameter is in the ICSF address space and the exit can access the parameter using an ALET of 0.

Return codes

You can pass a return code back in register 15.

The valid decimal return codes are:

Return Code Description

0

Use the record.

4

Do not use the record.

If you specify not to use any of the records that match the search value, ICSF returns control to the application. It returns with return code 12 and reason code 10024, which indicate that the exit rejected all the keys in the search.

PCF conversion program installation exit

Use the PCF conversion program to convert a CKDS from the Programmed Cryptographic Facility (PCF) format to the ICSF format. The conversion program converts each record in the PCF CKDS to the CKDS format that ICSF uses, and then writes the new record to an ICSF CKDS. The conversion program extends the label field to 64 bytes.

An ICSF CKDS record contains an installation data field that you can use to further identify the record. This field can contain any information about a record that your installation would like to use. You can use the conversion program exit to change the information in this field. You can also use the conversion program exit to have the conversion program not place a converted CKDS entry in the ICSF CKDS.

Chapter 8, [“Migration from PCF to z/OS ICSF,”](#) on page 199 contains more information about the PCF conversion program.

Purpose and use of the exit

The PCF conversion program installation exit (CSFCONVX) is called at three points during processing of the conversion program:

- **During conversion program initialization.** This is known as the conversion preprocessing invocation. At this point, you can use the exit to change the ICSF CKDS header record installation data field.
- **During conversion program individual record processing.** This is known as the record processing invocation. At this point, the conversion program is converting the PCF entry but has not yet placed the entry into the ICSF CKDS. You can use the exit to change the installation data field in the entry for the ICSF CKDS. You can also specify that the conversion program not place the entry into the ICSF CKDS.
- **Just prior to conversion program termination.** This is known as the conversion postprocessing invocation. At this point, like the preprocessing exit point, you can use the exit to change the ICSF CKDS header record installation data field.

Environment of the exit

The exit receives control with these characteristics:

- Problem program state.
- APF-authorized
- TCB mode
- Address Space Control mode=primary
- AMODE(31)
- RMODE(ANY)

The exit can change the characteristics during its processing. However, the exit must return to its caller with the same characteristics as on entry.

The exit runs in task mode in the caller's own address space.

Installing the exit

Install the load module that contains the exit into an APF-authorized library. ICSF uses this normal z/OS search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Joblib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and load module name on the EXIT keyword in the installation options data set. For more information about the installation options data set, see [“Parameters in the installation options data set” on page 30](#). The EXIT keyword has this syntax:

EXIT (ICSF name, load module name, FAIL (options))

The **ICSF name** portion of the keyword refers to the ICSF name for the exit. The ICSF name for the conversion program exit is CSFCONVX. The **load module name** is the name of the load module that contains the exit. This name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded. The valid FAIL options are **NONE**, **EXIT**, **SERVICE**, and **CSF**. For the conversion program exit, you can use these options only:

NONE

Initialization continues even if exit cannot be loaded.

ICSF

Initialization ends if exit cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit.

If the exit ends abnormally, the conversion program does also.

Input

ICSF supplies the address of the conversion program exit parameter block (CVXP) in register 2 each time it calls the PCF conversion program exit. The exit does not receive a parameter list. [“Entry and return specifications” on page 133](#) gives a complete list of the registers on entry to the conversion program exit.

[Table 27 on page 163](#) describes the contents of the exit control block.

<i>Table 27. CVXP Control Block Format</i>		
Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. The field contains the character string CVXP.
4	2	Version. The version of the control block. The field contains the character string 01.
6	2	Length. The length of the control block. The value is 28 in decimal.
8	4	Return Code. The value the exit returns. Valid decimal values for this field are: Return Code Description 0 Normal. 4 Do not process the entry. 8 End conversion program.
12	4	Address of the ICSF CKDS installation data area.
16	4	The value in decimal of the length of the ICSF CKDS installation data area.
20	1	Action. Bit 0 is set on if the action was to change an entry on the ICSF CKDS. Bit 0 is set off if the action was to add an entry to the ICSF CKDS. The rest of the bits in this byte are reserved.

Table 27. CVXP Control Block Format (continued)		
Offset (Dec)	Number of Bytes	Description
21	1	<p>Call Point.</p> <p>Indicates the invocation point of the exit. The exit cannot change this field and the conversion program does not use this field on return from the exit. You can determine the invocation point by the bit that is set on.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0 Conversion preprocessing invocation.</p> <p>1 Conversion postprocessing invocation.</p> <p>2 Record processing invocation.</p> <p>3-7 Reserved.</p>
22	6	Reserved.

Return codes

You can pass a return code back to the conversion program in the CVXP control block (offset +8). The exit can use return codes to reject records for conversion processing or end the conversion program.

Return Code Description

- 0**
Normal.
- 4**
Do not process the entry.
- 8**
End conversion program.

Single-record, Read-write installation exit

ICSF provides an exit that is called when a record is read from or written to a CKDS. ICSF calls the single-record, read-write (CSFSRRW) exit under these conditions:

- The PCF conversion program converts a record into ICSF CKDS format. The conversion program calls the exit before it writes a converted record to the ICSF CKDS.
- ICSF reenciphers a disk copy of a CKDS under a new master key. ICSF calls the exit two times during this processing; after ICSF reads a record to reencipher it and before ICSF writes the reenciphered record.
- ICSF refreshes the in-storage copy of a CKDS. ICSF calls this exit after reading a record from the disk copy to place into storage.

Using the exit, you can do such things as prevent the record from being processed, or add user information to the record.

Note: This exit is given control only for a fixed-length record CKDS. The exit does not work with the variable-length record format of the CKDS.

Purpose and use of the exit

The exit receives a parameter block that describes the CKDS record and the action occurring to the record. By setting a return code in the parameter block, the exit may affect the processing of the record. Depending on the return code, one of these actions occurs:

- ICSF continues to read the record.
- ICSF does not read or write the record.
- ICSF does not read or write the entire CKDS.

The parameter block contains the address of the CKDS record. The exit can add information into the installation data field of the record. For integrity reasons, ICSF receives only changes to this particular field. If the exit sets a return code to continue processing, ICSF processes the record with this information.

The KGUP exit, the PCF conversion program exit, and the single-record, read-write exit can add information to the installation data field of the CKDS header record to identify the data set. If the header record installation data field contains information identifying the CKDS, the single-record, read-write exit can check the field to ensure that it is processing the correct data set. If the exit finds that it is processing the wrong CKDS, the exit can set a return code to stop the processing of the entire data set.

You can use the exit to prevent processing of a record. You can check certain fields in the record and specify that the record not be processed. For example, during postprocessing conversion, you can prevent the processing of any record of a certain key type. However, the exit should never prevent processing of a record containing a system key because ICSF uses these keys in its processing. You differentiate a system key record from other key records by its key label. A system key record label contains all binary zeros. All other key labels contain an alphabetic first character with the remaining characters as either alphabetic or numeric.

Environment of the exit

The exit receives control with these characteristics:

- Problem program state
- APF-authorized
- TCB mode
- Address Space Control mode=primary
- AMODE(31)
- RMODE(ANY)

The exit can change the characteristics during its processing. However, the exit must return to ICSF with the same characteristics as on entry.

When the single-record, read-write exit is called, the exit parameter block is in the caller's address space. The exit is loaded in the caller's address space. The caller is either the PCF conversion program, the utility program (CSFEUTIL), or an ICSF panel.

Installing the exit

Install the load module that contains the exit into an APF-authorized library. ICSF uses this search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Joblib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and load module name on the EXIT keyword of the installation options data set. For more information about the installation options data set, see [“Parameters in the installation options data set”](#) on page 30. The EXIT keyword has this syntax:

EXIT (ICSF name, load module name, FAIL (options))

The **ICSF name** portion of the keyword refers to the ICSF name for the exit. The ICSF name for the single-record, read-write exit is CSFSRRW. The **load module name** is the name of the load module that contains the exit. The name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded or ends abnormally. The valid FAIL options are:

NONE

Do not take any action.

EXIT

Do not call this exit again.

SERVICE

Fail the service that called the exit.

ICSF

Fail the service that called the exit.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit. If you specify FAIL(ICSF) and the exit cannot be loaded, ICSF initialization does not continue. If you specify FAIL(ICSF) and the exit ends abnormally, ICSF issues an advisory message that ICSF should be ended.

Input

The single-record, read-write exit receives the address of the address list passed in register 1. The first address in the address list is for the read-write exit parameter block (RWXP). The exit does not receive a parameter list. [“Entry and return specifications”](#) on page 133 gives a complete list of the registers on entry to the single-record, read-write exit.

The RWXP parameter block contains the address of the CKDS record that is being processed and information about the situation in which the exit is called. The exit sets a return code in a field in the block to specify whether the processing should continue. [Table 28 on page 166](#) describes the RWXP control block.

<i>Table 28. RWXP Control Block Format</i>		
Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. The field contains the character string RWXP.
4	2	Version. The version of the control block. The field contains the character string 01.
6	2	Length. The length of the control block. The value of this field is 32 in decimal.

Table 28. RWXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
8	4	<p>Return Code.</p> <p>The value the exit returns. Valid decimal values for this field are:</p> <p>Return Code Description</p> <p>0 Process current CKDS record</p> <p>4 Do not process current CKDS record</p> <p>8 End processing</p>
12	4	Address of the CKDS record.
16	4	The value in decimal of the length of the CKDS record.
20	7	<p>Action.</p> <p>The field is a 7-byte character string describing the action performed on the CKDS record. The field can contain these values:</p> <ul style="list-style-type: none"> • READ • WRITE • DELETE • REWRITE <p>Note that the value of the field is left-justified and padded with blanks.</p>
27	1	<p>Exit Invocation Reason</p> <p>The reason that the exit was invoked. The field relates to only the CKDS and can contain one of these values:</p> <p>2 Refresh of the in-storage CKDS with a disk copy of a CKDS. The value of the Action field is READ.</p> <p>3 Reencipher of the in-storage CKDS from a disk copy of a CKDS. The value of the Action field is READ or WRITE.</p> <p>5 Conversion record postprocessing. The value of the Action field is WRITE.</p> <p>8 Key entry hardware input. The value of the Action field is READ or WRITE.</p>
28	4	Data set type.

Return codes

You can pass a return code back to the single-record, read-write process in the RWXP control block (offset +8). The exit can use the return code to reject records or to end the single record read-write process. These values are valid decimal return codes:

Return Code	Description
--------------------	--------------------

- | | |
|----------|---|
| 0 | Process the current CKDS record. |
| 4 | Do not process the current CKDS record. |
| 8 | End processing. |

Exit points for security installation exits

IBM-supplied security exit routines were removed in ICSF/MVS Version 2 Release 1. The exit points themselves are still available.

Security installation exits

ICSF provides these exit points to control access to the keys in the in-storage CKDS and to the services.

- Security Initialization Exit
- Security Termination Exit
- Security Service Exit
- Security Key Exit

Purpose and use of the exits

There are two groups of security exits. The security initialization exit (CSFESECI) and security termination exit (CSFESECT) are called during ICSF mainline processing to maintain a security communication area that is used by the other security exits.

Next is a description of each point where ICSF calls security exit routines.

Security initialization exit

ICSF calls this exit during initialization just before calling the ICSF mainline exit CSFEXIT. You can use this exit to anchor resource lists, work areas, and other data to the security communication area. The security service exit (CSFESECS) and security key exit (CSFESECK) can be used to control access to resources on ICSF and for logging in SMF the results of any authorization checks that are made. The security initialization exit defined in the options data set is only invoked if CSFESECS, CSFESECK, or both are also defined.

Security termination exit

ICSF calls this exit as the last function when ICSF ends, before deleting all the installation exits. You can use this exit to free whatever is anchored to the security communication area.

Security service exit

ICSF calls this exit when an application uses an IBM-supplied service, before calling any other installation exit that is associated with that service. You can use this exit to control access to a service. Refer to [Table 21 on page 146](#) for a list of services.

Security key exit

ICSF calls this exit when an application uses a key in the in-storage CKDS, before any other installation exit associated with that use of the key is called. You can use this exit to control access to the keys in the CKDS.

Environment of the exits

The security initialization and termination exits receive control with these characteristics:

- Supervisor state
- Key 0
- APF-authorized
- TCB mode
- Address Space Control mode=access register mode
- AMODE(31)
- RMODE(ANY)

The exits can change the characteristics during their processing. However, the exits must return to ICSF with the same characteristics as on entry.

The security service and key exits receive control with these characteristics:

- Supervisor state
- Key 0
- APF-authorized
- TCB mode
- Cross memory mode
- AR mode
- AMODE(31)
- RMODE(ANY)

The exits can change the characteristics during their processing. However, the exits must return to ICSF with the same characteristics as on entry.

Note: The security exits are not called in SRB mode.

Installing the exits

You install the security exits by installing the load module that contains the exit into an APF authorized library. ICSF uses this normal search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Use the EXIT keyword in the installation options data set to define the ICSF name and load module name. For information about the installation options data set, see [“Parameters in the installation options data set”](#) on page 30. The EXIT keyword has this syntax:

EXIT (ICSF name, load module name, FAIL (options))

The **ICSF name** portion of the keyword refers to the ICSF identifier for each exit, CSFESECI, CSFESECT, CSFESECS, and CSFESECK. The **load module name** is the name of the load module that contains the exit. The name can be any valid name your installation chooses. The action that the **FAIL** portion of the EXIT keyword specifies depends on the type of security exit.

For the security initialization and termination exits, the FAIL portion specifies the action ICSF takes if the exit cannot be loaded. The valid FAIL options mean:

NONE

Continue initialization even if exits cannot be loaded.

SERVICE

Continue initialization even if exits cannot be loaded.

EXIT

Continue initialization even if exits cannot be loaded.

ICSF

End ICSF if exits cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit.

If the security initialization exit ends abnormally, ICSF ends. If the security termination exit ends abnormally, ICSF continues to end.

For the security service and key exits, the FAIL portion specifies the action ICSF takes if the exit cannot be loaded or ends abnormally. When the service or key exit is loaded, the valid FAIL options mean:

NONE

Continue initialization even if exits cannot be loaded.

SERVICE

Continue initialization even if exits cannot be loaded.

EXIT

Continue initialization even if exits cannot be loaded.

ICSF

End ICSF if exits cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit.

When the security service exit ends abnormally, the valid FAIL options mean:

NONE

Process subsequent calls to the service as if no abnormal ending occurred. Call the exit for each call of a service.

SERVICE

Fail on subsequent calls to the particular service.

EXIT

Do not call the exit again. Bypass the exit on subsequent calls to any IBM service.

ICSF

End ICSF.

If the security service exit ends abnormally, ICSF ends the service call before performing the service.

When the security key exit ends abnormally, subsequent attempts to access the in-storage CKDS are processed as if no abnormal ending occurred. The exit continues to be called for each access attempt regardless of the FAIL option.

If the security key exit ends abnormally, ICSF ends the attempt to access the CKDS before performing the access.

Input

The security initialization and termination exits receive the address of an 8-byte security communication area in register 1. When ICSF starts, the security initialization exit can use this area as an anchor for resource lists, work areas, or any other data that your service or keys security exits need to check authorizations. When ICSF ends, the security termination exit can free any system resources that are anchored to this area and used by the service or keys security exits. For example, the exit can free storage that is allocated from the common storage area (CSA).

When a call to a service occurs, the security service exit receives the address of an address list passed in register 1. [Table 29 on page 171](#) describes the parameters the exit receives:

<i>Table 29. Parameters received by the Security Service Exit</i>		
Parameter	Number of Bytes	Description
1	8	The security communication area.
2	8	The character string CSFSERV.
3	8	The name of the service being called.

When an attempt to access a CKDS entry occurs, the security key exit receives the address of an address list passed in register 1. [Table 30 on page 171](#) describes the parameters this exit receives:

<i>Table 30. Parameters received by the Security Key Exit</i>		
Parameter	Number of Bytes	Description
1	8	The security communication area.
2	8	The character string representing the SAF class being checked. May be CSFKEYS or XCSFKEY.
3	64	The label of the key entry being accessed.

Register 0 contains the address of the exit parameter block (EXPB). See [Figure 4 on page 151](#) and [Table 23 on page 151](#).

Return codes

All the security exits can pass back a return code in register 15. The security initialization exit supports these decimal return codes:

Return Code **Description**

- 0**
Proceed with initialization.
- 16**
End ICSF.

Any return codes other than those listed cause ICSF to end abnormally.

The security termination exit supports these decimal return codes:

Return Code **Description**

- 0 or 16**
Proceed with termination.

Any return codes other than those listed cause ICSF to end abnormally.

The security service exit supports these decimal return codes:

Return Code **Description**

- 0 or 4**
Proceed with the service call.

Any return codes other than those that are listed cause the service call to fail.

The security key exit supports these decimal return codes:

Return Code	Description
-------------	-------------

0	Proceed with the access of the CKDS entry.
---	--

4	If the second input parameter is CSFKEYS, proceed with the access of the CKDS entry. Otherwise, the access is failed.
---	---

Any return codes other than those that are listed cause the access of the key to fail.

Key generator utility program installation exit

The key generator utility program (KGUP) generates and maintains keys in the cryptographic key data set (CKDS). You can use KGUP to generate or supply a key to update the CKDS. KGUP generates keys to use in key exchange with other systems. ICSF provides an exit for customizing KGUP processing. For information about using KGUP to managing cryptographic keys, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Purpose and use of the exit

You can use the KGUP installation exit (CSFKGUP) to modify records in the CKDS, write copies of records to alternate data sets, or put additional information in the SMF record. There are many other uses for the KGUP exit depending on your installation's needs. Examine the calling points for an exit and the active control block fields at each calling point to determine other applications for the exit.

KGUP calling points

After an ICSF administrator submits a KGUP job for processing, KGUP calls exits at four points in processing:

1. **During KGUP initialization.** This is known as the KGUP preprocessing exit. After the KGUP job begins but before KGUP starts processing a control statement, KGUP calls this exit.

You can use this exit to place additional information in the installation data field of the CKDS header record. You may want to do this if you need to process different cryptographic key data sets differently. You can place information in the installation data field of the record, and then subsequent calls of the exit can use this information as the basis for performing processes.

2. **Before KGUP processes a key that is identified by a control statement.** This is known as the record preprocessing exit. Before KGUP accesses the CKDS to retrieve the key that is requested in the control statement, KGUP calls the exit again.

Note: This call occurs before KGUP accesses the CKDS. If an exit routine alters a key entry at this call, KGUP accesses the CKDS with the altered entry.

You can use this exit to provide additional security for entering clear key values. When a user enters a clear key in a control statement, use the exit to change the value. In this way, the user never knows the actual clear value in the CKDS. For example, a user enters zeros for clear key values. Your exit generates some random number and replaces the user's clear key value. KGUP then processes the exit's random number as the value to write to the CKDS.

3. **Before KGUP updates the CKDS with a key entry.** This is known as the record postprocessing exit. After KGUP processes a key and before KGUP updates the CKDS, KGUP calls the exit a third time.

At this call, the installation exit can change any information in the Key Output Data Set. Changing the Key Output Data Set also enters the changed keys into the Control Statement Output Data Set, if the keys are exportable. You can use this exit to create audit trails.

KGUP will not call the exit for this calling point when the CKDS is in KDSR format.

4. **During KGUP termination.** This is known as the KGUP postprocessing exit. Calls to this exit occur after KGUP completes processing but before KGUP returns control to ICSF.

Note: If an error occurs in exit processing, KGUP does not call the remaining exit invocations. If an error occurs in KGUP processing that does not result in an abnormal ending, KGUP does not call the remaining exit invocations.

Processing in the exit

At each call, the exit receives the address of the KGUP exit parameter block (KGXP) in register 1. The exit can access any of the data in KGXP. The exit can alter some of the fields in KGXP, while others are simply references. Also, the KGUP exit can alter some fields at some calls but not at other calls.

A field in KGXP gives the calling point of the exit. The exit uses this field to determine when to call the exit to perform appropriate processing. [“Input” on page 174](#) gives a more detailed explanation of the KGXP control block, the values it contains, and when an exit can use or change the values.

Environment of the exit

The KGUP calls the exit only in the address space where KGUP is running. The exit receives control with these characteristics:

- Supervisor state
- APF-authorized
- TCB mode
- Address Space Control mode=primary
- AMODE(31)
- RMODE(ANY)

The exit can change the characteristics during its processing. However, the exit must return to its caller with the same characteristics as on entry.

Installing the exit

Install the load module that contains the exit into an APF authorized library. ICSF uses this search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Joblib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and load module name on the EXIT keyword in the installation options data set.

Note: The load module name must not be named CSFKGUP

For more information about the installation options data set, see [“Parameters in the installation options data set” on page 30](#). The EXIT keyword has this syntax:

EXIT (ICSF name, load module name, FAIL (options))

The **ICSF name** portion of the keyword refers to the ICSF name for the KGUP exit. The ICSF name for the KGUP exit is CSFKGUP. The **load module name** is the name of the load module that contains the exit. The name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded. The valid FAIL options are **NONE**, **EXIT**, **SERVICE**, and ICSF. The FAIL options available to the KGUP exit are:

NONE

Initialization continues even if exit cannot be loaded.

ICSF

Initialization ends if exit cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit. If the exit ends abnormally, KGUP also ends abnormally.

Input

At each of the invocation points, the exit receives the address of the KGUP exit parameter block (KGXP) in register 1. The exit does not receive a parameter list. [“Entry and return specifications” on page 133](#) gives a complete list of the registers on entry to the KGUP exit.

The KGUP exit can alter some of the fields in KGXP. Some fields only provide information to the exit and cannot be changed, and some fields do not apply to particular calls to the exit.

[Table 31 on page 174](#) describes the KGXP control block.

<i>Table 31. KGXP Control Block Format</i>		
Offset (Dec)	Number of Bytes	Description
0	4	Block Identifier. The name of the control block. The field must contain the character string KGXP. The exit must not change the value and KGUP does not use the field upon return from the exit.
4	2	Block Version Number. The version of the control block. The field must contain the character string 03. The exit cannot change this field and KGUP does not use this field on return from the exit.
6	2	Block Length. The length of the control block. The decimal value of the field is 408. The exit cannot change the field and KGUP does not use this field on return from the exit.
8	4	Return Code. The return code the exit supplies upon completion. Upon entry, KGUP initializes this field to zeros. The valid decimal return codes for each of the invocation points are: Record Pre- or postprocessing. 0 Normal, continue processing. 4 Reject control statement, but do not end KGUP. 8 End KGUP immediately. KGUP pre- or postprocessing. 0 Normal, continue processing. > 0 End KGUP immediately.

Table 31. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
12	1	<p>Call Point.</p> <p>Indicates the invocation point of the exit. The exit cannot change this field and KGUP does not use this field on return from the exit. You can determine the invocation point by the bit that is set on.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0 KGUP preprocessing invocation.</p> <p>1 KGUP postprocessing invocation.</p> <p>2 Record preprocessing invocation.</p> <p>3 Record postprocessing invocation.</p> <p>4-7 Reserved.</p>
13	1	<p>Options.</p> <p>Indicates the keywords specified on the KGUP control statement. The exit cannot change this field and KGUP does not use the field upon return from the exit. The field is used only during the record preprocessing and postprocessing invocations. You can determine the keywords on the control statement by the bits that are set on.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0 LABEL with multiple values specified.</p> <p>1 RANGE specified.</p> <p>2 KEY specified.</p> <p>3 CLEAR specified.</p> <p>4 SINGLE specified.</p> <p>5 NOCV specified.</p> <p>6 OUTTYPE specified.</p> <p>7 DOUBLEO specified.</p>

Table 31. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
14	1	<p>Verb Type.</p> <p>Indicates the verb used on the KGUP control statement. The exit cannot change this field and KGUP does not use this field on return from the exit. The field is used only for the record preprocessing and record postprocessing invocations. You can determine the verb on the control statement by the bit that is set on.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0 ADD</p> <p>1 UPDATE</p> <p>2 DELETE</p> <p>3 RENAME</p> <p>4 SET</p> <p>5 OPKYLOAD</p> <p>6–7 Reserved.</p>
15	1	<p>KGUP Flags.</p> <p>Indicates the processing conditions encountered by KGUP at the record postprocessing invocation. The exit cannot change this field and KGUP does not use the field upon return from the exit. The field is not used for the KGUP pre- or postprocessing invocations or the record preprocessing invocation. The processing conditions can be determined by examining whether bit 0 is set on.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0 Non-odd parity key was imported.</p> <p>1 Algorithm is AES</p> <p>2 Algorithm is DES</p> <p>3–7 Reserved.</p>

Table 31. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
16	72	<p>Action Key.</p> <p>Contains the key index accessed by the KGUP control statement. The key index consists of the key label and type fields of a CKDS record entry (“Debugging aids” on page 125 describes the CKDS record format in greater detail). The key index is the first 72 bytes of a CKDS record, and the information in the key index is used to differentiate one key from another.</p> <p>The exit can modify the field at the record preprocessing invocation. The field is not used for the KGUP pre- or postprocessing invocation or the record postprocessing invocation.</p> <p>If the exit modifies the field, KGUP uses the modified field to access the CKDS upon return from the exit.</p> <p>Before the record preprocessing invocation, KGUP places this in this field:</p> <ul style="list-style-type: none"> • The key label or key old label from the LABEL or key label from the RANGE keyword of the control statement • The key type from the TYPE keyword of the control statement <p>The exit cannot modify the key label, key old label, or key type.</p>
88	72	<p>Rename Key.</p> <p>Contains the key index used to rename a key when RENAME is the verb on the control statement. The key index consists of the key label and type fields of a CKDS record entry.</p> <p>The exit can modify the field at the record preprocessing invocation. The field is not used for the KGUP pre- or postprocessing or record postprocessing invocations.</p> <p>If the exit modifies the field, KGUP uses the modified field to access the CKDS upon return from the exit.</p> <p>Before the record preprocessing invocation, KGUP places this information in this field:</p> <ul style="list-style-type: none"> • The key new label from the LABEL keyword of the control statement. • The key type from the TYPE keyword of the control statement. <p>The exit cannot modify the key new label or the key type.</p>

Table 31. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
160	72	<p>Transkey key-label1.</p> <p>The key index of the TRANSKEY key-label1 on the KGUP control statement. The key index is the key label and type of the CKDS record entry.</p> <p>The exit can modify the field at the record preprocessing invocation. The field is not used for the KGUP pre- or postprocessing and record postprocessing invocations.</p> <p>If the exit modifies the field, KGUP uses the modified field to access the CKDS upon return from the exit.</p> <p>Before the record preprocessing invocation, KGUP places this information in this field:</p> <ul style="list-style-type: none"> • The key-label1 from the TRANSKEY keyword of the control statement. • The key type. The type is IMPORTER, if keys are supplied; the type is EXPORTER, if keys are not supplied. <p>The exit cannot modify the key-label1 or the key type.</p>
232	72	<p>Transkey key-label2.</p> <p>The key index of the TRANSKEY key-label2 on the KGUP control statement. The key index is the key label and type of the CKDS record entry.</p> <p>The exit can modify the field at the record preprocessing invocation. The field is not used for the KGUP pre- or postprocessing and record postprocessing invocations.</p> <p>If the exit modifies the field, KGUP uses the modified field to access the CKDS upon return from the exit.</p> <p>Before the record preprocessing invocation, KGUP places this information in this field:</p> <ul style="list-style-type: none"> • The key-label2 from the TRANSKEY keyword of the control statement. • The key type. The key type is IMPORTER, if keys are supplied; the type is EXPORTER, if keys are not supplied. <p>The exit cannot modify the key-label2 or the key type.</p>
304	8	<p>The OUTTYPE value, if specified. If no OUTTYPE is specified, this field set to binary zeros.</p>
312	4	<p>Key length in bytes.</p> <p>The value supplied by the LENGTH keyword or the byte length of the key value if the KEY option was selected.</p> <p>This value is for ease of processing the key values. The exit may not modify this value.</p>

Table 31. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
316	16	<p>Key key-value 1.</p> <p>The value of the key supplied on the KGUP control statement. The 16 bytes are hexadecimal characters representing the 8-byte hexadecimal key value. The field contains a value only if the KEY option was specified and a key value was supplied on the control statement. You can determine whether the KEY option was used by examining bit 2 at offset +13 in KGXP.</p> <p>If TRANSKEY was specified on the control statement, KGUP decrypts key-value1 under the transport key specified with the TRANSKEY keyword. If CLEAR was specified on the control statement, KGUP does not decrypt key-value1.</p> <p>The exit can modify the field at the record preprocessing invocation. This field is not used for the KGUP pre- or postprocessing invocations or the record postprocessing invocation. The field does not contain a value when generating keys.</p> <p>The exit is permitted to put values in this field only if a key was supplied on the control statement. The exit-supplied value must be edited for hexadecimal values and it then replaces the values entered on the input control statement.</p>
332	16	<p>Key key-value 2.</p> <p>The value of the second key supplied on the KGUP control statement. The 16 bytes are hexadecimal characters representing the 8-byte hexadecimal key value. The field contains a value only if the KEY option was specified and a key value was supplied on the control statement. You can determine whether the KEY option was used by examining bit 2 at offset +13 in KGXP.</p> <p>If TRANSKEY was specified on the control statement, KGUP decrypts the key-value 2 under the transport key specified with the TRANSKEY keyword. If SINGLE was specified on the control statement, the key-value 2 will be equal to the key-value. If CLEAR was specified on the control statement, KGUP does not decrypt the key-value 2.</p> <p>The exit can modify the field at the record preprocessing invocation. This field is not used at the KGUP pre- or postprocessing invocation or the record postprocessing invocation.</p> <p>The field does not contain a value when generating keys.</p> <p>The exit can put values in this field only if a key was supplied on the control statement. The exit-supplied value must be edited for hexadecimal values; it then replaces the values entered on the input control statement.</p>

Table 31. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
348	16	<p>Key key-value 3.</p> <p>The value of the third key supplied on the KGUP control statement. The 16 bytes are hexadecimal characters representing the 8-byte hexadecimal key value. The field contains a value only if the KEY option was specified and a key value was supplied on the control statement. You can determine whether the KEY option was used by examining bit 2 at offset +13 in KGXP.</p> <p>If TRANSKEY was specified on the control statement, KGUP decrypts the key-value 3 under the transport key specified with the TRANSKEY keyword. If CLEAR was specified on the control statement, KGUP does not decrypt the key-value 3.</p> <p>The exit can modify the field at the record preprocessing invocation. This field is not used at the KGUP pre- or postprocessing invocation or the record postprocessing invocation.</p> <p>The field does not contain a value when generating keys.</p> <p>The exit can put values in this field only if a key was supplied on the control statement. The exit-supplied value must be edited for hexadecimal values; it then replaces the values entered on the input control statement.</p>
364	16	<p>Key key-value 4.</p> <p>The value of the fourth key supplied on the KGUP control statement. The 16 bytes are hexadecimal characters representing the 8-byte hexadecimal key value. The field contains a value only if the KEY option was specified and a key value was supplied on the control statement. You can determine whether the KEY option was used by examining bit 2 at offset +13 in KGXP.</p> <p>The exit can modify the field at the record preprocessing invocation. This field is not used at the KGUP pre- or post-processing invocation or the record post-processing invocation. The field does not contain a value when generating keys.</p> <p>The exit can put values in this field only if a key was supplied on the control statement. The exit-supplied value must be edited for hexadecimal values; it then replaces the values entered on the input control statement.</p>

Table 31. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
380	4	<p>CSFKEYS record for transkey, key-label1.</p> <p>The address of the CSFKEYS data set record that is output for transkey key-label1 on the KGUP control statement. This field only contains a value when CLEAR keys are generated.</p> <p>The exit can modify the field at the record postprocessing invocation. KGUP sets the address to zero for the KGUP pre- or postprocessing and record preprocessing invocations.</p> <p>KGUP does not check the field upon return from the exit. Normal CSFKEYS processing applies. KGUP uses key values on control statement creation.</p> <p>For the format of the CSFKEYS record, refer to <i>z/OS Cryptographic Services ICSF Administrator's Guide</i>.</p>
384	4	<p>CSFKEYS record for transkey, key-label2.</p> <p>The address of the CSFKEYS data set record that is output for transkey key-label2 on the KGUP control statement. This field only contains a value when TRANSKEY key-label2 is specified for generated keys.</p> <p>The exit can modify the field at the record postprocessing invocation. KGUP sets the address to zero for the KGUP pre- or postprocessing and record preprocessing invocations.</p> <p>KGUP does not check the field upon return from the exit. Normal CSFKEYS processing applies. KGUP uses key values on control statement creation.</p>
388	4	<p>CSFCKDS header record.</p> <p>The address of the CSFCKDS data set header record.</p> <p>The exit can check the field at the KGUP pre- or postprocessing invocations. However, the exit can modify the field only at the KGUP postprocessing invocation. KGUP sets the value of the field to zero for the record pre- or postprocessing invocations.</p> <p>The exit can modify the installation data field of the CKDS header record (see “Debugging aids” on page 125 for a description of the CKDS header record. Offset +196 of the CKDS header record is the installation data field). The installation data field supplied by the exit is placed in the CKDS header record after the KGUP postprocessing invocation returns control to KGUP.</p>
392	4	<p>CSFCKDS record.</p> <p>The address of the CSFCKDS data set record processed by the KGUP control statement. KGUP sets the address to zero if the TRANSKEY keyword has two labels of transport keys.</p> <p>The exit can check the field only at the record postprocessing invocation. KGUP sets the address to zero for the record preprocessing and KGUP pre- or postprocessing invocations.</p> <p>The exit can modify the record area if the TRANSKEY keyword does not have two labels.</p>

Table 31. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
396	4	<p>RENAME CSFCKDS record.</p> <p>The address of the CSFCKDS data set record processed when the RENAME verb is used in a control statement. You can determine whether the RENAME verb was used by examining bit 3 at offset +14 in KGXP.</p> <p>The exit can modify the field at the record postprocessing invocation. KGUP sets the address to zero for the record preprocessing and KGUP pre- or postprocessing invocations.</p> <p>The exit can modify the record area. KGUP does not check this field upon return from the invocation. Normal CSFCKDS processing applies.</p>
400	4	<p>Installation data.</p> <p>The address of the data specified on the INSTDATA keyword of the KGUP control statement. The address of the area is zero if a SET control statement has not been processed. “The SET statement” on page 182 describes how to use the field in greater detail.</p>
404	4	<p>Installation exit area.</p> <p>The address of an area set by the installation that is preserved across all invocations of the exit. The first byte of the area contains the length of the area (including the length byte). After KGUP completes, the first 64 bytes of the area are written to the SMF data set. The exit has exclusive control of modifying this area. The area is only used as input to SMF processing upon completion of KGUP.</p>

The SET statement

Use the SET control statements to specify data to send to a KGUP installation exit. For a more detailed description of the SET statement, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

The installation data field in KGXP (offset +396) contains the address of the data SET statement specifies. Data that is specified on a SET statement can be especially useful if you alter key entries. You may want to keep track of the entries you change by putting the original data and the changed data in the installation data area.

Return codes

You can pass a return code back to KGUP in the KGXP control block (offset +8). The exit can use the return code to cause KGUP to reject control statements or to end KGUP. Return code values, in decimal, for record pre- or postprocessing exit calls are:

Return Code Description

- 0**
Normal, continue processing.
- 4**
Reject control statement, but do not end KGUP.
- 8**
End KGUP.

All other return codes are not valid and cause KGUP to end.

Return code values, in decimal, for the KGUP pre- or postprocessing invocations are:

Return Code	Description
--------------------	--------------------

0	Normal, continue processing.
>0	End KGUP.

Chapter 6. Installation-defined Callable Services

This topic contains Programming Interface information.

ICSF provides callable services that perform cryptographic functions. For example, the ICSF encipher callable service enciphers data. You call and pass parameters to a callable service from an application program. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for a description of the ICSF callable services.

Besides the callable services that ICSF provides, you can write your own callable services; these are known as *installation-defined callable services*.



Attention: Only an experienced system programmer should attempt to write an installation-defined callable service. The writing and installation of such a service require a thorough knowledge of system programming in an z/OS environment. If, without having this knowledge, you attempt to write or to install installation-defined callable services, you run the risk of seriously degrading the performance of your system and causing complete system failure.

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. Use the SERVICE installation option keyword to specify a number to identify the service and the load module that contains the service.

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

During ICSF startup, ICSF loads the load module that contains the service into the ICSF address space with the ICSF callable services. ICSF binds the service with the service number that you specified in the installation options data set.

This topic describes how to perform these tasks:

- Write a callable service.
- Define a callable service.
- Write a service stub.

Writing a callable service

An installation-defined callable service receives parameters from the application program when the program calls the service stub that is associated with the service. An installation-defined service can also access information in the secondary parameter block (SPB). The address of the SPB is passed in register 0. See *“Secondary parameter block” on page 154* for a description of the SPB.

The service receives control with these characteristics.

- Supervisor state
- Key 0
- APF authorized
- TCB or SRB mode
- Cross memory mode
- AR mode
- AMODE(31) or AMODE(64)
- RMODE(ANY)

The service can change the characteristics during their processing. However, the service must return to its caller with the same characteristics as on entry.

You must write the services in assembler, because you are in Access Register and cross memory mode, and the addresses of some of the parameters you may access are ALET-qualified. In particular, parameters passed into a callable service are in the user's address space, which you can access with an ALET of 1. See *z/OS MVS Programming: Extended Addressability Guide* for information about cross memory and AR mode.

Contents of registers

The contents of the registers on entry to the callable service are:

Register 0

Address of the secondary parameter block (SPB)

Register 1

Address of the parameter list

Register 2–13

Unpredictable

Register 14

Return address

Register 15

Service entry point address

The contents of the registers on exit from the callable service are:

Register 0

Reason code

Register 1–14

Same as on entry

Register 15

Return code

[Figure 5 on page 187](#) shows an example of entry and exit code for a generic service.

```

MYSERV    CSECT
MYSERV    AMODE    31
MYSERV    RMODE    ANY
          USING    *,15
          B        PROLOG                Branch around header text
          DC        C'some text'
          DC        C'compile date/time'
PROLOG     EQU      *
          DROP     15
          BSM      R14,0
          BAKR     14,0                Save callers info on stack
          LAE      12,0                Clear access register 12
          LR       12,15                Load reg 15 into 12
PROGSTRT   EQU      *
          USING    MYSERV,12            Set up base register
          *                                     addressability
          .
          .
          .
          Get dynamic area for program
          .. STORAGE OBTAIN or CELLPOL or own scheme ...
          .
          .
          Free dynamic area for program
          .
          .
          .
RETURN     L        0,REASON_CODE        Put reason code in reg 0
          L        15,RETURN_CODE        Put return code in reg 15
          PR

```

Figure 5. Example of a service entry and exit

The example uses the instructions BAKR and PR to replace standard linkage. With these instructions, you no longer need to pass the save area in a register.

If the callable service ends abnormally, ICSF takes a system dump. The ICSF service functional recovery routine (FRR) PROTECTS an installation-defined service. You can, however, write your own recovery routine.

Security access control checking

For the ICSF-defined services, ICSF performs security access control checking to determine if the caller is authorized to access the service and the results of the authorization check can be logged in SMF. This checking is not performed by ICSF for installation-defined services or UDXs. Any security access control checking must be performed by the installation-defined service or UDX itself.

Checking the parameters

For the ICSF-defined services, ICSF checks the integrity of user-passed parameters. An error in a parameter that causes a system abend does not cause a system dump. For an installation-defined callable service, you must perform your own integrity checking of parameters. An error in a user parameter that results in a system abend causes a system dump. You can suppress the system dump by setting a bit on in the SPB. To suppress the dump, set the bit on before you check the integrity of the parameters. This bit (the SPBTERM bit) is the third bit of the flag byte at offset 16 in the SPB.

Link-editing the callable service

After you write the callable service, you need to link-edit it into a load module, and install the load module into an APF authorized library. ICSF uses this normal search order to locate the service:

- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Defining a callable service

Use the SERVICE keyword in the installation options data set to specify information about the callable service. ICSF uses this information at ICSF startup to enable the service. See [“Steps to create the installation options data set”](#) on page 21 for more information about ICSF installation options.

The SERVICE keyword has this syntax:

SERVICE(service-number,load-module-name,FAIL(fail-option))

The service-number is a number that identifies the service to ICSF. The valid service numbers are 1 through 32767, inclusive. The load-module-name is the name of the module that contains the service your installation wrote. During ICSF startup, ICSF loads the module and binds it to the service number you specified.

Using the fail-option, you specify the action ICSF takes if the loading of the service ends abnormally. ICSF loads all installation-defined services at ICSF startup.

Specify one of these values for the fail-option:

YES

ICSF abends if your service cannot be loaded.

NO

ICSF continues to start if your service cannot be loaded.

If the callable service ends abnormally while it is processing, ICSF does not end.

This SERVICE installation option statement identifies a specific installation-defined service to ICSF:

```
SERVICE(50,KSUST,FAIL(NO))
```

When ICSF starts, it binds the service number 50 to the load module KSUST, which contains the callable service you wrote. Because the fail option is NO, if your service cannot be loaded, ICSF continues to start anyway.

Writing a service stub

Besides writing the callable service itself, you must write a service stub, which is the connection between the application program and the installation-defined service. In an application program, you call the service stub, which accesses the installation-defined service. The service stub can be any name you choose to call it.

The service stub must:

- Check that ICSF is active.
- Place the service number for the installation-defined callable service into register 0.
- Call the IBM-supplied processing routine, CSFAPRPC.

CSFAPRPC is used to access the callable services on ICSF. In the service stub, you must call CSFAPRPC. ICSF stores the address of the CSFAPRPC entry point in the CCVTPRPC field of the ICSF cryptographic communication vector table (CCVT). If running in a CICS address space, then, after you call CSFVCCPP, the system calls the callable service that corresponds to the service number in register 0. [“The Cryptographic Communication Vector Table \(CCVT\)”](#) on page 329 describes the format of the CCVT.

The contents of the registers on entry to the service stub are:

Register 0

Unpredictable

Register 1

Address of the parameter list

Register 2–13

Unpredictable

Register 14

Return address

Register 15

Service stub entry point address

The contents of the registers on exit from the service stub are:

Register 0

Reason code

Register 1–14

Same as on entry

Register 15

Return code

To run an installation-defined callable service, an application program calls the service stub. You must link-edit the service stub with the application program that calls the service stub. Any application program that calls a service stub must be link-edited with the service stub.

To call an installation-defined service from an application program, use this statement:

```
CALL <service-stub-name> <service-parameters>
```

The service-stub-name is the name of the service stub for the installation-defined callable service. The service-parameters are the parameters you want to pass to the installation-defined service. You supply the parameters according to the syntax of the programming language that you use to write the application program.

Example of a service stub

Figure 6 on page 190 through [Figure 10 on page 194](#) show an example of a service stub for an installation-defined callable service.

```

**** START OF SPECIFICATIONS ****
*
*   MODULE NAME = CSFGEN
*   DESCRIPTIVE NAME = SERVICE STUB
*
*   FUNCTION =
*   THIS IS A SAMPLE SERVICE STUB. IT IS MEANT TO BE LINKEDITED
*   WITH THE APPLICATION AND ENTERED VIA A CALL CSFGEN. THIS STUB
*   CAUSES THE EXECUTION OF THE SERVICE WITH SERVICE NUMBER = 50
*   (DECIMAL).
*   MODULE TYPE = ASSEMBLER
*   PROCESSOR = ASSEMBLER
*   MODULE SIZE = ONE BASE REGISTER
*
**** END OF SPECIFICATIONS ****
CSFGEN  START 0
GENSNUM EQU 50
CSFGEN  CSECT
CSFGEN  AMODE 31
CSFGEN  RMODE ANY
MAINENT DS 0H
        USING *,R15
        LAE  R15,0(R15,0)
        L    R15,=A(CICSTEST)
        BAKR 0,R15          PR from CICSTEST will restore GPRs
        LTR  R15,R15
        BC   2,NOCICS
*
YESCICS DS 0H
        SAC  0
        STM  R14,R12,12(R13)
        LR   R12,R15
        DROP R15
        USING MAINENT,R12
        LR   R3,R0
        B    NORMAL
*
        NOCICS DS 0H
        USING MAINENT,R12
        BSM  R14,0
        BAKR R14,0
        LAE  R12,0
        LR   R12,R15
        SLR  R13,R13
*****
* At this point, R0 must contain the service number.
*
*       If we are to call the TRUE, R13 is non-zero
*
*       R1 points to the caller's parameter list.
*****
NORMAL  DS 0H
        LA  R0,GENSNUM          R0 gets service number
        SLR R10_ZERO,R10_ZERO
        LR  RC,R10_ZERO
        L   R2,CVTPTR
        USING CVT,R2
        L   R2,CVTABEND

```

Figure 6. Example of a service stub (1 of 5)

```

        CLR    R2,R10_ZERO
        BC     8,NOICSF
        USING  SCVTSECT,R2
        L      R2,SCVTCCVT
        CLR    R2,R10_ZERO
        BC     8,NOICSF
        USING  CCVT,R2
        TM     CCVTSFG1,B'00110000' IS ICSF ACTIVE
        BC     1,YESICSF
NOICSF   LA     RC,12             Set return code to 12 decimal
        L      R7,RETURN_CODE_PTR(,R1)
        ST     RC,RETURN_CODE(,R7)
        SLR    R0,R0
        L      R7,REASON_CODE_PTR(,R1)
        ST     R0,REASON_CODE(,R7)
        B      FINISHED
YESICSF DS    0H
*****
* Note that, if we're in CICS, the prolog code pointed R3 at the AFCB
* and R13 at the caller's savearea--they're still pointing. Also, R0
* contains the service number, with the high order bit ON if the TRUE
* has been tried and found wanting. In this last case, CSFVCCPP will
* check the high order bit and not attempt to call the TRUE.
* If R13 is zero, we're using the linkage stack. That means we can
* call CSFAPRPC.
* If R13 is not zero, we're using non-stack linkage. That means the
* caller's savearea will be used. CSFVCCPP uses this kind of linkage.
* But note that CSFVCCPP will not return here. Instead, it will return
* directly to the caller--that is, to the owner of the only save
* area around.
*****
        CLR    R13,R10_ZERO
        BC     8,EXECPRPC
        L      R15,CCVTPRPD
        BALR   R14,R15
LR       RC,R15
        B      FINISHED
EXECPRPC L     R15,CCVTPRPC
        BALR   R14,R15
        LR     RC,R15
FINISHED DS    0H
*
*****
* This routine uses the linkage stack to save the caller's regs
* if this is not a CICS environment. In CICS, it uses the save
* area pointed to by register 13. So the epilog code takes one
* of two forms. If this is CICS (i.e. if R13 is non-zero),
* return is via LM and BR 14. If this is not CICS, return is
* via PR.
*
* On return, the PR of ESA linkage does not restore registers
* 0, 1, 14 and 15. In the LM of normal BR 14 linkage, however,
* everything but 13 gets restored. Since this routine has no
* autodata, there's no way to pass back return and reason codes
* unless we leave 0 and 15 intact. The solution is to deviate
* slightly from normal BR 14 linkage and restore only registers
* 1 through 12 and 14.
*****
        LTR    R13,R13
        BC     8,ENDNOCICS

```

Figure 7. Example of a service stub (2 of 5)

```

ENDCICS      LR      R15,RC
              L       R14,SAVE14(,R13)
              LM      R1,R12,24(R13)
              BR      R14

*
EDNOCICS     DS      0H
              LR      R15,RC
              LA      R7,12
              CR      R15,R7
              BNE     ENDSVC
              LA      R7,16
              CR      R0,R7
              BNE     ENDSVC
              L       R7,RETURN_CODE_PTR(,R1)
              ST      R15,RETURN_CODE(,R7)
              L       R7,REASON_CODE_PTR(,R1)
              ST      R0,REASON_CODE(,R7)
ENDSVC       LR      R15,RC
              PR

*****
*****
**  CICSTEST:  Decides whether this is a CICS environment
*****
*****
CICSTEST     DS      0H
              LAE     R12,0          Clear AR 12
              LR      R12,R15       Addressability via R12
              USING   CICSTEST,R12
              L       R15,=A(CSFGEN) R15 gets caller's base reg
              L       R2,CVTPTR     GET CVT POINTER
              USING   CVT,R2
              L       R2,CVTABEND    AND SECONDARY CVT POINTER
              USING   SCVTSECT,R2
              L       R2,SCVTCCVT    POINT TO CSF CCVT
              LTR     R2,R2          IS CRYPTO INSTALLED?
              BZ      RETRN          IF NOT, GO HOME
              USING   CCVT,R2
              TM      CCVTSFG1,B'00110000' IS ICSF ACTIVE
              BNO     RETRN          IF NOT , GO HOME

* Check for wait list routine
*
              TM      CCVTCICS,B'10000000' Q. CCVTPRPA ON?
              BZ      RETRN          no---No CICS capability
              TM      CCVTCICS,B'01000000' Q. CCVTCKWL ON?
              BZ      CKWLHERE      no---use imbedded routine
                                   yes--use installed routine
*
              LA      R0,GENSNUM     R0 gets service number
              LR      R3,R1          R3 saves R1
              LR      R4,R14         R4 saves R14
              LR      R5,R15         R5 saves R15
              L       R15,CCVTCKWL   R15 gets routine address
              BALR    R14,R15         Go check for CICS
              LR      R0,R15         Save return code in R0
              LR      R15,R5         Restore R15
              LR      R14,R4         Restore R14
              LR      R1,R3          Restore R1
              LTR     R0,R0          Q. CICS?
              BZ      RETRN          no---return
                                   yes--pass info along
*
              O       R15,M_CICS     Enable high bit of R15 to CICS
              B       RETRN          Return

```

Figure 8. Example of a service stub (3 of 5)


```

* Cannot use installed routine. Use imbedded routine
*
CKWLHERE DS      0H                      Imbedded check for TRUE routine
        SLR      R0,R0                      Init R0 to 0
        CPYA     R8,R12                     Zero AR 8
        SLR      R8,R8                      Init R8 to 0
        USING    PSA,R8
        L        R8,PSATOLD                 R8->TCB
        USING    TCB,R8
        LTR      R8,R8                      Q. Is there a TCB?
        BC       8,RETRN                     no---return
*                                           yes--check state and key
        CPYA     R11,R12                     Zero AR 11
        LA       R11,1                      Get PSW state and key in R6
        ESTA     R6,R11
        LR       R7,R6                      Copy of state & key in R7
        N        R7,M_KEY                    Q. problem key?
        BZ       RETRN                       no---return
*                                           yes--check state
        N        R6,M_STATE                  Q. problem state?
        BZ       RETRN                       no---return
*                                           yes--get the CICS eye-catcher
        LA       R6,2                      Set ARs 6 and 8 to home
        SAR      R6,R6
        SAR      R8,R6
        L        R8,TCBEXT2                 R8->TCB extension
        USING    TCBXTNT2,R8
        ICM      R4,B'1111',TCBCAUF        R4 gets AFCX address
*                                           Q. Address there?
        BZ       RETRN                       no---return
*                                           yes--check eye-catch
        CLC      0(4,R4),CICS_EYE           Q. CICS?
        BNE      RETRN                       no---return
*                                           yes--pass info along
        LR       R0,R4                      R0 gets the AFCX pointer
        O        R15,M_CICS                 Enable high order bit of R15
RETRN    DS      0H
        DROP     R12                       Free R12
*                                           Return from CICSTEST subroutine
        PR
*
        LTORG
        DS      0D
*
GENSDATA DS      0F
R10_ZERO EQU     10
R0        EQU     0
R1        EQU     1
R2        EQU     2
R3        EQU     3
R4        EQU     4
R5        EQU     5
R6        EQU     6
R7        EQU     7
R8        EQU     8
R9        EQU     9
R10       EQU     10
R11       EQU     11
R12       EQU     12
R13       EQU     13
R14       EQU     14
R15       EQU     15
*

```

Figure 9. Example of a service stub (4 of 5)

```

INPUT_PARMs      EQU 0,8,C'C'
RETURN_CODE_PTR  EQU INPUT_PARMs,4,C'A'
REASON_CODE_PTR  EQU INPUT_PARMs+4,4,C'A'
RETURN_CODE      EQU 0,4,C'F'
REASON_CODE      EQU 0,4,C'F'
*
SAVAREA EQU 0,72,C'C'
SAVE14  EQU SAVAREA+12,4,C'A'
SAVE01  EQU SAVAREA+24,4,C'A'
SCVTSPTR EQU CVTABEND,4,C'F'
TCBPTR  EQU PSATOLD,4,C'F'
DS      0D
*
M_KEY    DS 0F                      Align
M_STATE  DC X'00800000'             Problem key mask
M_STATE  DC X'00010000'             Problem state mask
M_NOCICS DC X'7FFFFFFF'             Not-CICS mask
M_CICS   DC X'80000000'             Yes-CICS mask
DS      0D
CICS_EYE DC CL4'AFCX'               CICS eye catcher
*
      IHAPSA
      TITLE 'DSECT CVT'
      CVT DSECT=YES
      TITLE 'DSECT SCVT'
      IHASCVT DSECT=YES
      TITLE 'DSECT TCB'
      IKJTCB
      TITLE 'DSECT CCVT'
      CSFCCVT
*
      LA R7,12
      CR R15,R7
      BNE ENDGSVC
      LA R7,16
      CR R0,R7
      BNE ENDGSVC
      L  R7,RETURN_CODE_PTR(,R1)
      ST R15,RETURN_CODE(,R7)
      L  R7,REASON_CODE_PTR(,R1)
      ST R0,REASON_CODE(,R7)
      ENDGSVC DS 0H

      END

```

Figure 10. Example of a service stub (5 of 5)

In Figure 6 on page 190, the service stub, CSFGEN, checks that ICSF is active, places the service number 50 into register 0, and calls CSFAPRPC.

The service number 50 (in the case of this example) must be bound to the installation-defined service by using the SERVICE keyword in the installation options data set. The service number is bound to the service when ICSF interprets the SERVICE installation option statement and loads the service at ICSF startup. To run the callable service that is associated with service number 50, call the service stub CSFGEN from an application program.

For flexibility, to create a service stub for a different installation-defined callable service, you can copy an existing service stub and just change the service number that you load into register 0.

Chapter 7. Converting a CKDS from fixed length to variable length record format

ICSF provides a CKDS conversion program, CSFCNV2, that converts a fixed length record format CKDS to a variable length record format. There will be no changes to the key token in the CKDS record. Only the format of the record will be changed.

Note: There are three formats of the CKDS:

- The original fixed-length record format.
- The variable-length record format (introduced in HCR7780).
- The KDSR variable-length record format (introduced in ICSF FMID HCR77A1).

The CSFCNV2 utility converts a fixed-length format CKDS to a variable-length format. To convert a fixed-length or variable-length format CKDS to the KDSR format, see [“Migrating to the common record format \(KDSR\) key data set”](#) on page 69.

You can also use the CSFCNV2 utility to rewrap encrypted DES values in the CKDS. For more information on this capability of the CSFCNV2 utility, refer to *z/OS Cryptographic Services ICSF Administrator's Guide*.

There is no conversion from variable length to fixed length records.

You run the conversion utility program by submitting a batch job. On the EXEC statement, specify PGM=CSFCNV2.

This example is a JCL that runs the conversion program:

```
//CKDSCNV2 EXEC PGM=CSFCNV2,PARM='FORMAT,OLD.CKDS,NEW.CKDS'
```

Where:

OLD.CKDS

The fixed length record format CKDS to be converted. This is the source CKDS for the conversion.

NEW.CKDS

An empty disk copy of a variable length record format CKDS. This is the CKDS into which the conversion utility writes the converted records. The data set must be defined and empty before you run the conversion program.

Refer to the SYS1.SAMPLIB CSFCKD2 member sample described in [“Steps to create the CKDS”](#) on page 12 for example JCL that defines a VSAM CKDS for variable length records.

The CSFV0560 message in the joblog will indicate the results of processing.

Return Code

Meaning

0

Process successful.

4

Minor error occurred.

8

RACF authorization check failed.

12

Process unsuccessful.

60 or 92

CKDS processing has failed. A return code 60 indicates the error was detected in the new KDS. A return code 92 indicates the error was detected with the old KDS.

When the program is invoked from another program, the invoking program receives the reason code in General Register 0 along with the return code in General Register 15. The following list describes the meaning of the reason codes. If a particular reason code is not listed, refer to the listing of ICSF and TSS return and reason codes in the *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Return code 0 has this reason code:

Reason Code	Meaning
--------------------	----------------

36132	
--------------	--

	CKDS reencipher/Change MK processed only tokens encrypted under the DES master key.
--	---

Return code 4 has these reason codes:

Reason Code	Meaning
--------------------	----------------

0	
----------	--

	Parameters are incorrect.
--	---------------------------

4004	
-------------	--

	Rewrapping is not allowed for one or more keys.
--	---

36112	
--------------	--

	CKDS conversion completed successfully but some tokens could not be rewrapped because the control vector prohibited rewapping from the enhanced wrapping method.
--	--

36164	
--------------	--

	Input CKDS is already in the variable-length record format. No conversion is necessary.
--	---

Return code 8 has this reason code:

Reason Code	Meaning
--------------------	----------------

16000	
--------------	--

	Invoker has insufficient RACF access authority to perform function.
--	---

Return code 12 has these reason codes:

Reason Code	Meaning
--------------------	----------------

0	
----------	--

	ICSF has not been started
--	---------------------------

11060	
--------------	--

	The required cryptographic coprocessor was not active or the master key has not been set
--	--

36000	
--------------	--

	Unable to change master key. Check hardware status.
--	---

36008	
--------------	--

	Crypto master key register or registers in improper state.
--	--

36020	
--------------	--

	Input CKDS is empty or not initialized (authentication pattern in the control record is invalid).
--	---

36036	
--------------	--

	The new master key register for Coprocessor 1 (C1) is not full, but C0 is ready and the current master key is valid.
--	--

36040	
--------------	--

	The new master key register for C0 is not full, but C1 is ready and the current master key is valid.
--	--

36044	
--------------	--

	The master key authentication pattern for the CKDS does not match the authentication pattern of the coprocessors, which are not equal.
--	--

36048

The master key authentication pattern for the CKDS does not match the authentication pattern of either of the coprocessors, which are not equal.

36052

A valid new master key is present in C0, but its authentication pattern does not match that of C1 or the CKDS, which are equal.

36056

A valid new master key is present in C1, but its authentication pattern does not match that of C0 or the CKDS, which are equal.

36060

The new master key register or registers are not full.

36064

Both new master key registers are full but not equal.

36068

The input KDS is not enciphered under the current master key.

36076

The new master key register for C0 is not full, but the CPUs are online.

36080

The new master key register for C1 is not full, but the CPUs are online.

36084

The master key register cannot be changed since ICSF is running in compatibility mode.

36104

Option not available. There were no Cryptographic Coprocessors available to perform the service that was attempted.

36108

PKA callable services are enabled, and the PKDS is the active PKDS as specified in the options data set.

36120

The CKDS is unusable. The CKDS does not support record level authentication.

36124

The CKDS is unusable. The CKDS only supports encrypted AES keys and encrypted DES support is required.

36128

The CKDS is unusable. The CKDS does not support encrypted DES keys which is required.

36160

The attempt to reencipher the CKDS failed because there is an enhanced token in the CKDS.

36168

A CKDS has an invalid LRECL value for the requested function. For wrapping, the input and output CKDS LRECLs must be the same.

36172

The level of hardware required to perform the operation is not available.

Return code 60 or 92 has these reason codes:

Reason Code**Meaning****3078**

The CKDS was created with an unsupported LRECL.

5896

The CKDS does not exist.

6008

A service routine has failed.

The service routines that may be called are:

CSFMGN

MAC generation

CSFMVR

MAC verification

CSFMKVR

Master key verification

6012

The single-record, read-write installation exit (CSFSRRW) returned a return code greater than 4.

6016

An I/O error occurred reading or writing the CKDS.

6020

The CSFSRRW installation exit abended and the installation options EXIT keyword specifies that the invoking service should end.

6024

The CSFSRRW installation exit abended and the installation options EXIT keyword specifies that ICSF should end.

6028

The CKDS access routine could not establish the ESTAE environment.

6040

The CSFSRRW installation exit could not be loaded and is required.

6044

Information necessary to set up CSFSRRW installation exit processing could not be obtained.

6048

The system keys cannot be found while attempting to write a complete CKDS data set.

6052

For a write CKDS record request, the current master key verification pattern (MKVP) does not match the CKDS header record MKVP.

6056

The output CKDS is not empty.

Note: It is possible that you will receive MVS reason codes rather than ICSF reason codes, for example, if the reason code indicates a dynamic allocation failure. For an explanation of Dynamic Allocation reason codes, see [*z/OS MVS Programming: Authorized Assembler Services Guide*](#).

Chapter 8. Migration from PCF to z/OS ICSF

If your installation uses the cryptographic product, Programmed Cryptographic Facility (PCF), ICSF helps you migrate PCF applications to ICSF. You can run PCF applications on ICSF to gain the enhanced performance and availability of ICSF and to test ICSF. Eventually, you should convert these applications to use ICSF services, rather than the PCF macros.

During migration, you can run PCF applications on ICSF because ICSF continues to support the PCF macros (GENKEY, RETKEY, EMK, and CIPHER). If GENKEY or RETKEY macro exits exist, you should reevaluate their applicability to ICSF. If an exit performs a necessary function, you need to rewrite the exit for ICSF. Exits exist for the compatibility services on ICSF.

If a PCF application uses a key in the PCF cryptographic key data set, you must convert the key to an ICSF cryptographic key data set before you run the PCF application on ICSF. ICSF provides a program to make this conversion.

Running PCF and z/OS ICSF on the same system

You can run PCF and ICSF simultaneously on the same z/OS system or separately in three different modes. You can run ICSF in compatibility, coexistence, or noncompatibility mode.

In compatibility mode, you can run either PCF or ICSF, but you cannot run them simultaneously on the same z/OS system. You can continue to run PCF applications on PCF or you can run PCF applications on ICSF. ICSF supports the PCF macros that the PCF applications call. However, you cannot run the PCF key generator utility program (KGUP) on ICSF. You do not have to reassemble PCF applications to run the applications on ICSF.

In coexistence mode, you can run PCF and ICSF simultaneously on the same z/OS system. You can continue to run a PCF application on PCF or you can reassemble the PCF application to run on ICSF. In this mode, ICSF supports the PCF macros when a reassembled PCF application calls these macros.

In noncompatibility mode, you can run PCF and ICSF simultaneously and independently on the same z/OS system. You can run PCF applications on PCF and ICSF applications on ICSF. You cannot run PCF applications on ICSF, because ICSF does not support the PCF macros in this mode.

You can run PCF simultaneously and independently in coexistence and noncompatibility mode. Therefore, in these modes, you can run PCF KGUP on PCF while running ICSF. The PCF KGUP updates keys on a PCF CKDS.

The ICSF installation option COMPAT(YES, COEXIST or NO) allows you to specify which mode you want ICSF to run in. You specify COMPAT(YES) for compatibility mode, COMPAT(COEXIST) for coexistence mode, and COMPAT(NO) for noncompatibility mode. See [“Steps to create the installation options data set” on page 21](#) for information about creating the installation options data set and [“Parameters in the installation options data set” on page 30](#) for details about these options.

Running in compatibility mode

In compatibility mode, you can run a PCF application on ICSF without reassembling the application. A PCF application running on ICSF can still use PCF macros, because ICSF supports these macros. The PCF application gains the enhanced performance, reliability, and availability of ICSF.

You cannot run PCF and ICSF simultaneously on the same z/OS system in compatibility mode. If you start PCF, you must stop PCF before you can start ICSF. If you start ICSF, you must stop ICSF before you can start PCF.

A PCF application may have used keys on the PCF cryptographic key data set (CKDS). When you run the application on ICSF, these keys must be in the ICSF CKDS. The format of a key entry on the PCF CKDS differs from the format of a key entry on the ICSF CKDS. Therefore, you need to run a conversion program

to convert the PCF CKDS entries and place the entries in the ICSF CKDS. See [“Converting a PCF CKDS to ICSF format” on page 202](#) for a description of how to convert a PCF CKDS.

For encryption, ICSF supports the Data Encryption Standard (DES).

PCF macros receive identical error return codes if they run on ICSF or PCF, with one exception. If a key is installed on the ICSF CKDS with the correct label but with the wrong key type, an attempt to use that key by RETKEY or GENKEY results in a return code of 8 from PCF. This indicates that the key was not of the correct type. ICSF issues return code 12, indicating that it could not find the key. Ensure that PCF LOCAL or CROSS 1 keys are installed in the ICSF CKDS as EXPORTER keys. Also, ensure that REMOTE and CROSS 2 keys are installed in the ICSF CKDS as IMPORTER keys.

In compatibility mode, the safest method for changing the master key is to re-IPL the system. To change the master key in compatibility mode, see [“Changing the DES master key in compatibility or coexistence mode” on page 201](#).

Note: To use AMS REPRO encryption, you need to run ICSF in compatibility mode.

Running in coexistence mode

In coexistence mode, you can run ICSF and PCF simultaneously on the same z/OS system and run a PCF application on PCF or on ICSF. A PCF application running on ICSF gains the enhanced performance, reliability, and availability of ICSF.

A PCF application running on ICSF can still use PCF macros, because ICSF supports these macros. ICSF ships changed PCF macros in SAMPLIB that run only on ICSF. Because these changed PCF macros already exist unchanged on PCF, the changed PCF macros shipped with ICSF are named differently.

On ICSF, in SAMPLIB:

- The changed PCF EMK macro is named CSFEMK.
- The changed PCF CIPHER macro is named CSFCIPH.
- The changed PCF RETKEY macro is named CSFRKY.
- The changed PCF GENKEY macro is named CSFGKY.

You can rename these macros to the PCF names when you want to run a PCF application on ICSF.

To run a PCF application on ICSF, you must:

- Rename the changed PCF macro shipped in ICSF SAMPLIB to the appropriate PCF name.
- Place the macro in the appropriate macro library.
- Reassemble the PCF application against the changed PCF macro.

Then the application can run only on ICSF. To run a PCF application on PCF, just run the application without reassembling the application.

During migration, you can start ICSF and start PCF so that both products are running simultaneously. If you want to run a PCF application using the PCF macros on PCF, do not reassemble the application. If you want to run a PCF application using the changed PCF macros on ICSF, reassemble the application against the changed macros. Coexistence mode enables you to run the products simultaneously and choose whether to run a PCF application on PCF or ICSF.

A PCF application can use keys on the PCF CKDS. When you run the application on ICSF, those keys must be in the ICSF CKDS. The format of a key entry on the PCF CKDS differs from the format of a key entry on the ICSF CKDS. Therefore, you need to run a conversion program to convert the PCF CKDS entries and place the entries in the ICSF CKDS. See [“Converting a PCF CKDS to ICSF format” on page 202](#) for a description of how to convert a PCF CKDS.

In coexistence mode, the safest method for changing the master key is to re-IPL the system. See [“Changing the DES master key in compatibility or coexistence mode” on page 201](#) for a description of the process used to change the master key in coexistence mode.

Changing the DES master key in compatibility or coexistence mode

In compatibility and coexistence modes, the safest way to activate the DES master key after changing it is to re-IPL the system. This process is different from the usual process for entering and activating a master key. For information about changing the master key, see [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

A re-IPL ensures that a program does not access a cryptographic service with a key that is encrypted under a different master key. If a program is using an operational key, the program either re-creates the key or imports the key again.

In compatibility or coexistence mode, the ICSF administrator can use the ICSF panels to enter the key value into the new master key register. However, the master key cannot be *activated* using the panels in compatibility or coexistence mode. The value entered remains in the new master key register until you re-IPL the system. (In noncompatibility mode, the ICSF administrator can use the ICSF panels to enter the key value into the new master key register and to activate the master key.)

If the new master key is different than the current master key, the ICSF administrator must reencipher the CKDS under this new master key. To do this, choose the REENCIPHER CKDS option on the master key management panel. This reenciphers a CKDS under the master key in the new master key register. Reencipher all the disk copies of the CKDSs, and leave the ICSF panels without changing the master key.

Then re-IPL the system and restart ICSF. In the installation options data set, the CKDSN installation option must specify a disk copy of the CKDS that is reenciphered under the new master key. When ICSF starts again, it detects that the current master key is not the one that enciphered the CKDS that is specified in the installation options data set. ICSF detects that the CKDS is enciphered under the new master key and makes that master key active.

If your installation requires 24-hour availability and it is not possible to re-IPL the system, an alternative method is to stop all cryptographic applications, especially those using PCF macros. This helps eliminate inadvertent use of operational keys that are encrypted under the old master key. After you restart CSF, applications using an operational key can either re-create or reimport the key.

Running in noncompatibility mode

In noncompatibility mode PCF and ICSF can run simultaneously and independently. You can run both ICSF and PCF at the same time. Just start one and then the other. Both ICSF and PCF run completely separate from each other. Each has its own applications and each uses its own services and CKDS.

You cannot run a PCF application on ICSF, even if you reassemble it. If you run an application on ICSF that calls a PCF macro, the application ends abnormally, because ICSF does not support the PCF macros in noncompatibility mode.

Because each product runs separately, neither product loses any function in exchange for compatibility. When ICSF is in compatibility or coexistence mode, you can no longer change the master key dynamically. In noncompatibility mode, this function is still possible. Therefore, except for when your installation is migrating to ICSF, you probably want to run ICSF in noncompatibility mode.

Note: When you initialize ICSF for the first time, noncompatibility mode must be active.

Specifying compatibility modes during migration

The process and duration to migrate from PCF to ICSF depend on your installation. You can use different modes in different stages of migration. To change modes, change the COMPAT option in the installation options data set and restart ICSF. When you complete migration to ICSF, you can run in noncompatibility mode to use the full function of ICSF.

When you first install an ICSF system, you can continue to run PCF for production and just test ICSF. Because you are running the products separately but simultaneously on the same z/OS system, you can run in noncompatibility or coexistence mode. To run in compatibility mode, you need more than one z/OS system. You can run the test applications on ICSF on one z/OS system while you run your production on PCF on another z/OS system.

When you begin testing ICSF, you can run existing applications in either compatibility mode or coexistence mode to test the PCF macros on ICSF. After you run the test applications, you may want to bring up production using PCF applications on ICSF. When you bring over PCF applications to ICSF, you can run in coexistence mode. In this mode, you can run an application on PCF and then reassemble the application to run the application on ICSF.

While, or after, you bring PCF applications into production on ICSF, you can run test applications that call ICSF services. You can then convert the applications that call PCF macros to applications that call the ICSF services. The ICSF services provide enhanced key separation, performance, and function. After you convert all your PCF applications to ICSF applications, you can activate noncompatibility mode and have the full function of ICSF.

Converting a PCF CKDS to ICSF format

During migration, you may need to convert a PCF CKDS into ICSF CKDS format if:

- PCF applications running on ICSF use keys stored in a PCF CKDS.
- Your installation uses the PCF key generator utility program to create keys and uses ICSF for other cryptographic operations. To use the keys in ICSF applications, you must convert the PCF CKDS.

ICSF provides a PCF conversion program, CSFCONV, that converts a PCF CKDS into an ICSF CKDS. The conversion program runs with certain defaults. The program converts all the entries in a PCF CKDS and converts the PCF key types into certain corresponding ICSF key types. You can use the conversion program override file to instruct the conversion program not to convert certain entries. You can also tell the conversion program to convert a PCF key type into a different ICSF key type than the default.

These topics describe how:

- The conversion program runs with certain defaults
- To use the override file to make it run differently
- To run the conversion program

How the PCF conversion program runs

You can run the PCF conversion program only after you initialize the master key and CKDS for ICSF.

When the conversion program processes a PCF CKDS, the program duplicates the single length key values to create double length keys.

The conversion program merges the PCF CKDS with an input ICSF CKDS. The input ICSF CKDS is an existing disk copy of an ICSF CKDS. The input ICSF CKDS must contain a header record. For information about initializing an ICSF CKDS, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

The PCF conversion program places the input ICSF CKDS entries and the converted PCF entries into an output CKDS. You must create an empty VSAM data set to be the output CKDS before running the conversion program. See [“Steps to create the CKDS” on page 12](#) for information about creating the data set.

The PCF conversion program converts all the entries in a PCF CKDS. When you run the PCF conversion program, the program does these conversions of PCF key types into ICSF key types:

- Converts each PCF local key entry into an ICSF NOCV exporter key-encrypting key entry.
- Converts each PCF remote key entry into an ICSF NOCV importer key-encrypting key entry.
- Converts each PCF cross key entry into two ICSF key entries: an NOCV exporter key-encrypting key and an NOCV importer key-encrypting key.

You use the override file to not convert all the entries in a PCF CKDS or to convert a PCF key into a different key type than the default key type.

When the PCF conversion program converts a PCF entry, the program places any installation data from the installation data field of the PCF entry into the ICSF entry. You can use the override file to place different installation data into the ICSF entry.

Note: ICSF copies any installation data in the input CSF CKDS header record into the output ICSF CKDS header record.

As the conversion program reads the PCF CKDS, the input ICSF CKDS, and the override file, the program places key entries into a virtual image of the output ICSF CKDS. When the virtual image CKDS is complete, the conversion program reenciphers the key values of the PCF entries from under the PCF master key to under the ICSF master key. The conversion program places the reenciphered entries into the actual output CKDS.

As the conversion program creates the virtual image ICSF CKDS, the conversion program takes information from the PCF entry and possibly the override file. For each PCF entry, the conversion program checks if its key label exists in the override file. If the label does exist in the override file, the conversion program takes the action that is specified in the override file. The program either converts or bypasses the entry. If the key label does not exist in the override file, ICSF converts the entry.

The conversion program compares the converted PCF entries by label and type with the ICSF entries that already exist in the input ICSF CKDS. If there is a match, the conversion program replaces the key value from the converted entry of the PCF source into the virtual image CKDS. If there is not a match, the conversion program converts each PCF entry after checking the override file. If the label matches and the type does not, the conversion program checks to see if the type requires a unique label. If a unique label is not required, the conversion program converts the PCF entry after checking the override file. If a unique label is required, the conversion program does not convert the PCF entry and issues an error message. If the record type is DATA, DATAXLAT, MAC, MACVER, or NULL the CKDS record requires a unique label. The CKDS record also requires a unique label if the record has ever been updated by the dynamic CKDS update callable services. The conversion program also places all the input ICSF CKDS entries into the virtual image CKDS.

Calling installation exits during conversion

You can call two installation exits during conversion program processing: the conversion program exit (CSFCONVX) and the single-record, read-write exit (CSFSRRW). The conversion program calls the exit at three different times: before, during, and after conversion program processing. See [Chapter 5, “Installation exits,” on page 131](#) for a description of the conversion program and single-record, read-write exit control blocks.

The conversion program calls the CSFCONVX exit after you submit the conversion program job, but before the program actually begins processing. At this point, you can use the exit to change the output ICSF CKDS header record installation data field.

The conversion program also calls the CSFCONVX exit during processing as the conversion program completes the virtual image ICSF CKDS, but before the conversion program reenciphers the key values. The conversion program calls the exit as it writes each record to the virtual image ICSF CKDS. At this point, you can use the exit to specify that the conversion program not place an entry into the output ICSF CKDS.

The conversion program also calls the CSFCONVX exit after the conversion program completes processing. At this point, you can use the exit to change the output ICSF CKDS header record installation data field.

As the conversion program reads the records from the virtual image ICSF CKDS to the actual output ICSF CKDS, it calls the single-record, read-write exit. The conversion program calls the single-record, read-write exit as it writes each record to the output ICSF CKDS. You can use this exit to specify that the conversion program not place an entry into the output ICSF CKDS.

The conversion program writes every entry from the PCF CKDS and input ICSF CKDS into the output ICSF CKDS unless an override record or installation exit indicates that the conversion program should bypass the entry from the PCF CKDS.

Using the conversion program override file

The conversion program converts all entries in a PCF CKDS into ICSF entries. The conversion program also converts each type of PCF key into a specific ICSF key type. If you want the conversion program to bypass certain key entries or convert a specific key or key type differently than it does by default, use the override file.

By specifying override records, you can have the conversion program:

- Bypass conversion of key entries.
- Include information in key entries.
- Convert key types differently than it does by default.

These actions can relate to entries explicitly identified with a key label or entries that are identified globally.

You specify information in certain fields in an override record and leave other fields blank, depending on the action you want the conversion program to take. You can specify a global record affecting more than one PCF CKDS entry or a record that affects only one PCF CKDS entry.

All the override data set records should be in ascending sequence by key label and old key type. If you use global entries, they must be the initial entries in the override record. [Table 32 on page 204](#) shows the syntax of a record in the override file.

Note: All the fields should contain character values and be left-justified.

If you specify a key label in an override record, the conversion program processes the key entry identified by that key label. If you do not specify a key label in an override record, you are using a global override record. The conversion program processes all the key labels that pertain to the information specified by the override file.

You can use a global override record to affect all the entries in a CKDS and then use override records to explicitly affect entries you did not want to have that global override record affect.

Table 32. Format of Records in the Override File		
Column	Length	Description
1	8	Key Label The key label of the PCF entry you want to convert The field can have these values: <ul style="list-style-type: none">• Blanks• A key label existing in the PCF CKDS that you want to convert
9	1	This field must be blank.
10	8	Old Key Type The key type of the key entry you want to convert in the PCF CKDS. The field can have these values: <ul style="list-style-type: none">• Blanks• LOCAL• REMOTE
18	1	This field must be blank.

Table 32. Format of Records in the Override File (continued)

Column	Length	Description
19	8	<p>New Key Type</p> <p>The key type that you want the converted key entry to be in the ICSF CKDS. The master key variant for the key type enciphers the key in the ICSF CKDS entry that the conversion program creates.</p> <p>The field can have these values:</p> <ul style="list-style-type: none"> • Blanks • OPINENC • EXPORTER • IPINENC • IMPORTER
27	1	This field must be blank.
28	8	<p>Ignored</p> <p>In ICSF/MVS Version 1 Release 1, this field contained the key qualifier. The CKDS for ICSF/MVS Version 1 Release 2 or higher does not support key qualifiers. If your installation has a PCF conversion program override file created with ICSF/MVS Version 1 Release 1, you can still use it with z/OS ICSF. Any key qualifier entries are ignored.</p>
36	1	This field must be blank.
37	1	<p>Bypass Flag</p> <p>Used to indicate that an input CKDS entry is not to be included in the new ICSF CKDS. If you set this field to Y, the conversion program does not convert the entry.</p> <p>The field can have these values:</p> <ul style="list-style-type: none"> • Blank (same as N) • N • Y
38	1	This field must be blank.
39	52	<p>Installation Data</p> <p>Any additional information your installation records about a key. The information appears in the installation data field of the new ICSF CKDS.</p> <p>The field can contain any value.</p>

Bypassing conversion of entries

Using an override record, you can bypass a PCF entry so it is not converted and placed in the ICSF CKDS. You can use a global override record to bypass all the entries in the data set and then use explicit override records to convert certain entries. You can also convert most of a PCF CKDS and just bypass certain entries using explicit override records.

These are some examples of override records for bypassing conversion.

Example 1

This example shows an override record specifying that the conversion program not convert any PCF CKDS entry with a certain key label.

EXTOATM3	Y
----------	---

The conversion program bypasses any PCF CKDS entry with the label EXTOATM3.

Example 2

This example shows an override record specifying that the conversion program not convert any PCF CKDS entry with a certain key label and key type.

CRLABEL4 REMOTE	Y
-----------------	---

The conversion program bypasses any PCF CKDS entry with the label CRLABEL4 and key type REMOTE.

Example 3

This example shows a global override record specifying that the conversion program bypass all the entries in a PCF CKDS.

	Y
--	---

The conversion program does not convert any of the entries in the PCF CKDS.

After you specify this global override record, you can use explicit override records to convert certain entries in the PCF CKDS. For example, you can use an override record like this one to explicitly convert PCF entries with a certain label.

ATM03	N
-------	---

In this example, the conversion program converts any PCF CKDS entry with the label ATM03.

Example 4

This example shows a global override record specifying that the conversion program bypass all the entries with a certain PCF key type in a PCF CKDS.

REMOTE	Y
--------	---

The conversion program does not convert any of the entries with a key type of REMOTE in the PCF CKDS. After you specify this global override record, you can use explicit override records to convert specific entries with a key type of REMOTE in the PCF CKDS.

Including information in a key entry

An ICSF key entry contains an installation data field that an installation can use to further identify a key. The installation data field contains any information that an installation wants to supply about a key.

PCF records contain an installation data field. The conversion program places the information in the field into the installation data field of the converted entry in the output ICSF CKDS. You can use an override record to specify installation data information for the converted entry in the output ICSF CKDS. The installation data information supplied in the override record overrides any information from the PCF installation data field. If you do not use an override record, the conversion program places any installation data from the PCF entry into the leftmost 8 bytes of the ICSF entry.

These are examples of override records for including key information.

Example 1

This example shows an override record providing the conversion program with installation data information to place in the ICSF CKDS for any converted PCF entry with a certain key label.

ATMKEY12	CONVERTED FROM CUSP1.CKDS 10/01/98
----------	------------------------------------

When the conversion program converts an entry that is labeled ATMKEY12, it places CONVERTED FROM CUSP1.CKDS 10/01/98 in the installation data field for the converted entry.

Example 2

This example shows an override record providing the conversion program with installation data information to place in the ICSF CKDS for any converted PCF entry with a certain key label and key type.

LOCAL890 LOCAL	CONVERTED FROM PCF12.CKDS
----------------	---------------------------

When the conversion program converts an entry that is labeled LOCAL890 with a key type of LOCAL, it places CONVERTED FROM PCF12.CKDS in the installation data field for the converted entry.

Example 3

This example shows a global override record that provides the conversion program with installation data information to place in the ICSF CKDS for all converted entries.

CONVERTED FROM PCF10.CKDS

When the conversion program converts the PCF CKDS, it places CONVERTED FROM PCF10.CKDS in the installation data field. The information is placed into every converted key entry. After you specify this global override record, you can use explicit override records to provide different information for specific entries in the PCF CKDS.

Converting key types

By default, the conversion program converts PCF keys into certain ICSF key types. You can use the override file to override the defaults. For example:

- Instead of automatically converting a PCF local key into a NOCV exporter key-encrypting key, you can convert the local key into an output PIN-encrypting key.
- Instead of automatically converting a PCF remote key into a NOCV importer key-encrypting key, you can convert the remote key into an input PIN-encrypting key.
- Instead of automatically converting a PCF cross key into a NOCV exporter key-encrypting key and a NOCV importer key-encrypting key, you can convert the cross key into an output PIN-encrypting key and an input PIN-encrypting key.

You can use a global override record to convert all keys of a certain type into a type other than the conversion program default key type. Then using an explicit override record, you can specify that the conversion program convert a specific record into a the default key type. For example, you can use a global override record to convert all remote keys into input PIN-encrypting keys, and then use an override record to convert specific remote entries into importer key-encrypting keys.

These are some examples of override records for key type conversion.

Example 1

This example shows an override record specifying that the conversion program convert a local key to an output PIN-encrypting key for any PCF CKDS entry with a certain key label. The override record also provides the conversion program with installation data.

CRLABEL1	LOCAL	OPINENC	OPINENC FOR ATM123
----------	-------	---------	--------------------

When the conversion program converts any PCF entry labeled CRLABEL1 with a key type of local, it converts the key from a local key type to an output PIN-encrypting key type. The program also places OPINENC FOR ATM123 in the installation data field.

If you did not specify this override record, the conversion program would automatically convert the entry from a local key type to an exporter key-encrypting key type.

Example 2

This example shows an override record specifying that the conversion program convert a remote key to an input PIN-encrypting key for any PCF CKDS entry with a certain key label. The override record also provides the conversion program with installation data.

CRLABEL2	REMOTE	IPINENC	IPINENC FOR ATM123
----------	--------	---------	--------------------

When the conversion program converts any PCF CKDS entry labeled CRLABEL2 with a key type of remote, it converts the key from a remote key type to an input PIN-encrypting key type. The program also places IPINENC FOR ATM123 in the installation data field.

If you did not specify this override record, the conversion program would automatically convert the entry from a remote key type to an importer key-encrypting key type.

Example 3

This example shows an override record specifying that the conversion program convert a local key to an exporter key-encrypting key for any PCF CKDS entry with a certain key label. The override record also provides the conversion program with installation data.

LOLABEL1	LOCAL	EXPORTER	EXPORTER CONVERTED FROM CUSP12.CKDS
----------	-------	----------	-------------------------------------

The conversion program automatically converts a local key to an exporter key-encrypting key. You can use this override record if you previously submitted an override record that had the conversion program convert all the local key types to output PIN-encrypting keys. You can use this override record to explicitly convert the key entry that is labeled LOLABEL1 from a local key type to an exporter key-encrypting key type.

With the example override record, when the conversion program converts any PCF entry labelled LOLABEL1 with a key type of local, it converts the key from a local key type to an exporter key-encrypting key type. The program also places EXPORTER CONVERTED FROM CUSP12.CKDS in the installation data field.

Example 4

This example shows an override record specifying that the conversion program convert a remote key to an importer key-encrypting key for any PCF CKDS entry with a certain key label. The override record also provides the conversion program with installation data.

RECKDS12	REMOTE	IMPORTER	IMPORTER CONVERTED FROM CUSP12.CKDS
----------	--------	----------	-------------------------------------

The conversion program automatically converts remote keys to importer key-encrypting keys. You can use this override record if you supplied an override record to convert all the remote key types to input key-encrypting keys. Use this override record to explicitly convert key entries labeled RECKDS12 from remote key types to importer key-encrypting key types.

With the example override record, when the conversion program converts any PCF entry labeled RECKDS12 with a key type of remote, it converts the key from a remote key type to an importer key-encrypting key type. The program also places IMPORTER CONVERTED FROM CUSP12.CKDS in the installation data field.

Example 5

This example shows a global override record specifying that the conversion program convert a local key to an output PIN-encrypting key for any PCF CKDS entry with a key type of local. The override record also provides the conversion program with installation data.

LOCAL	OPINENC	OPINENC FROM CUSP.PIN12.CKDS
-------	---------	------------------------------

When the conversion program converts any PCF entry with a key type of local, the program converts the key from a local key type to an output PIN-encrypting key type. The program also places OPINENC FROM CUSP.PIN12.CKDS in the installation data field. After you specify this global override record, you can use explicit override records to place different installation data in the ICSF CKDS entries.

Example 6

This example shows a global override record specifying that the conversion program convert a remote key to an input PIN-encrypting key for any PCF CKDS entry with a key type of remote. The override record also provides the conversion program with installation data.

REMOTE	IPINENC	IPINENC FROM CUSP.PIN12.CKDS
--------	---------	------------------------------

When the conversion program converts any CUSP/PCF entry with a key type of remote, it converts the key from a remote key type to an input PIN-encrypting key type. The program also places IPINENC FROM CUSP.PIN12.CKDS in the installation data field for the entry in the ICSF CKDS. After you specify this global override record, you can use explicit override records to place different installation data information in the ICSF CKDS entries.

Running the conversion program

You can run the conversion program only after you initialize the master key and CKDS for ICSF. The CKDS you specify at ICSF startup must be initialized to contain NOCV-enablement keys. For information about defining keys on ICSF, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

If the PCF master key and the ICSF master key are not the same, you must define the PCF master key in the input ICSF CKDS. Define the PCF master key as an importer key-encrypting key in the input ICSF CKDS. You define the key by entering the key through the key entry hardware, or by importing the key using the ICSF key generator utility program. For information about direct key entry through the key entry hardware and the key generator utility program, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Note: Be careful defining the PCF master key in the input ICSF CKDS, because there is no programmed way to determine its validity.

You run the conversion program by submitting a batch job. On the EXEC statement, specify PGM=CSFCONV. If the PCF master key and ICSF master key are not the same in the PARM= field on the EXEC statement, specify the label of the PCF master key entry in the input ICSF CKDS. If you do not specify the parameter, the conversion program assumes that the PCF master key and ICSF master key are the same.

This example is a JCL that runs the conversion program:

```
//CKDSCONV EXEC PGM=CSFCONV,PARM='CUSPMKEY'  
//CSFVSRD DD DSN=PROD.CUSP.CKDS,DISP=SHR  
//CSFVINP DD DSN=TEST.CSF.CKDS,DISP=SHR  
//CSFVOVR DD DSN=OVERRIDE.DATA,DISP=OLD  
//CSFVNEW DD DSN=MERGED.CSF.CKDS,DISP=OLD  
//CSFVRPT DD SYSOUT=A  
//
```

In the example, CUSPMKEY is the label of the entry in the input ICSF CKDS for the PCF master key in importer key-encrypting key form. All the data sets necessary to run the conversion program are specified using DD statements.

The conversion program uses these data sets:

CSFVSRC

The PCF CKDS containing entries that you want to convert into ICSF format and place in the output ICSF CKDS. This is the source CKDS for the conversion. For a description of the PCF CKDS record format, see *OS/VS1 and OS/VS2 MVS Programmed Cryptographic Facility*.

CSFVINP

A disk copy of the input ICSF CKDS. The input CKDS should already contain the header record and the ICSF system keys and can contain other ICSF key entries. If the CKDS does not contain NOCV-enablement keys, the output ICSF CKDS will not contain NOCV-enablement keys. For more information about NOCV-enablement keys, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Note: The input ICSF CKDS does not have to be the CKDS you specify when you start ICSF.

CSFVOVR

The override file with information specifying how you want the conversion program to process PCF key entries. If no override data is required, this data set is optional. However, you must code a dummy DD statement in the JCL.

This JCL is an example of a dummy DD statement for an override file:

```
//CSFVOVR DD DUMMY,DCB=(RECFM=FB,LRECL=90,BLKSIZE=3600)
```

See [“Using the conversion program override file” on page 204](#) for a description of when and how to use the override file.

CSFVNEW

An empty disk copy of an ICSF CKDS. This is the ICSF CKDS into which the conversion program places key entries. The conversion program places key entries from the input ICSF CKDS and the PCF CKDS into the output ICSF CKDS. The data set must be defined and empty before you run the conversion program.

CSFVRPT

The activity report that the conversion program creates. The report describes any override records and gives a summary of CKDS entries that were affected by the conversion program.



Attention: If a conversion program run ends prematurely, the results of the job are unpredictable. You should not read a CKDS involved in the conversion into storage for use. For a description of the conversion program return codes, see the explanation of message CSFV0026 in [z/OS Cryptographic Services ICSF Messages](#).

When you run the conversion program, the program produces information about the conversion in an activity report. The activity report lists each override entry, the action each override entry applies to the input PCF CKDS, and any error messages. The activity report also lists the data sets that were used in the conversion and a summary of processing. The summary of processing contains totals that apply to CKDS entries in the conversion program job.

Example of a Conversion Initial Activity Report

Figure 11 on [page 211](#) is an example of an activity report with five explicit override records and no global override records.

```

CRYPTOGRAPHIC CONVERSION ACTIVITY REPORT                                DATE: 2001/06/01 (YYYY/MM/DD) TIME: 10:13:09 PAGE: 1
OVERRIDE--> CRLABEL3 LOCAL      OPINENC      Used in transfers to Main Office.
>>>CSFV0192 TYPE FOR KEY ENTRY CRLABEL3 LOCAL CONVERTED TO OPINENC.
>>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY CRLABEL3 OPINENC SET TO Used in transfers to Main Office

OVERRIDE--> CRLABEL3 REMOTE      IPINENC      Used in receiving from the Main Office
>>>CSFV0192 TYPE FOR KEY ENTRY CRLABEL3 REMOTE CONVERTED TO IPINENC.
>>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY CRLABEL3 IPINENC SET TO Used in receiving from the Main Office.

OVERRIDE--> KGLABEL1 LOCAL      OPINENC      Used for sending encrypted PINs
>>>CSFV0292 NO KEY ENTRY FOUND FOR KGLABEL1 LOCAL.

OVERRIDE--> LOLABEL2                                Valid for January 2001
>>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY LOLABEL2 EXPORTER SET TO Valid for January 2001.

OVERRIDE--> ZZZZ1      LOCAL      Y Eliminate Key from output CKDS
>>>CSFV0382 ADD/CHANGE SPECIFICATIONS IGNORED ON OVERRIDE ENTRY. BYPASS_FLAG VALUE IS "Y".
>>>CSFV0292 NO KEY ENTRY FOUND FOR ZZZZ1 LOCAL.

>>>CSFV0012 CONVERSION PROCESSING COMPLETED. RETURN CODE = 4.

```

```

CRYPTOGRAPHIC CONVERSION ACTIVITY REPORT                                DATE: 2001/06/01 (YYYY/MM/DD) TIME: 10:13:09 PAGE: 2

CKDS DDNAME      Data Set Name
-----
CSFVSRC          PROD.CUSP.CKDS
CSFVINP          TEST.CSF.CKDS
CSFVNEW          MERGED.CSF.CKDS

PROCESSING SUMMARY

Source CKDS Entries      Converted Entries      ICSF Entries
-----
LOCAL                    4      * Candidates                    16      + Changed Input Entries      2
REMOTE                  4      Bypassed by Overrides            ( 0)      + Unchanged Input Entries    13
CROSS                   4
-----
* TOTAL Source Entries    12      TOTAL Converted Entries    16      + TOTAL ICSF Input Entries   15
                                           + Entries Added from Source   14
                                           + Entries Bypassed by Exit   ( 0)
                                           -----
                                           TOTAL Output ICSF Entries    29

* One Source CKDS CROSS entry converts to two Candidates.
+ Total Converted Entries = Changed Input Entries + Entries Added from Source.

```

Figure 11. Example of a Conversion Initial Activity Report

In the report, the first override record specifies that when the conversion program converts a PCF entry labeled CRLABEL3 with a key type of local, the program should convert the entry into an output PIN-encrypting key. The conversion program also places the information Used in transfers to Main Office in the installation data field of the output ICSF CKDS entry.

The second override record specifies that when the conversion program converts a PCF entry labeled CRLABEL3 with a key type of remote, the program should convert the key into an input PIN-encrypting key. The conversion program places the information Used in receiving from the Main Office in the installation data field of the output ICSF CKDS entry.

The label specified by the third override record does not exist in the PCF CKDS. Therefore, the conversion program ignores this override record.

The fourth override record specifies that when the conversion program converts a PCF entry labelled LOLABEL2, the program should place the information Valid for January 2001 in the installation data field of the output ICSF CKDS record.

The label specified by the fifth override record does not exist on the PCF CKDS that the conversion program is converting. Therefore, the conversion program ignores this override record.

The message that the conversion processing has been completed is followed by a return code. Return codes are listed under message CSFV0026 in [z/OS Cryptographic Services ICSF Messages](#).

After describing the five override records, the conversion report lists the data sets the conversion program used in the conversion. PROD.CUSP.CKDS is the PCF CKDS that the program converted. TEST.CSF.CKDS is the input ICSF CKDS containing the ICSF entries input during the conversion. MERGED.CSF.CKDS is the output ICSF CKDS where the conversion program placed the converted entries.

Then the activity report lists totals pertaining to the conversion. The PCF CKDS has a total of 12 entries: four with a key type of local, four with a key type of remote, and four with a key type of cross. Because the conversion of each cross key entry results in two ICSF entries, the total ICSF entries that are candidates for conversion from the PCF is 16. None of these candidates was bypassed because of an override record, so 16 PCF entries were converted.

There were 15 entries in the input ICSF CKDS, and two of these entries were updated because they had identical key labels in the PCF CKDS. Fourteen new output ICSF CKDS entries were added from the PCF CKDS. The total number of entries in the output ICSF CKDS is 29. This includes the 15 entries in the input ICSF CKDS and the 14 entries added from the PCF CKDSN. No entries were bypassed because of the conversion program exit.

Example of a Conversion Update Activity Report

Figure 12 on page 212 is an example of an activity report with a global override record that has the conversion program bypass all the entries in the PCF CKDS. Then two override records are used to convert specific entries.

CRYPTOGRAPHIC CONVERSION ACTIVITY REPORT
OVERRIDE-->
>>>CSFV0172 ALL ENTRIES BYPASSED.
OVERRIDE--> CRLABEL3 LOCAL OPINENC
>>>CSFV0222 KEY ENTRY CRLABEL3 LOCAL NOT BYPASSED.
>>>CSFV0192 TYPE FOR KEY ENTRY CRLABEL3 LOCAL CONVERTED TO OPINENC.
>>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY CRLABEL3 OPINENC SET TO Used in transfers to Main Office.
OVERRIDE--> LOLABEL2
>>>CSFV0222 KEY ENTRY LOLABEL2 LOCAL NOT BYPASSED.
>>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY LOLABEL2 EXPORTER SET TO Valid for January 2001.
>>>CSFV0012 CONVERSION PROCESSING COMPLETED. RETURN CODE = 0.

CRYPTOGRAPHIC CONVERSION ACTIVITY REPORT
DATE: 2001/06/01 (YYYY/MM/DD) TIME: 10:13:09 PAGE: 2
CKDS DDNAME Data Set Name
CSFVSRC PROD.PCF.CKDS
CSFVINP INTEST.CSF.CKDS
CSFVNEW NEWTEST.CSF.CKDS
PROCESSING SUMMARY
Source CKDS Entries Converted Entries ICSF Entries
LOCAL 4 * Candidates 16 + Changed Input Entries 1
REMOTE 4 Bypassed by Overrides (14) Unchanged Input Entries 27
CROSS 4
* TOTAL Source Entries 12 TOTAL Converted Entries 2 + TOTAL ICSF Input Entries 28
+ Entries Added from Source 1
Entries Bypassed by Exit (0)
TOTAL Output ICSF Entries 29
* One Source CKDS CROSS entry converts to two Candidates.
+ Total Converted Entries = Changed Input Entries + Entries Added from Source.

Figure 12. Example of a Conversion Update Activity Report

The first override record specifies that the conversion program bypass all the entries in the PCF CKDS. The second override record specifies that the conversion program convert a PCF entry labeled CRLABEL3 with a key type of local into an output PIN-encrypting key. This second override record also instructs the conversion program to place the phrase `Used in transfers to Main Office` in the installation data field of the output ICSF CKDS entry. The third override record specifies that the conversion program convert a PCF entry labeled LOLABEL2 and place `Valid for January 2001` in the installation data field of the output ICSF CKDS entry.

After describing the three override records, the conversion report lists the data sets the conversion program used in the conversion. `PROD.PCF.CKDS` is the PCF CKDS that the program converted. `INTEST.CSF.CKDS` is the input ICSF CKDS that contains the ICSF entries input containing the ICSF entries

input during the conversion. NEWTEST.CSF.CKDS is the output ICSF CKDS where the conversion program placed the converted entries.

Then the activity report lists totals pertaining to the conversion. The PCF CKDS has a total of 12 entries: four with a key type of local, four with a key type of remote, and four with a key type of cross. Because the conversion of each cross key entry results in two ICSF entries, the total ICSF records that are candidates for conversion from PCF is 16. Fourteen of those 16 entries were bypassed because of the global override record.

There were 28 entries in the input ICSF CKDS, and one of these entries was updated because it had an identical key label in the PCF CKDS. The total number of entries in the output ICSF CKDS is 29. This includes the 28 entries in the input ICSF CKDS plus the one added from the PCF CKDS. No entries were bypassed because of the conversion program exit.

Appendix A. Diagnosis reference information

This appendix contains Diagnosis, Modification, or Tuning Information.

This appendix contains descriptions of the cryptographic key data set (CKDS), the public key data set (PKDS), PKA key tokens, the Cryptographic Communication Vector Table (CCVT), and Cryptographic Communication Vector Table Extension (CCVE) data areas.

For more information about key tokens, refer to [*z/OS Cryptographic Services ICSF Application Programmer's Guide*](#).

Cryptographic Key Data Set (CKDS) formats

There are three formats of the CKDS: a fixed length record format (supported by all releases of ICSF), a variable length record format (supported by ICSF FMID HCR7780 and later releases), and KDSR record format which is common to all KDS types (supported by ICSF FMID HCR77A1 and later releases). The variable length record format is only required if AES or HMAC variable-length key tokens are to be stored in the CKDS. The variable length record format can be used to store all existing symmetric keys and the AES and HMAC variable-length key tokens. KDSR is a variable length record format and supports all the function of the original variable length record format and also allows ICSF to track key usage if so configured.

Format of the CKDS header record

Table 33. Cryptographic Key Data Set Header Record Format			
Offset (Dec)	Number of Bytes	Field Name	Description
0	72	<i>Constant</i>	The field is set to binary zeros and is not used for the header record.
72	8	<i>Creation date</i>	The date the CKDS was initialized in the format <i>yyyymmdd</i> .
80	8	<i>Creation time</i>	The initial time the CKDS was created in the format <i>hhmmssst</i> .
88	8	<i>Last update date</i>	The most recent date the CKDS was updated, in the format <i>yyyymmdd</i> . This field is no longer updated when a record is updated.
96	8	<i>Last update time</i>	The most recent time the CKDS was updated, in the format <i>hhmmssst</i> . This field is no longer updated when a record is updated.
104	2	<i>Sequence number</i>	Initially zero in binary. Incremented each time the data set is processed. This field is no longer updated.

Table 33. Cryptographic Key Data Set Header Record Format (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
106	2	CKDS header flag bytes	<p>Flag bytes.</p> <p>Bit Meaning When Set On</p> <p>0 The DES master key verification pattern is valid.</p> <p>1 Reserved.</p> <p>2 The AES master key verification pattern is valid.</p> <p>3–7 Reserved.</p> <p>8 Record level authentication is disabled.</p> <p>9 The record format is variable. Set on for either variable length record format or KDSR record format.</p> <p>10 CKDS not completely written, missing records.</p> <p>11–15 Reserved.</p> <p>Note: After the bits are set on, the given values remain constant in ICSF.</p>
108	8	DES master key verification pattern	The system DES master key verification pattern.
116	8	Reserved	
124	8	AES master key verification pattern.	The AES master key verification pattern.
132	4	Record length	Length of the record in bytes. X'00000000' for fixed length record format. X'000000FC' for either variable length record format or KDSR record format.
136	1	Record version	Version number of the CKDS in binary. Set to X'00' for fixed length record format or variable length record format. Set to X'02' or greater for KDSR record format.
137	59	Reserved	
196	52	Installation data	Installation data associated with the CKDS record, as supplied by an installation exit.

Table 33. Cryptographic Key Data Set Header Record Format (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
248	4	Authentication code	The code generated by the authentication process that ensures that the CKDS record has not been modified since the last update. The authentication code is placed in the CKDS header record when the CKDS is initialized. ICSF verifies the CKDS header record authentication code whenever a CKDS is reenciphered, refreshed, or converted from PCF to ICSF format. This field is not used when the record level authentication flag is set in the CKDS header flag bytes field of the CKDS header record.

Format of the fixed-length CKDS record

Table 34. Cryptographic Key Data Set Record Format

Offset (Dec)	Number of Bytes	Field Name	Description
0	64	Key label	The key label specified by the KGUP control statement or Clear Key Input panel when the record was created. When using KGUP and the callable services, you can specify the label to identify the record. The key label is the first field of the key index.
64	8	Key type	The type of key the record contains. The master key variant for the key type enciphers the key. A KGUP control statement or Clear Key Input panel specifies the key type when the record is created. The key type is the second field of the key index.
72	8	Creation date	The initial date the CKDS record was created in the format <i>yyyymmdd</i> .
80	8	Creation time	The initial time the CKDS record was created in the format <i>hhmmssst</i> .
88	8	Last update date	The most recent date the CKDS record was updated in the format <i>yyyymmdd</i> .
96	8	Last update time	The most recent time the CKDS record was updated in the format <i>hhmmssst</i> .
104	64	Key token	The internal key token. A key token contains the key value. The value in byte four of the internal key token indicates whether the key is AES or DES. Refer to “AES internal fixed-length key token” on page 253 and “DES fixed-length key token” on page 255 for the format of the internal key token.

Table 34. Cryptographic Key Data Set Record Format (continued)			
Offset (Dec)	Number of Bytes	Field Name	Description
168	2	CKDS flag bytes	<p>Flag bytes.</p> <p>Bit Meaning When Set On</p> <p>0 The key within the key token field (offset 104) is a partial key. The key is unusable.</p> <p>1 Reserved.</p> <p>2 CKDS label must be unique.</p> <p>3–7 Reserved.</p>
170	26	Reserved	Reserved.
196	52	Installation data	Installation data associated with the CKDS record as supplied by an installation exit.
248	4	Authentication code	The code generated by the authentication process that ensures the CKDS record has not been modified since the last update. The authentication code is placed in the CKDS record when the record is created. When you refresh, reencipher, or convert a CKDS, ICSF verifies each CKDS record as ICSF performs the action. This field is not used when the record level authentication flag is set in the CKDS header flag bytes field of the CKDS header record.

Format of the variable-length CKDS record

The following table presents the format of each variable-length data set record.

Table 35. Variable-Length Cryptographic Key Data Set Record Format			
Offset (Dec)	Number of Bytes	Field Name	Description
0	64	Key label	The label or name of this CKDS record. The key label is the first field of the key index.
64	8	Key type	The type of key the record contains. The key type is the second field of the key index.
72	8	Creation date	The initial date the CKDS record was created in the format <i>yyyymmdd</i> .
80	8	Creation time	The initial time the CKDS record was created in the format <i>hhmmssst</i> .
88	8	Last update date	The most recent date the CKDS record was updated in the format <i>yyyymmdd</i> .
96	8	Last update time	The most recent time the CKDS record was updated in the format <i>hhmmssst</i> .
104	4	Record length	Length of the entire record including the key token.

Table 35. Variable-Length Cryptographic Key Data Set Record Format (continued)			
Offset (Dec)	Number of Bytes	Field Name	Description
108	60		Reserved.
168	2	CKDS flag bytes	Flag bytes. Bit Meaning When Set On 0 The key within the key token field is a partial key. The key is unusable. 1 Reserved. 2 CKDS label must be unique. 3 The record format is variable — always 1 4–7 Reserved.
170	26		Reserved.
196	52	Installation data	
248	20	Authentication code	The record authentication code.
268	variable	Key token	The key token.

Format of KDSR CKDS record

See “Common record format (KDSR)” on page 250 for more information on this CKDS record.

Public Key Data Set (PKDS) format

The PKDS record includes the PKDS header and the PKA key token. These tables show the format of each of these records.

Format of the PKDS header record

Table 36. Public Key Data Set Header Record Format			
Offset (Dec)	Number of Bytes	Field Name	Description
0	64	PKHVKEY	VSAM key of the PKDS header.
64	8		Reserved.
72	8	PKHCRDTE	The date the PKDS was created in the format <i>yyyymmdd</i> .
80	8	PKHCRTIM	The initial time the PKDS was created in the format <i>hhmmssst</i> .
88	8	PKHUPDTE	The most recent date the PKDS header was updated, in the format <i>yyyymmdd</i> . This field is no longer updated when a record is updated.

Table 36. Public Key Data Set Header Record Format (continued)			
Offset (Dec)	Number of Bytes	Field Name	Description
96	8	PKHUPTIM	The most recent time the PKDS header was updated, in the format <i>hhmmssst</i> . This field is no longer updated when a record is updated.
104	4	PKHRLN	Length of the PKDS header entry.
108	16	Reserved	
124	16	PKHSMKHP	The hash pattern of the RSA MK.
140	8	PKHEMKVP	The verification pattern of the ECC MK.
148	10	Reserved	
158	1	PKHVER	Version number of the PKDS in binary. <ul style="list-style-type: none"> Set to X'00' for PKDS record format. Set to X'02' or greater for KDSR record format.
159	1		Flag bytes. Bit Meaning When Set On 0 PKDS not completely written, missing records. 1-7 Reserved.
160	20	PKHAUTH	PKDS header authentication code.

Format of the PKDS record

Table 37. Public Key Data Set Record Format			
Offset (Dec)	Number of Bytes	Field Name	Description
0	64	PKDLABEL	Label or name of this PKDS entry.
64	8		Reserved.
72	8	PKDCRDTE	The date this PKDS record was created in the format <i>yyyymmdd</i> .
80	8	PKDCRTIM	The initial time this PKDS record was created in the format <i>hhmmssst</i> .
88	8	PKDUPDTE	The most recent date this PKDS record was updated, in the format <i>yyyymmdd</i> .
96	8	PKDUPTIM	The most recent time this PKDS record was updated, in the format <i>hhmmssst</i> .
104	4	PKDRLEN	Length of the entire PKDS record entry.

Table 37. Public Key Data Set Record Format (continued)			
Offset (Dec)	Number of Bytes	Field Name	Description
108	52	PKDUDATA	User data.
160	20	PKDAUTH	The entry authentication code.
180	1868	PKDTOKEN	The public or private key token.

Token data set (TKDS) format

A z/OS PKCS #11 token represents a virtual cryptographic device, and can contain multiple objects. The token data set (TKDS) contains definitions of z/OS PKCS #11 tokens and token objects.

The token data set includes a header record and records for each of the individual z/OS PKCS #11 tokens and token objects. Each object associated with a particular z/OS PKCS #11 token has the token's name in its handle. The records are variable length records, and contain a length field specifying the total length of the record.

Format of the header record of the token data set

There is one header record for the token data set.

Table 38. Format of the header record of the token data set		
Offset (decimal)	Length of field (bytes)	Description
0	72	VSAM key of the TKDS header. Bytes 0-39: Binary zeros. Bytes 40-43: EBCDIC 'THDR'. Bytes 44-71: Binary zeros.
72	8	Reserved for IBM's use.
80	8	The date that the TKDS was created, in the format <i>yyyymmdd</i> .
88	8	The time that the TKDS was created, in the format <i>hhmmssst</i> .
96	8	The most recent date that the TKDS header was updated, in the format <i>yyyymmdd</i> . This field is no longer updated when a record is updated.
104	8	The most recent time that the TKDS header was updated, in the format <i>hhmmssst</i> . This field is no longer updated when a record is updated.
112	4	Length of the TKDS header record.
116	16	P11 MKVP.
132	16	RCS MKVP.
148	6	Reserved.

Table 38. Format of the header record of the token data set (continued)		
Offset (decimal)	Length of field (bytes)	Description
154	1	Version number of the TKDS in binary. <ul style="list-style-type: none"> Set to X'00' for fixed length record format or variable length record format. Set to X'02' or greater for KDSR record.
155	1	Flag bytes. Bit Meaning When Set On 0 TKDS not completely written, missing records. 1-7 Reserved.

Format of the token and object records

Each z/OS PKCS #11 token record and token object record begins with the same 188 bytes of data. The remainder of the record is specific to the token or object.

Common section of the token and object records

Every record in the token data set, with the exception of the header record, begins with these 188 bytes of data.

Table 39. Format of the common section of the token and object records		
Offset (decimal)	Length of field (bytes)	Description
0	72	Handle of token or object Bytes 0-31: Token name Bytes 32-39: Sequence number Byte 40: Character "T" for clear token object Character "Y" for secure token object Bytes 41-43 Blank characters Bytes 44-71: Binary zeros
72	8	Reserved for IBM's use
80	8	The date that this record was created, in the format <i>yyyymmdd</i>
88	8	The time that this record was created, in the format <i>hhmmssst</i>
96	8	The most recent date that this record was updated, in the format <i>yyyymmdd</i>
104	8	The most recent time that this record was updated, in the format <i>hhmmssst</i>
112	4	Length of the entire TKDS record entry

Table 39. Format of the common section of the token and object records (continued)

Offset (decimal)	Length of field (bytes)	Description
116	20	Reserved for IBM's use
136	52	User data
188	variable	The TKDS token or object (see mappings)

Format of the token-specific section of the token record

Each z/OS PKCS #11 token record begins with the 188 bytes. The remainder of the record contains the contents of the token. The mapping of the record shows the data beginning at offset 0, which is its offset into the token-specific portion of the record; however, that portion of the record is at an offset of 188 into the entire record.

Table 40. Format of the unique section of the token record

Offset (decimal) 188 +	Length of field (bytes)	Description
0	4	Eye catcher for token: "TOKEN"
4	2	Version number of structure: EBCDIC '00'
6	2	Length of structure in bytes
8	4	Reserved for IBM's use. Must be zeros.
12	8	Last assigned sequence number
20	32	Manufacturer identification
52	16	Model
68	16	Serial number
84	8	Date of the most recent update to this token, expressed as Coordinated Universal Time (UTC) in the format <i>yyyymmdd</i> . This includes any update to token information or to a token object.
92	8	Time of the most recent update to this token, expressed as Coordinated Universal Time (UTC) in the format <i>hhmmssst</i> . This includes any update to token information or to a token object.
100	44	Reserved for IBM's use
144		End of token

Format of the object-specific sections of the token object records

The following classes of objects can be associated with a z/OS PKCS #11 token:

- Certificate
- Public key
- Private key
- Secret key
- Data objects
- Domain parameters

The token object record for each begins with the common section described “Common section of the token and object records” on page 222, followed by a section specific to the class of object. Each of the object-specific sections begins with a 12-byte header record, followed by a variable-length section. Each 12-byte header contains a 4-byte flag field that has the same mapping for all classes of objects.

This 4-byte flag field occurs in the object header section of each token object record.

Table 41. Format of the token object flags		
Flag bytes	Field name	Description
Flag byte 1		
Bit 0	OBJ_IS_TOKOBJ	When on, the object is a token object. When off, the object is a session object.
Bit 1	OBJ_IS_PRVOBJ	When on, the object is a private object. When off, the object is a public object.
Bit 2	OBJ_IS_MODOBJ	When on, the object is modifiable.
Bit 3	KEY_DERIVE	When on, the key supports key derivation.
Bit 4	KEY_LOCAL	When on, the key was generated locally.
Bit 5	KEY_ENCRYPT	When on, the key supports encryption.
Bit 6	KEY_DECRYPT	When on, the key supports decryption.
Bit 7	KEY_VERIFYA	When on, the key supports verification where the signature is an appendix to the data.
Flag byte 2		
Bit 0	KEY_VERIFYR	When on, the key supports verification where the data is recovered from the signature
Bit 1	KEY_SIGA	When on, the key supports signatures where the signature is an appendix to the data.
Bit 2	KEY_SIGR	When on, the key supports signatures where the data is recovered from the signature.
Bit 3	KEY_WRAP	When on, the key supports wrapping.
Bit 4	KEY_UNWRAP	When on, the key supports unwrapping.
Bit 5	KEY_EXTRACT	When on, the key is extractable.
Bit 6	KEY_IS_SENSITIVE	When on, the key is sensitive.
Bit 7	KEY_IS_ALWAYS_SENSITIVE	When on, the SENSITIVE attribute (KEY_IS_SENSITIVE) is always true.
Flag byte 3		
Bit 0	KEY_NEVER_EXTRACT	When on, the EXTRACTABLE attribute (KEY_EXTRACT) is never true. When off, the EXTRACTABLE attribute (KEY_EXTRACT) can be true.
Bit 1	OBJ_IS_TRUSTED	When on, the certificate can be trusted for the application for which it was created.

Table 41. Format of the token object flags (continued)		
Flag bytes	Field name	Description
Bit 2	CERT_IS_DEFAULT	When on, this is the default certificate.
Bit 3	FIPS140	When on, key is only to be used in a FIPS-compliant manner.
Bit 4	KEY_IS_SECURE	When on, key is a secure PKCS #11 key.
Bit 5	KEY_ATTRBOUND	When on, key is attribute bound.
Bit 6	WRAP_WITH_TRUSTED	When on, key may only be wrapped with another key marked OBJ_IS_TRUSTED
Bit 7	KEY_IS_ALWAYS_SECURE	When on, KEY_IS_SECURE is always true.
Flag byte 4		
Bit 0	KEY_IS_REGIONAL	When on, key requires a regional cryptographic server.
Bits 1-7		Reserved for IBM's use.

Table 42. Format of the token certificate object		
Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for certificate object: "CERT"
4	2	Version: EBCDIC '00'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	TYPE attribute: X'00000000': CKC_X_509
16	4	Certificate category 0 Undefined 1 Token user 2 Certificate authority 3 Other entity
20	8	Reserved for IBM's use
28	32	Reserved for IBM's use
60	2	Length of SUBJECT attribute in bytes (<i>aa</i>)
62	2	Length of ID attribute in bytes (<i>bb</i>)

Table 42. Format of the token certificate object (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
64	2	Length of ISSUER attribute in bytes (<i>cc</i>)
66	2	Length of SERIAL_NUMBER attribute in bytes (<i>dd</i>)
68	2	Length of VALUE attribute in bytes (<i>ee</i>)
70	2	Length of LABEL attribute in bytes (<i>ff</i>)
72	2	Length of APPLICATION attribute in bytes (<i>gg</i>)
74	22	Reserved for IBM's use
96	4	Offset of SUBJECT attribute in bytes
100	4	Offset of ID attribute in bytes
104	4	Offset of ISSUER attribute in bytes
108	4	Offset of SERIAL_NUMBER attribute in bytes
112	4	Offset of VALUE attribute in bytes
116	4	Offset of LABEL attribute in bytes
120	4	Offset of APPLICATION attribute in bytes
124	44	Reserved for IBM's use
168	$aa + bb + cc + dd + ee + ff + gg$	Certificate attributes (variable length)
$168 + aa + bb + cc + dd + ee + ff + gg$		End of certificate object

Table 43. Format of the token public key object (Version 0)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for public key object: "PUBK"
4	2	Version: EBCDIC '00'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	TYPE attribute: CKK_RSA
16	8	Start date for the key, in the format <i>yyyymmdd</i>
24	8	End date for the key, in the format <i>yyyymmdd</i>

Table 43. Format of the token public key object (Version 0) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
72	4	Length in bits of modulus n
76	256	Modulus n
332	256	Reserved
588	256	Public exponent e
844	256	Reserved
1100	2	Length of SUBJECT attribute in bytes (<i>aa</i>)
1102	2	Length of ID attribute in bytes (<i>bb</i>)
1104	2	Length of LABEL attribute in bytes (<i>cc</i>)
1106	2	Length of APPLICATION attribute in bytes (<i>dd</i>)
1108	20	Reserved
1128	4	Offset of SUBJECT attribute in bytes
1132	4	Offset of ID attribute in bytes
1136	4	Offset of LABEL attribute in bytes
1140	4	Offset of APPLICATION attribute in bytes
1144	40	Reserved
1184	<i>aa+bb+cc+dd</i>	Public key attributes (variable length)
1184 + <i>aa+bb+cc+dd</i>		End of public key object

Table 44. Format of the token public key object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for public key object: "PUBK"
4	2	Version: EBCDIC '01'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	TYPE attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH

Table 44. Format of the token public key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
16	8	Start date for the key, in the format <i>yyyymmdd</i>
24	8	End date for the key, in the format <i>yyyymmdd</i>
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus <i>n</i>
76	512	Modulus <i>n</i>
588	512	Public exponent <i>e</i>
Algorithm-specific section (DSA)		
72	4	Length in bits of prime <i>p</i>
76	128	Reserved
204	128	Prime <i>p</i>
332	128	Reserved
460	128	Base <i>g</i>
588	128	Reserved
716	128	Value <i>y</i>
844	20	Reserved
864	20	Subprime <i>q</i>
884	216	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime <i>p</i>
76	256	Prime <i>p</i>
332	256	Base <i>g</i>
588	256	Value <i>y</i>
844	256	Reserved
Algorithm-specific section (EC)		

Table 44. Format of the token public key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	128	Reserved
204	136	EC point Q (DER encoded)
340	760	Reserved
Variable length attribute section		
1100	2	Length of SUBJECT attribute in bytes (<i>aa</i>)
1102	2	Length of ID attribute in bytes (<i>bb</i>)
1104	2	Length of LABEL attribute in bytes (<i>cc</i>)
1106	2	Length of APPLICATION attribute in bytes (<i>dd</i>)
1108	20	Reserved
1128	4	Offset of SUBJECT attribute in bytes
1132	4	Offset of ID attribute in bytes
1136	4	Offset of LABEL attribute in bytes
1140	4	Offset of APPLICATION attribute in bytes
1144	40	Reserved
1184	<i>aa+bb+cc+dd</i>	Public key attributes (variable length)

Table 44. Format of the token public key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
1184 <i>+aa+bb+cc+dd</i>		End of public key object

Table 45. Format of the token public key object (Version 2)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for public key object: "PUBK"
4	2	Version: EBCDIC '02'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	TYPE attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key, in the format <i>yyyymmdd</i>
24	8	End date for the key, in the format <i>yyyymmdd</i>
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus <i>n</i>
76	512	Modulus <i>n</i>
588	512	Public exponent <i>e</i>
Algorithm-specific section (DSA)		
72	4	Length in bits of prime <i>p</i>
76	256	Prime <i>p</i>
332	256	Base <i>g</i>
588	256	Value <i>y</i>
844	8	Reserved
852	32	Subprime <i>q</i>
884	216	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime <i>p</i>

Table 45. Format of the token public key object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
76	256	Prime p
332	256	Base g
588	256	Value y
844	256	Reserved
Algorithm-specific section (EC)		
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	128	Reserved
204	136	EC point Q (DER encoded)
340	760	Reserved
Variable length attribute section		
1100	2	Length of SUBJECT attribute in bytes (aa)
1102	2	Length of ID attribute in bytes (bb)
1104	2	Length of LABEL attribute in bytes (cc)
1106	2	Length of APPLICATION attribute in bytes (dd)
1108	20	Reserved
1128	4	Offset of SUBJECT attribute in bytes
1132	4	Offset of ID attribute in bytes

Table 45. Format of the token public key object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
1136	4	Offset of LABEL attribute in bytes
1140	4	Offset of APPLICATION attribute in bytes
1144	40	Reserved
1184	$aa+bb+cc+dd$	Public key attributes (variable length)
1184 + $aa+bb+cc+dd$		End of public key object

Table 46. Format of the token public key object (Version 3)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for public key object: "PUBK"
4	2	Version: EBCDIC '03'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	TYPE attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key, in the format $yyyymmdd$
24	8	End date for the key, in the format $yyyymmdd$
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	2	Reserved
38	2	Length of secure key material in bytes (ee)
40	4	Offset to secure key material in bytes
44	28	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus n
76	512	Modulus n
588	512	Public exponent e
Algorithm-specific section (DSA)		
72	4	Length in bits of prime p
76	256	Prime p

Table 46. Format of the token public key object (Version 3) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
332	256	Base g
588	256	Value y
844	8	Reserved
852	32	Subprime q
884	216	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	256	Value y
844	256	Reserved
Algorithm-specific section (EC)		
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	128	Reserved
204	136	EC point Q (DER encoded)
340	760	Reserved
Variable length attribute section		

Table 46. Format of the token public key object (Version 3) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
1100	2	Length of SUBJECT attribute in bytes (<i>aa</i>)
1102	2	Length of ID attribute in bytes (<i>bb</i>)
1104	2	Length of LABEL attribute in bytes (<i>cc</i>)
1106	2	Length of APPLICATION attribute in bytes (<i>dd</i>)
1108	20	Reserved
1128	4	Offset of SUBJECT attribute in bytes
1132	4	Offset of ID attribute in bytes
1136	4	Offset of LABEL attribute in bytes
1140	4	Offset of APPLICATION attribute in bytes
1144	40	Reserved
1184	<i>aa+bb+cc+dd+ee</i>	Public key attributes (variable length)
1184 + <i>aa+bb+cc+dd+ee</i>		End of public key object

Table 47. Format of the token private key object (Version 0)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for private key object: "PRIV"
4	2	Version: EBCDIC '00'
6	2	Length of object (in bytes)
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	Type attribute: CKK_RSA
16	8	Start date for the key (in the format <i>yyyymmdd</i>)
24	8	End date for the key (in the format <i>yyyymmdd</i>)
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
72	4	Length in bits of modulus <i>n</i>
76	256	Modulus: modulus <i>n</i>
332	256	Reserved

Table 47. Format of the token private key object (Version 0) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
588	256	Public exponent e
844	256	Reserved
1100	32	Reserved
1132	256	Private exponent d
1388	256	Reserved
1644	136	Prime p
1780	128	Reserved
1908	128	Prime q
2036	128	Reserved
2172	136	Private exponent d modulo p-1
2300	128	Reserved
2428	128	Private exponent d modulo q-1
2556	128	Reserved
2684	136	CRT coefficient q-1 mod p
2820	128	Reserved
2948	2	Length of SUBJECT attribute in bytes (xx)
2950	2	Length of ID attribute in bytes (yy)
2952	2	Length of LABEL attribute in bytes (zz)
2954	2	Length of APPLICATION attribute in bytes (ww)
2956	20	Reserved
2976	4	Offset of SUBJECT attribute in bytes
2980	4	Offset of ID attribute in bytes
2984	4	Offset of LABEL attribute in bytes
2988	4	Offset of APPLICATION attribute in bytes
2992	40	Reserved
3032	xx+yy+zz+ww	Private key attributes (variable length)
3032 +xx+yy+zz+ww		End of private key object

Table 48. Format of the token private key object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		

Table 48. Format of the token private key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
0	4	Eye catcher for private key object: "PRIV"
4	2	Version: EBCDIC '01'
6	2	Length of object (in bytes)
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	Type attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key (in the format <i>yyyymmdd</i>)
24	8	End date for the key (in the format <i>yyyymmdd</i>)
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus <i>n</i>
76	512	Modulus: modulus <i>n</i>
588	512	Public exponent <i>e</i>
1100	32	Reserved
1132	512	Private exponent <i>d</i>
1644	264	Prime <i>p</i>
1908	256	Prime <i>q</i>
2164	264	Private exponent <i>d</i> modulo <i>p</i> -1
2428	256	Private exponent <i>d</i> modulo <i>q</i> -1
2684	264	CRT coefficient <i>q</i> -1 mod <i>p</i>
Algorithm-specific section (DSA)		
72	4	Length in bits of prime <i>p</i>
76	128	Reserved
204	128	Prime <i>p</i>
332	128	Reserved
460	128	Base <i>g</i>
588	236	Reserved
824	20	Value <i>x</i>
844	20	Reserved
864	20	Subprime <i>q</i>

Table 48. Format of the token private key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
884	2064	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	236	Reserved
824	20	Value x
844	2104	Reserved
Algorithm-specific section (EC)		
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	64	Reserved
140	66	Value d
206	2742	Reserved
Variable length attribute section		
2948	2	Length of SUBJECT attribute in bytes (xx)
2950	2	Length of ID attribute in bytes (yy)
2952	2	Length of LABEL attribute in bytes (zz)

Table 48. Format of the token private key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
2954	2	Length of APPLICATION attribute in bytes (<i>ww</i>)
2956	20	Reserved
2976	4	Offset of SUBJECT attribute in bytes
2980	4	Offset of ID attribute in bytes
2984	4	Offset of LABEL attribute in bytes
2988	4	Offset of APPLICATION attribute in bytes
2992	40	Reserved
3032	<i>xx+yy+zz+ww</i>	Private key attributes (variable length)
3032 + <i>xx+yy+zz+ww</i>		End of private key object

Table 49. Format of the token private key object (Version 2)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for private key object: "PRIV"
4	2	Version: EBCDIC '02'
6	2	Length of object (in bytes)
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	Type attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key (in the format <i>yyyymmdd</i>)
24	8	End date for the key (in the format <i>yyyymmdd</i>)
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus <i>n</i>
76	512	Modulus: modulus <i>n</i>
588	512	Public exponent <i>e</i>
1100	32	Reserved
1132	512	Private exponent <i>d</i>

Table 49. Format of the token private key object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
1644	264	Prime p
1908	256	Prime q
2164	264	Private exponent d modulo $p-1$
2428	256	Private exponent d modulo $q-1$
2684	264	CRT coefficient $q-1 \bmod p$
Algorithm-specific section (DSA)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	224	Reserved
812	32	Value x
844	8	Reserved
852	32	Subprime q
884	2064	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	256	Value x
844	4	Length in bits of value x
848	2100	Reserved
Algorithm-specific section (EC)		

Table 49. Format of the token private key object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	64	Reserved
140	66	Value <i>d</i>
206	2742	Reserved
Variable length attribute section		
2948	2	Length of SUBJECT attribute in bytes (xx)
2950	2	Length of ID attribute in bytes (yy)
2952	2	Length of LABEL attribute in bytes (zz)
2954	2	Length of APPLICATION attribute in bytes (ww)
2956	20	Reserved
2976	4	Offset of SUBJECT attribute in bytes
2980	4	Offset of ID attribute in bytes
2984	4	Offset of LABEL attribute in bytes
2988	4	Offset of APPLICATION attribute in bytes
2992	40	Reserved
3032	xx+yy+zz+ww	Private key attributes (variable length)

Table 49. Format of the token private key object (Version 2) (continued)		
Offset (decimal) 188 +	Length of field (bytes)	Description
3032 +xx+yy+zz+ww+ee		End of private key object

Table 50. Format of the token private key object (Version 3)		
Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for private key object: "PRIV"
4	2	Version: EBCDIC '03'
6	2	Length of object (in bytes)
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	Type attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key (in the format <i>yyyymmdd</i>)
24	8	End date for the key (in the format <i>yyyymmdd</i>)
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	2	Reserved
38	2	Length of secure key material (ee)
40	4	Offset to secure key material in bytes
44	28	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus <i>n</i>
76	512	Modulus: modulus <i>n</i>
588	512	Public exponent <i>e</i>
1100	32	Reserved
1132	512	Private exponent <i>d</i>
1644	264	Prime <i>p</i>
1908	256	Prime <i>q</i>
2164	264	Private exponent <i>d</i> modulo <i>p</i> -1
2428	256	Private exponent <i>d</i> modulo <i>q</i> -1
2684	264	CRT coefficient <i>q</i> -1 mod <i>p</i>

Table 50. Format of the token private key object (Version 3) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
Algorithm-specific section (DSA)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	224	Reserved
812	32	Value x
844	8	Reserved
852	32	Subprime q
884	2064	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime p
76	256	Prime p
332	256	Base g
588	256	Value x
844	4	Length in bits of value x
848	2100	Reserved
Algorithm-specific section (EC)		

Table 50. Format of the token private key object (Version 3) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	64	Reserved
140	66	Value <i>d</i>
206	2742	Reserved
Variable length attribute section		
2948	2	Length of SUBJECT attribute in bytes (xx)
2950	2	Length of ID attribute in bytes (yy)
2952	2	Length of LABEL attribute in bytes (zz)
2954	2	Length of APPLICATION attribute in bytes (ww)
2956	20	Reserved
2976	4	Offset of SUBJECT attribute in bytes
2980	4	Offset of ID attribute in bytes
2984	4	Offset of LABEL attribute in bytes
2988	4	Offset of APPLICATION attribute in bytes
2992	40	Reserved
3032	xx+yy+zz+ww+ee	Private key attributes (variable length)

Table 50. Format of the token private key object (Version 3) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
3032 +xx+yy+zz+ww+ee		End of private key object

Table 51. Format of the token secret key object (Version 0)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for secret key object: "SECK"
4	2	Version: EBCDIC '00'
6	2	Length of the object in bytes
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	Type of key: CKK_DES, CKK_DES2, CKK_DES3, CKK_AES
16	8	Start date for the key (in the format <i>yyyymmdd</i>)
24	8	End date for the key (in the format <i>yyyymmdd</i>)
32	4	Key generate mechanism CK_UNAVAILABLE_INFORMATION
36	2	Length of the key in bytes
38	32	Reserved
70	64	VALUE: value of the key
134	538	Reserved
672	4	Usage counter field
676	2	Reserved
678	2	Length of LABEL attribute in bytes (xx)
680	2	Length of APPLICATION attribute in bytes (yy)
682	2	Length of the ID attribute in bytes (zz)
684	20	Reserved
704	4	Offset of LABEL attribute in bytes
708	4	Offset of APPLICATION attribute in bytes
712	4	Offset of the ID attribute in bytes
716	40	Reserved
756	xx+yy+zz	Secret key attributes (variable length)

Table 51. Format of the token secret key object (Version 0) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
756 +xx+yy+zz		End of secret key object

Table 52. Format of the token secret key object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for secret key object: "SECK"
4	2	Version: EBCDIC '01'
6	2	Length of the object in bytes
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	Type of key: CKK_DES, CKK_DES2, CKK_DES3, CKK_BLOWFISH, CKK_RC4, CKK_GENERIC_SECRET, and CKK_AES.
16	8	Start date for the key (in the format <i>yyyymmdd</i>)
24	8	End date for the key (in the format <i>yyyymmdd</i>)
32	4	Key generate mechanism CK_UNAVAILABLE_INFORMATION
36	2	Length of the key in bytes
38	32	Reserved
70	256	VALUE: value of the key
326	346	Reserved
672	4	Usage counter field
676	2	Reserved
678	2	Length of LABEL attribute in bytes (xx)
680	2	Length of APPLICATION attribute in bytes (yy)
682	2	Length of the ID attribute in bytes (zz)
684	20	Reserved
704	4	Offset of LABEL attribute in bytes
708	4	Offset of APPLICATION attribute in bytes
712	4	Offset of the ID attribute in bytes
716	40	Reserved

Table 52. Format of the token secret key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
756	xx+yy+zz	Secret key attributes (variable length)
756 +xx+yy+zz		End of secret key object

Table 53. Format of the token secret key object (Version 3)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for secret key object: "SECK"
4	2	Version: EBCDIC '03'
6	2	Length of the object in bytes
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	Type of key: CKK_DES, CKK_DES2, CKK_DES3, CKK_BLOWFISH, CKK_RC4, CKK_GENERIC_SECRET, and CKK_AES.
16	8	Start date for the key (in the format <i>yyyymmdd</i>)
24	8	End date for the key (in the format <i>yyyymmdd</i>)
32	4	Key generate mechanism CK_UNAVAILABLE_INFORMATION
36	2	Length of the key in bytes
38	2	Length of secure key material (ee)
40	4	Offset to secure key material in bytes
44	26	Reserved
70	256	VALUE: value of the key
326	342	Reserved
668	1	Key field flags: Bit 0 Key check value present Bits 1-7 Reserved for IBM's use
672	4	Usage counter field
676	2	Reserved
678	2	Length of LABEL attribute in bytes (xx)

Table 53. Format of the token secret key object (Version 3) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
680	2	Length of APPLICATION attribute in bytes (yy)
682	2	Length of the ID attribute in bytes (zz)
684	20	Reserved
704	4	Offset of LABEL attribute in bytes
708	4	Offset of APPLICATION attribute in bytes
712	4	Offset of the ID attribute in bytes
716	40	Reserved
756	$xx+yy+zz+ee$	Secret key attributes (variable length)
756 + $xx+yy+zz+ee$		End of secret key object

Table 54. Format of the token domain parameters object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for token domain object: "DOMP"
4	2	Version: EBCDIC '01'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	TYPE attribute: CKK_DSA or CKK_DH
16	28	Reserved
Algorithm-specific section (DSA)		
44	4	Length in bits of prime p
48	128	Reserved
176	128	Prime p
304	128	Reserved
432	128	Base g
560	20	Reserved
580	20	Subprime q
600	636	Reserved
Algorithm-specific section (DH)		
44	4	Length in bits of prime p

Table 54. Format of the token domain parameters object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
48	4	Reserved
52	256	Prime p
308	256	Reserved
564	256	Base g
820	416	Reserved
Variable length attribute section		
1236	2	Length of LABEL attribute in bytes (aa)
1238	2	Length of APPLICATION attribute in bytes (bb)
1240	20	Reserved
1260	4	Offset of LABEL attribute in bytes
1264	4	Offset of APPLICATION attribute in bytes
1268	40	Reserved
1308	$aa+bb$	Domain parameters attributes (variable length)
1308 + $aa+bb$		End of domain parameters object

Table 55. Format of the token domain parameters object (Version 2)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for token domain object: "DOMP"
4	2	Version: EBCDIC '02'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	TYPE attribute: CKK_DSA
16	28	Reserved
Algorithm-specific section (DSA)		
44	4	Length in bits of prime p
48	256	Prime p
304	256	Base g
560	8	Reserved
568	32	Subprime q

<i>Table 55. Format of the token domain parameters object (Version 2) (continued)</i>		
Offset (decimal) 188 +	Length of field (bytes)	Description
600	636	Reserved
Variable length attribute section		
1236	2	Length of LABEL attribute in bytes (<i>aa</i>)
1238	2	Length of APPLICATION attribute in bytes (<i>bb</i>)
1240	20	Reserved
1260	4	Offset of LABEL attribute in bytes
1264	4	Offset of APPLICATION attribute in bytes
1268	40	Reserved
1308	<i>aa+bb</i>	Domain parameters attributes (variable length)
1308 <i>+aa+bb</i>		End of domain parameters object

<i>Table 56. Format of the token data object</i>		
Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for data object: "DATA"
4	2	Version: EBCDIC '00'
6	2	Length of object, in bytes
8	4	Flags (see Table 41 on page 224)
Object type-specific section		
12	4	Reserved for IBM's use
16	28	Reserved for IBM's use
44	2	Length of VALUE attribute in bytes (<i>aa</i>)
46	2	Length of OBJECT_ID attribute in bytes (<i>bb</i>)
48	2	Length of LABEL attribute in bytes (<i>cc</i>)
50	2	Length of APPLICATION attribute in bytes (<i>dd</i>)
52	2	Length of ID attribute in bytes (<i>ee</i>)
54	22	Reserved for IBM's use
76	4	Offset of VALUE attribute in bytes
80	4	Offset of OBJECT_ID attribute in bytes
84	4	Offset of LABEL attribute in bytes
88	4	Offset of APPLICATION attribute in bytes

Table 56. Format of the token data object (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
92	4	Offset of ID attribute in bytes
96	44	Reserved for IBM's use
140	$aa + bb + cc + dd + ee$	Data attributes (variable length)
$140 + aa + bb$ $+ cc + dd + ee$		End of data object

Common record format (KDSR)

The common record format (KDSR) is a record format for all KDS types (CKDS, PKDS, and TKDS) that allows for reference date tracking. KDSR format records were introduced in ICSF FMID HCR77A1. Version X'02' of the KDSR records have three distinct sections: a 140-byte fixed area, a variable length area that contains the cryptographic key material (key token), and a variable length metadata area that is used to store reference dates and other data.

Format of the KDSR format record (Version X'02')

KDSR record sections:

- Fixed data area – 140 bytes
- Cryptographic key material (key token) – variable length
- Metadata area – variable length

Table 57. Format of the KDSR record fixed data area

Offset (Decimal)	Number of bytes	Field name	Description
0	72	VSAM Key	CKDS Bytes 0-63 Key Label. Bytes 64-71 Key Type. PKDS Bytes 0-63 Key Label. Bytes 64-71 Reserved. TKDS Bytes 0-31 Token name. Bytes 32-39 Sequence number. Byte 40 Blank for token. Character "T" for clear token object. Character "Y" for secure token object. Bytes 41-43 Blank characters. Bytes 44-71 Binary zeros.
72	8		Reserved.
80	1	Record Version	Version of the KDSR record format.
81	1	KDS Type	1=CKDS, 2=PKDS, 3=TKDS.
82	1	KDS Flags	Bit Meaning when set On 0 The key within the key material field is a partial key. (CKDS only). 1 Label must be unique. (CKDS only).
83	1	KFP Count	Count of key fingerprints.
84	4	KDS Length	Length of the entire KDS record including key material and metadata.
88	8	Creation Date	The initial date the KDS record was created in the format <i>yyyymmdd</i> .

Table 57. Format of the KDSR record fixed data area (continued)

Offset (Decimal)	Number of bytes	Field name	Description
96	8	Creation Time	The initial time the KDS record was created in the format <i>hhmmssst</i> .
104	8	Update Date	The most recent date that this record was updated, in the format <i>yyyymmdd</i> or binary zero if the record has not been updated since creation.
112	8	Update Time	The most recent time that this record was updated, in the format <i>hhmmssst</i> or binary zero if the record has not been updated since creation.
120	4	Key Material Length	Length of the key material portion of the record.
124	4	Key Material Offset	Offset of the key material portion of the record, which is calculated from the start of the record.
128	4	Metadata Length	Length of the metadata area.
132	4	Metadata Offset	Offset of the metadata area in the record, which is calculated from the start of the record.
136	4	Reserved	Reserved.

Table 58. Format of KDSR metadata area

Offset (Decimal)	Number of bytes	Field name	Description
0	1	KDSR_MD_VERSION	
1	7		Reserved for IBM use.
8	8	KDSR_MD_REFDATE_STCKE	Reference date in STCKE format, high 8 bytes. Low bit in Byte 5 represents one second.
16	8	KDSR_MD_REFDATE	Reference date in the format <i>yyyymmdd</i> .
24	8	KDSR_MD_STARTDATE	Key material validity start date in the format <i>yyyymmdd</i> .
32	8	KDSR_MD_ENDDATE	Key material validity end date in the format <i>yyyymmdd</i> .
40	Variable		Reserved for IBM use.

Table 59. Format of KDSR variable-length metadata block

Offset (Decimal)	Number of bytes	Field name	Description
0	2	KDSR_MD_TLV_TAG	Tag for block.

Table 59. Format of KDSR variable-length metadata block (continued)			
Offset (Decimal)	Number of bytes	Field name	Description
2	2	KDSR_MD_TLV_LEN	Length of the block, which includes the length of the tag and length fields.
4	Variable	KDSR_MD_DATA	Data.

AES key token format

AES internal fixed-length key token

Fixed-length AES key tokens are 64 bytes and consist of an internal key token identifier and a token version number, reserved fields, a flag byte containing various flag bits, and a token validation value.

Depending on the flag byte, the key token either contains an encrypted key, a clear key, or the key is absent. An encrypted key is encrypted under an AES master key that is identified by a master-key verification pattern (MKVP) in the key token. The key token contains a two-byte integer that specifies the length of the clear-key value in bits, valued to 0, 128, 192, or 256, and a two-byte integer that specifies the length of the encrypted-key value in bytes, valued to 0 or 32. An LRC checksum byte of the clear-key value is also in the key token.

All keys in fixed-length AES key tokens are DATA keys. If the flag byte indicates that a control vector (CV) is present, it must be all binary zeros. An all-zero CV represents the CV value of an AES DATA key. If a key is present without a control vector in a key token, that is accepted and the key is interpreted as an AES DATA key.

The AES internal key token is the structure that is used to hold AES keys that are either encrypted with the AES master-key or in clear text format.

Table 60 on page 253 shows the format for an AES internal key token.

Table 60. AES internal fixed-length key token format		
Offset (Dec)	Length of field (Bytes)	Description
00	1	X'01' (flag indicating that this is an internal key token)
01	3	Implementation-dependent bytes (X'000000' for ICSF)
04	1	Key token version number (X'04')
05	1	Reserved - must be set to X'00'

Table 60. AES internal fixed-length key token format (continued)		
Offset (Dec)	Length of field (Bytes)	Description
06	1	<p>Flag byte</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0 Encrypted key and master key verification pattern (MKVP) are present. Off for a clear key token. On for an encrypted key token.</p> <p>1 Control vector (CV) value in this token has been applied to the key.</p> <p>2 No key is present or the AES MKVP is not present if the key is encrypted.</p> <p>3- 7 Reserved. Must be set to 0.</p>
07	1	1-byte LRC checksum of clear key value.
08	8	Master key verification pattern (MKVP). (For a clear AES key token, this value is hex zeros.)
16	32	<p>Key value, if present. Contains either:</p> <ul style="list-style-type: none"> • A 256-bit encrypted-key value. The clear key value is padded on the right with binary zeros, and the entire 256-bit value is encrypted under the AES master-key using AES CBC mode with an initialization vector of binary zeros. • A 128-bit, 192-bit, or 256-bit clear-key value left-aligned and padded on the right with binary zeros for the entire 256-bit field.
48	8	8-byte control vector. (For a clear AES key token, this value is hex zeros.)
56	2	2-byte integer that specifies the length in bits of the clear key value.
58	2	2-byte integer that specifies the length in bytes of the encrypted key value. (For a clear AES key token, this value is hex zeros.)
60	4	Token validation value (TVV). For more information, see “Token validation value” on page 254.

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 fixed-length key token

Fixed-length DES key tokens are 64 bytes and consist of a DES-enciphered key, a control vector, various flag bits, a token identifier and version number, reserved fields, and a token-validation value. An internal key-token also includes a master-key verification pattern or master-key version number, depending on the key-token version number.

If an internal fixed-length DES key-token has a key present, it contains a key multiply-enciphered by a DES master key. If an external fixed-length DES key-token has a key present, it contains a key multiply-enciphered by a key-encrypting key.

Version X'00' tokens are single-length and double-length keys for all key types except DATA. DATA key tokens are version X'00' for single-length keys and version X'01' for double-length and triple-length keys.

Table 61 on page 255 shows the format for a DES internal key token.

Table 61. DES internal fixed-length key token format		
Offset (Dec)	Length of field (Bytes)	Description
00	1	X'01' (flag indicating this is an internal key token).
01	3	Implementation-dependent bytes (X'000000' for ICSF).
04	1	Key token version number (X'00' or X'01').
05	1	Reserved (X'00').
06	1	<div>Flag byte</div> <div>Bit</div> <div>Meaning When Set On</div> <div>0 Encrypted key and master key verification pattern (MKVP) are present.</div> <div>1 Control vector (CV) value in this token has been applied to the key.</div> <div>2 Key is used for no control vector (NOCV) processing. Valid for transport keys only.</div> <div>3 Reserved.</div> <div>4 Reserved.</div> <div>5 Reserved.</div> <div>6 Reserved.</div> <div>7 Export prohibited.</div>

Table 61. DES internal fixed-length key token format (continued)		
Offset (Dec)	Length of field (Bytes)	Description
07	1	Bit Meaning When Set On 0-2 Key value encryption method. <ul style="list-style-type: none"> • 000 - The key is encrypted by using the original CCA method (ECB). • 001 - The key is encrypted by using the X9.24 enhanced method (CBC). These bits are ignored if the token contains no key or a clear key. 3-7 Reserved.
08	8	Master key verification pattern (MKVP).
16	8	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	8	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	8	The control vector (CV) for a single-length key or the left half of the control vector for a double-length key.
40	8	X'0000000000000000' if a single-length key or the right half of the control vector for a double-length operational key.
48	8	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	3	Reserved (X'000000').
59	1	Key length: Value Meaning B'00000000' Single-length key (version 0 only). B'00010000' Double-length key (version 1 only). B'00100000' Triple-length key (version 1 only).
60	4	Token validation value (TVV).

Note: A fixed-length key token that is stored in a non-KDSR 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. For more information, see [“Token validation value” on page 254](#).

[Table 62 on page 257](#) shows the format for a DES external fixed-length key token.

Table 62. Format of DES external fixed-length key tokens

Offset (Dec)	Length of field (Bytes)	Description
00	1	X'02' (flag indicating an external key token).
01	1	Reserved (X'00').
02	2	Implementation-dependent bytes (X'0000' for ICSF).
04	1	Key token version number (X'00' or X'01').
05	1	Reserved (X'00').
06	1	<p>Flag byte.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0</p> <p>Encrypted key is present.</p> <p>1</p> <p>Control vector (CV) value has been applied to the key.</p> <p>Other bits are reserved and are binary zeros.</p>
07	1	<p>Flag byte.</p> <p>Bit</p> <p>Meaning When Set On</p> <p>0-2</p> <p>Key value encryption method.</p> <ul style="list-style-type: none"> • 000 - The key is encrypted by using the original CCA method (ECB). • 001 - The key is encrypted by using the X9.24 enhanced method (CBC). <p>These bits are ignored if the token contains no key or a clear key.</p> <p>3-7</p> <p>Reserved.</p>
08	8	Reserved (X'0000000000000000').
16	8	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	8	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	8	Control vector (CV) for single-length key or left half of CV for double-length key.
40	8	X'0000000000000000' if single-length key or right half of CV for double-length key.
48	8	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	4	Reserved (X'000000').

Table 62. Format of DES external fixed-length key tokens (continued)		
Offset (Dec)	Length of field (Bytes)	Description
59	1	Key Length. Value Meaning B'00000000' Single-length key (version 0 only). B'00010000' Double-length key (version 1 only). B'00100000' Triple-length key (version 1 only).
60-63	4	Token validation value. For more information, see “Token validation value” on page 254 .

DES external key token

[Table 63 on page 258](#) shows the format for a DES external key token.

Table 63. 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 Bit Meaning When Set On 0 Encrypted key is present. 1 Control vector (CV) value has been applied to the key. Other bits are reserved and are binary zeros.
7	Bit Meaning When Set On 0-2 Key value encryption method. <ul style="list-style-type: none"> • 000 - the key is encrypted using the original CCA method (ECB). • 001 - the key is encrypted using the X9.24 enhanced method (CBC). These bits are ignored if the token contains no key or a clear key. 3-7 Reserved.

Table 63. Format of External Key Tokens (continued)	
Bytes	Description
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	B'00' Indicates single-length key (version 0 only). B'01' Indicates double-length key (version 1 only). B'10' Indicates triple-length key (version 1 only).
59 bits 4–7	B'0000'
60–63	Token validation value (see “Token validation value” on page 254 for a description).

External RKX DES key token

Table 64 on page 259 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 DES key-storage callable services (for example, Key Record Write). No other callable services use or reference an RKX key-token or key-token record.

Note: Callable services other than CSNDRKX and the DES key-storage 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 64. 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

Table 64. External RKX DES key-token format, version X'10' (continued)		
Offset	Length	Meaning
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) 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>

Note:

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 CSNDTBC callable service and then converted to internal form by the CSNDPKI callable service prior to the CSNDRKX call.

DES null key token

Table 65 on page 260 shows the format for a fixed length DES null key token.

Table 65. 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).

Table 65. Format of Null Key Tokens (continued)	
Bytes	Description
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 format for a variable-length symmetric key token. The length of the token depends on the key type and algorithm.

Table 66. Variable-length symmetric key token		
Offset (Dec)	Length of Field (Bytes)	Description
		Header
0	1	Token flag X'00' for null tokens 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' (May be X'00' for null tokens)
5	3	Reserved (X'000000')
		Wrapping information

Table 66. 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. <ul style="list-style-type: none"> 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

Table 66. Variable-length symmetric key token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
28	1	Payload version X'00' Variable-length payload X'01' Fixed-length payload All other values are reserved and must not be used.
29	1	Reserved (X'00')
		Associated data section
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')
41	1	Type of algorithm for which the key can be used X'01' DES X'02' AES X'03' HMAC

Table 66. Variable-length symmetric key token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
42	2	<p>Key type:</p> <p>For algorithm AES:</p> <p>X'0001' CIPHER</p> <p>X'0002' MAC</p> <p>X'0003' EXPORTER</p> <p>X'0004' IMPORTER</p> <p>X'0005' PINPROT</p> <p>X'0006' PINCALC</p> <p>X'0007' PINPRW</p> <p>X'0009' DKYGENKY</p> <p>X'000A' SECMSG</p> <p>For algorithm HMAC:</p> <p>X'0002' MAC</p> <p>For algorithm DES:</p> <p>X'0008' DESUSECV</p>
44	1	<p>Key-usage field count (<i>kuf</i>) - (1 byte)</p> <p>Key-usage field information defines restrictions on the use of the key.</p>

Table 66. Variable-length symmetric key token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	$kuf * 2$	<p>Key-usage fields ($kuf * 2$ bytes)</p> <ul style="list-style-type: none"> • For HMAC algorithm keys, refer to Table 68 on page 266. • For AES algorithm Key-Encrypting keys (Exporter or Importer), refer to Table 75 on page 277. • For AES algorithm CIPHER keys, refer to Table 76 on page 280. • For AES algorithm MAC keys, refer to Table 69 on page 268. • For AES algorithm PINCALC keys, refer to Table 70 on page 269. • For AES algorithm PINPROT keys, refer to Table 71 on page 270. • For AES algorithm PINPRW keys, refer to Table 72 on page 272. • For AES algorithm DKYGENKY keys, refer to Table 73 on page 273. • For AES algorithm SECMMSG keys, refer to Table 74 on page 277. • For DESUSECV keys, refer to Table 67 on page 266.
45 + $kuf * 2$	1	<p>Key-management field count (kmf) - (2 byte):</p> <ul style="list-style-type: none"> • For AES and HMAC keys: 2 (no pedigree information) or 3 (has pedigree information) • For DESUSECV keys: 1 <p>Key-management field information describes how the data is to be managed or helps with management of the key material.</p>
46 + $kuf * 2$	$kuf * 2$	<p>Key-management fields ($kmf * 2$ bytes):</p> <ul style="list-style-type: none"> • For AES and HMAC algorithm keys, refer to Table 77 on page 282. • For DESUSECV keys, refer to Table 78 on page 286.
46 + $kuf * 2 + kmf * 2$	kl	Key name
46 + $kuf * 2 + kmf * 2 + kl$	$iead$	IBM extended associated data
46 + $kuf * 2 + kmf * 2 + kl + iead$	uad	Installation-defined associated data
		Clear key or encrypted payload

Table 66. Variable-length symmetric key token (continued)		
Offset (Dec)	Length of Field (Bytes)	Description
30 + <i>adl</i>	$(pl+7)/8$	<p>Encrypted AESKW payload (internal keys): 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.</p> <p>Encrypted PKOAEP2 payload (external keys): 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>Clear key payload: When the key is clear, only the key material will be in the payload padded to the nearest byte with binary zeros.</p>

Table 67. DESUSECV key-usage fields		
Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (<i>kuf</i>): 1
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p>B'0000 0000' Reserved</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>B'0000 0000' Reserved</p> <p>All unused bits are reserved and must be zero.</p>

Table 68. HMAC algorithm key-usage fields		
Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (<i>kuf</i>): 2

Table 68. 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>1xxx xxxx Key can be used for generate.</p> <p>x1xx xxxx Key can be used for verify.</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>
47	2	<p>Key-usage field 2</p> <p>High-order byte:</p> <p>1xxx xxxx SHA-1 hash method is allowed for the key.</p> <p>x1xx xxxx SHA-224 hash method is allowed for the key.</p> <p>xx1x xxxx SHA-256 hash method is allowed for the key.</p> <p>xxx1 xxxx SHA-384 hash method is allowed for the key.</p> <p>xxxx 1xxx 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 69. AES algorithm MAC key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	<p>Key-usage field count (kuf): 2 – 3 Count is based on whether the key is DK enabled or not:</p> <p>kuf</p> <p>2 DK enabled</p> <p>3 No</p> <p>3 Yes</p>
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p>B'00xx xxxx' Undefined.</p> <p>B'01xx xxxx' Key cannot be used for generate; key can be used for verify.</p> <p>B'10xx xxxx' Key can be used for generate; key cannot be used for verify.</p> <p>B'11xx xxx*' Key can be used for generate and verify. Not valid if offset 50 is X'01'.</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 <i>uuu</i> 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>X'01' CMAC mode.</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 69. AES algorithm MAC key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
49	2	<p>Key-usage field 3</p> <p>High-order byte when DK enabled:</p> <p>X'01' PIN_OP (DKPINOP)</p> <p>X'03' PIN_ADMIN1 (DKPINAD1)</p> <p>X'04' PIN_ADMIN2 (DKPINAD2)</p> <p>All unused values are reserved and must not be used.</p> <p>Low-order byte:</p> <p>X'01' DK enabled.</p> <p>All unused values are reserved and must not be used.</p>

Table 70. AES algorithm PINCALC key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (<i>kuf</i>): 3
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p>B'00xx xxxx' Undefined.</p> <p>B'10xx xxxx' Key can be used for generate; key cannot be used for verify.</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 <i>uuu</i> are UDX-defined bits.</p> <p>All unused bits are reserved and must be zero.</p>

Table 70. AES algorithm PINCALC key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
47	2	<p>Key-usage field 2</p> <p>High-order byte:</p> <p>X'00'</p> <p>Key can be used for Cipher Block Chaining (CBC).</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>
49	2	<p>Key-usage field 3</p> <p>High-order byte when DK enabled:</p> <p>X'01'</p> <p>PIN_OP (DKPINOP)</p> <p>All unused values are reserved and must not be used.</p> <p>Low-order byte:</p> <p>X'01'</p> <p>DK enabled.</p> <p>All unused values are reserved and must not be used.</p>

Table 71. AES algorithm PINPROT key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (kuf): 3

Table 71. AES algorithm PINPROT key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p>B'00xx xxxx' Undefined.</p> <p>B'01xx xxxx' Key cannot be used for encryption; key can be used for decryption.</p> <p>B'10xx xxxx' Key can be used for encryption; key cannot be used for decryption.</p> <p>B'11xx xxxx' Undefined.</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 <i>uuu</i> 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>X'00' Key can be used for Cipher Block Chaining (CBC).</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>
49	2	<p>Key-usage field 3</p> <p>High-order byte when DK enabled:</p> <p>X'01' PIN_OP (DKPINOP)</p> <p>X'02' PIN_OPP (DKPINOPP)</p> <p>X'03' PIN_ADMIN1 (DKPINAD1)</p> <p>All unused values are reserved and must not be used.</p> <p>Low-order byte:</p> <p>X'01' DK enabled.</p> <p>All unused values are reserved and must not be used.</p>

Table 72. AES algorithm PINPRW key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (kuf): 3
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p>B'00xx xxxx' Undefined.</p> <p>B'01xx xxxx' Key cannot be used for generate; key can be used for verify.</p> <p>B'10xx xxxx' Key can be used for generate; key cannot be used for verify.</p> <p>B'11xx xxxx' Undefined.</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 <i>uuu</i> 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>X'01' CMAC mode</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>
49	2	<p>Key-usage field 3</p> <p>High-order byte when DK enabled:</p> <p>X'01' PIN_OP (DKPINOP)</p> <p>All unused values are reserved and must not be used.</p> <p>Low-order byte:</p> <p>X'01' DK enabled.</p> <p>All unused values are reserved and must not be used.</p>

Table 73. AES algorithm DKYGENKY key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	<p>Key-usage field count (kuf): 2, 4, 5, or 6.</p> <p>Count is based on the type of key to diversify (value of offset 45):</p> <p>Value at offset 45 Type of key to diversify / kuf count</p> <p>X'00' D-ALL / kuf count: 2</p> <p>X'01' D-CIPHER / kuf count: 4</p> <p>X'02' D-MAC / kuf count: 4 (not DK enabled) or 5 (DK enabled)</p> <p>X'03' D-EXP / kuf count: 6</p> <p>X'04' D-IMP / kuf count: 6</p> <p>X'05' D-PPROT / kuf count: 5</p> <p>X'06' D-PCALC / kuf count: 5</p> <p>X'07' D-PPRW / kuf count: 5</p> <p>X'08' D-SECMSG / kuf count: 4</p> <p>Each key-usage field is 2 bytes in length. The value in this field indicates how many 2-byte key usage fields follow.</p>

Table 73. AES algorithm DKYGENKY key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	<p>Key-usage field 1</p> <p>High-order byte: Defines the key type to be generated.</p> <p>X'00' Any type listed below (D-ALL)</p> <p>X'01' CIPHER (D-CIPHER)</p> <p>X'02' MAC (D-MAC)</p> <p>X'03' EXPORTER (D-EXP)</p> <p>X'04' IMPORTER (D-IMP)</p> <p>X'05' PINPROT (D-PPROT)</p> <p>X'06' PINCALC (D-PCALC)</p> <p>X'07' PINPRW (D-PPRW)</p> <p>X'08' SECMSG (D-SECMSG)</p> <p>All other values are reserved and undefined.</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 <i>uuu</i> are UDX-defined bits.</p> <p>All unused bits are reserved and must be zero.</p>

Table 73. AES algorithm DKYGENKY key associated data (continued)		
Offset (Dec)	Length of Field (Bytes)	Description
47	2	<p>Key-usage field 2: Indicates the key usage.</p> <p>High-order byte (key-usage field level of control):</p> <p>B'1xxx xxxx' The key usage fields of the key to be generated must be equal (KUF-MBE) to the related generated key usage fields that start with key usage field 3 below.</p> <p>B'0xxx xxxx' The key usage fields of the key identifier to be generated must be permitted (KUF-MBP) based on the related generated-key usage fields that start with key usage field 3 below. A key to be diversified is not permitted to have a higher level of usage than the related key usage fields permit. The key to be diversified is only permitted to have key usage that is less than or equal to the related key usage fields. The UDX-ONLY bit of the related key usage fields must always be equal in both the generating key and the generated key.</p> <p>Undefined when the value at offset 45 = X'00' (D-ALL). All other values are reserved and undefined.</p> <p>Low-order byte (key-derivation sequence level):</p> <p>X'00' DKYL0. Generate a key based on the key usage byte at offset 45.</p> <p>X'01' DKYL1. Generate a level 0 diversified key with key type DKYGENKY.</p> <p>X'02' DKYL2. Generate a level 1 diversified key with key type DKYGENKY.</p> <p>All other values are reserved and undefined.</p>
49 (if defined)	2	<p>Key-usage field 3 (related generated key usage fields):</p> <p>These values determine allowable key usage of key to be generated.</p> <p>Meaning depends on value of offset 45:</p> <p>X'01' Same as key-usage field 1 of AES CIPHER key.</p> <p>X'02' Same as key-usage field 1 of AES MAC key.</p> <p>X'03' Same as key-usage field 1 of AES EXPORTER key.</p> <p>X'04' Same as key-usage field 1 of AES IMPORTER key.</p> <p>X'05' Same as key-usage field 1 of AES PINPROT key.</p> <p>X'06' Same as key-usage field 1 of AES PINCALC key.</p> <p>X'07' Same as key-usage field 1 of AES PINPRW key.</p> <p>X'08' Same as key-usage field 1 of AES SECMSG key.</p>

Table 73. AES algorithm DKYGENKY key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
51 (if defined)	2	<p>Key-usage field 4 (related generated key usage fields):</p> <p>These values determine allowable key usage of key to be generated.</p> <p>Meaning depends on value of offset 45:</p> <p>X'01' Same as key-usage field 2 of AES CIPHER key.</p> <p>X'02' Same as key-usage field 2 of AES MAC key.</p> <p>X'03' Same as key-usage field 2 of AES EXPORTER key.</p> <p>X'04' Same as key-usage field 2 of AES IMPORTER key.</p> <p>X'05' Same as key-usage field 2 of AES PINPROT key.</p> <p>X'06' Same as key-usage field 2 of AES PINCALC key.</p> <p>X'07' Same as key-usage field 2 of AES PINPRW key.</p> <p>X'08' Same as key-usage field 2 of AES SECMSG key.</p>
53 (if defined)	2	<p>Key-usage field 5 (related generated key usage fields):</p> <p>These values determine allowable key usage of key to be generated.</p> <p>Meaning depends on value of offset 45:</p> <p>X'02' Same as key-usage field 3 of AES MAC key.</p> <p>X'03' Same as key-usage field 3 of AES EXPORTER key.</p> <p>X'04' Same as key-usage field 3 of AES IMPORTER key.</p> <p>X'05' Same as key-usage field 3 of AES PINPROT key.</p> <p>X'06' Same as key-usage field 3 of AES PINCALC key.</p> <p>X'07' Same as key-usage field 3 of AES PINPRW key.</p>
55 (if defined)	2	<p>Key-usage field 6 (related generated key usage fields):</p> <p>These values determine allowable key usage of key to be generated.</p> <p>Meaning depends on value of offset 45:</p> <p>X'03' Same as key-usage field 4 of AES EXPORTER key.</p> <p>X'04' Same as key-usage field 4 of AES IMPORTER key.</p>

Table 74. AES algorithm SECMSG 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: Secure message encryption enablement:</p> <p>Value</p> <p>Meaning</p> <p>X'00' Enable the encryption of PINs in an EMV secure message (SMPIN). All other values are reserved and undefined.</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 <i>uuu</i> are UDX-defined bits. All unused bits are reserved and must be zero.</p>
47	2	<p>Key-usage field 2: Indicates the key usage.</p> <p>High-order byte: Service restriction:</p> <p>Value</p> <p>Meaning</p> <p>X'00' Any verb can use this key (ANY-USE).</p> <p>X'01' Only CSNBDPC can use this key (DPC-ONLY). All other values are reserved and undefined.</p> <p>Low-order byte (reserved). All unused bits are reserved and must be zero</p>

Table 75. AES algorithm KEK key-usage fields		
Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (<i>kuf</i>): 4

Table 75. 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 75. 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>1xxx xxxx 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>xxxx xxx1 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>1xxx xxxx Key can wrap DES keys</p> <p>x1xx xxxx Key can wrap AES keys</p> <p>xx1x xxxx Key can wrap HMAC keys</p> <p>xxx1 xxxx Key can wrap RSA keys</p> <p>xxxx 1xxx 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>

Table 75. AES algorithm KEK key-usage fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
51	2	<p>Key-usage field 4</p> <p>High-order byte:</p> <p>1xxx xxxx Key can wrap DATA class keys</p> <p>x1xx xxxx Key can wrap KEK class keys</p> <p>xx1x xxxx Key can wrap PIN class keys</p> <p>xxx1 xxxx Key can wrap DERIVATION class keys</p> <p>xxxx 1xxx Key can wrap CARD class keys</p> <p>xxxx x1xx Key can wrap CVAR 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 76. AES algorithm CIPHER key associated data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (<i>kuf</i>): 2

Table 76. AES algorithm CIPHER key associated data (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	<p>Key-usage field 1</p> <p>High-order byte:</p> <p>1xxx xxxx Key can be used for encryption.</p> <p>x1xx xxxx Key can be used for decryption.</p> <p>xx1x xxxx Key can be used for data translate.</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>
47	2	<p>Key-usage field 2</p> <p>High-order byte:</p> <p>X'00' Key can be used for Cipher Block Chaining (CBC).</p> <p>X'01' Key can be used for Electronic Code Book (ECB).</p> <p>X'02' Key can be used for Cipher Feedback (CFB).</p> <p>X'03' Key can be used for Output Feedback (OFB).</p> <p>X'04' Key can be used for Galois/Counter Mode (GCM)</p> <p>X'05' Key can be used for XEX-based Tweaked CodeBook Mode with CipherText Stealing (XTS)</p> <p>X'FF' Key can be used for any mode of encryption</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>

Table 77. AES and HMAC algorithm key-management fields

Offset (Dec)	Length of Field (Bytes)	Description
48	2	<p>Key-management field 1</p> <p>High-order byte:</p> <p>1xxx xxxx Allow export using symmetric key.</p> <p>x1xx xxxx Allow export using unauthenticated asymmetric key.</p> <p>xx1x xxxx Allow export using authenticated asymmetric key.</p> <p>xxx1 xxxx Allow export in RAW format.</p> <p>All other bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>--symmetric--</p> <p>1xxx xxxx Prohibit export using DES key.</p> <p>x1xx xxxx Prohibit export using AES key.</p> <p>--asymmetric--</p> <p>xxxx 1xxx Prohibit export using RSA key.</p> <p>All other bits are reserved and must be zero.</p>

Table 77. AES and HMAC algorithm key-management fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
48 + kuf * 2	2	<p>Key-management field 2</p> <p>High-order byte:</p> <p>11xx xxxx Key, if present, is incomplete. Key requires at least 2 more parts.</p> <p>10xx xxxx Key, if present, is incomplete. Key requires at least 1 more part.</p> <p>01xx xxxx Key, if present, is incomplete. Key can be completed or have more parts added.</p> <p>00xx xxxx 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>xxx1 xxxx Key was encrypted with an untrusted KEK.</p> <p>xxxx 1xxx Key was in a format without type/usage attributes.</p> <p>xxxx x1xx Key was encrypted with key weaker than itself.</p> <p>xxxx xx1x Key was in a non-CCA format.</p> <p>xxxx xxx1 Key was encrypted in ECB mode.</p> <p>All other bits are reserved and must be zero.</p>

Table 77. AES and HMAC algorithm key-management fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
50 + kuf * 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>X'00' Unknown (Key Token Build2, Key Translate2)</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' 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 77. AES and HMAC algorithm key-management fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
50 + kuf * 2 (cont'd)	2 (cont'd)	<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>

Table 77. AES and HMAC algorithm key-management fields (continued)		
Offset (Dec)	Length of Field (Bytes)	Description
50 + kuf * 2 (cont'd)	2 (cont'd)	<p>Low-order byte: Pedigree Current</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>

Table 78. DESUSECV key-management fields		
Offset (Dec)	Length of Field (Bytes)	Description
47	1	Key-management field count (<i>kmf</i>): 1
48	2	<p>Key-management field 1</p> <p>High-order byte:</p> <p>B'0000 0000' Reserved</p> <p>All unused bits are reserved and must be zero.</p> <p>Low-order byte:</p> <p>B'0000 0000' Reserved</p> <p>All unused 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 79. Variable-length symmetric null token	
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 (zero).

PKA key token formats

As with DES key tokens, the first byte of a PKA key token indicates the type of token. If the first byte of the key identifier is X'1E' or X'1F', this indicates that it is a **PKA key 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.

A first byte of X'1F' indicates an internal token with a cleartext public key and a private key that is enciphered by the master key and ready for internal use.

PKA tokens are of variable length because they contain either RSA 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 optional sections, which depend on the token type.

A PKA key token can be a public or private key token, and a private key token can be internal or external. Therefore, there are three basic types of tokens, each of which can contain either RSA or ECC information:

- Public key tokens
- Private external key tokens
- Private internal key tokens

Public key tokens contain only the public key. Private key tokens contain the public and private key pair.

Internal PKA tokens

PKA private internal key tokens contain both private and public key information. There is no need for an internal token with only the public key information because the public values are in the clear.

The first byte of X'1F' indicates an internal token with a cleartext public key and a private key that is enciphered with a PKA master key and ready for local (internal) use.

The format of a PKA private internal key token is similar to that of a private external token. The only differences are changes in the private key section and the addition of some internal information at the end of the token. This last section starts with the eyecatcher 'PKTN' rather than with a token or section marker.

PKA null key token

[Table 80 on page 287](#) shows the format for a PKA null key token.

Table 80. 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

This topic describes the different 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 81 on page 288 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.

Table 81. RSA Public Key Token		
Offset (Dec)	Number of Bytes	Description
Token Header (required)		
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.
RSA Public Key Section (required)		
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 $2^{16}+1$)
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.
 - X'08' which indicates an optimized Chinese Remainder Theorem form private key section with modulus bit length of up to 4096.
 - X'09' which indicates a modulus-exponent form RSA private key section (not optimized) with modulus length of up to 4096 bits.
 - X'30' which indicates a modulus-exponent form RSA private key section with modulus length of up to 4096 bits with an AES object protection key.
 - X'31' which indicates an Chinese Remainder Theorem form private key section with modulus bit length of up to 4096 bits with an AES object protection key.
- 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 82 on page 289 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. 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 82. RSA Private External Key Token Basic Record Format		
Offset (Dec)	Number of Bytes	Description
		Token Header (required)
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.
		RSA Private Key Section (required) <ul style="list-style-type: none"> For 1024-bit Modulus-Exponent form refer to “RSA private key token, 1024-bit modulus-exponent external format” on page 290. For 4096-bit Modulus-Exponent form refer to “RSA private key token, 4096-bit modulus-exponent external format” on page 291. For 4096-bit Chinese Remainder Theorem form refer to “RSA private key token, 4096-bit chinese remainder Theorem external format” on page 292. For 4096-bit Modulus-Exponent form with AES OPK refer to “RSA private key, 4096-bit modulus-exponent format with AES encrypted OPK section (X'30') external form” on page 294. For 4096-bit Chinese Remainder Theorem form with AES OPK refer to “RSA private key, 4096-bit chinese remainder Theorem format with AES encrypted OPK section (X'31') external form” on page 297.
		RSA Public Key Section (required)
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. Note: 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).)
		Private Key Name (optional)
000	001	X'10', section identifier, private key name.
001	001	X'00', version.

Table 82. RSA Private External Key Token Basic Record Format (continued)		
Offset (Dec)	Number of Bytes	Description
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 format

Table 83. 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. Bit Meaning When Set On 0 Key management usage permitted. 1 Signature usage not permitted. 6 The key is translatable. All other bits reserved, set to binary zero.
054	006	Reserved; set to binary zero.
060	024	Reserved; set to binary zero.
		Start of the optionally-encrypted secure subsection.
084	024	Random number, confounder.

Table 83. 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 format

This RSA private key token and the external X'09' token is supported on a CCA Crypto Express coprocessor.

Table 84. 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'.

Table 84. RSA Private Key Token, 4096-bit Modulus-Exponent external format (continued)		
Offset (Dec)	Number of Bytes	Description
050	001	Key use flag bits. Bit Meaning When Set On 0 Key management usage permitted. 1 Signature usage not permitted. 6 The key is translatable All other bits reserved, set to binary zero.
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.
		Start of the optionally-encrypted secure subsection.
124	008	Random number, confounder.
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 token, 4096-bit chinese remainder Theorem external format

This RSA private key token with up to 2048-bit modulus is supported on all coprocessors. The modulus size is increased to 4096-bit on the z9 EC, z9 BC, z10 EC, z10 BC, or later machines with the Nov. 2007 or later version of the licensed internal code installed on the CCA Crypto Express coprocessor.

Table 85. 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)

Table 85. RSA Private Key Token, 4096-bit Chinese Remainder Theorem external format (continued)		
Offset (Dec)	Number of Bytes	Description
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.
024	004	Reserved; set to binary zero.
028	001	Key format and security: X'40' Unencrypted RSA private-key subsection identifier, Chinese Remainder form. X'42' 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. Bit Meaning When Set On 0 Key management usage permitted. 1 Signature usage not permitted. 6 The key is translatable. 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: qqg.
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	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.
		Start of the optionally-encrypted secure subsection.

Table 85. RSA Private Key Token, 4096-bit Chinese Remainder Theorem external format (continued)		
Offset (Dec)	Number of Bytes	Description
124	008	Random number, confounder.
132	ppp	Prime number, p.
132 + ppp	qqq	Prime number, q
132 + ppp + qqq	rrr	$d_p = d \bmod(p - 1)$
132 + ppp + qqq + rrr	sss	$d_q = d \bmod(q - 1)$
132 + ppp + qqq + rrr + sss	uuu	$U = q^{-1} \bmod(p)$.
132 + ppp + qqq + 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.
132 + ppp + qqq + rrr + sss + uuu + 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 (X'30') external form

This RSA private key token is supported on the Crypto Express3 Coprocessor and Crypto Express4 Coprocessor.

Table 86. 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.
		Start of Associated Data

Table 86. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external form (continued)

Offset (bytes)	Length (bytes)	Description
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.
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) All other values are undefined. Translation control bit: B'xxxx xx1x' Private key translation is allowed (XLATE-OK) B'xxxx xx0x' Private key translation is not allowed (NO-XLATE) All other bits are reserved and must be zero.

Table 86. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section (X'30') external form (continued)

Offset (bytes)	Length (bytes)	Description
051	001	<p>Format restriction byte for digital-signature hash formatting method.</p> <p>Value:</p> <p>B'0000 0000' No format restriction.</p> <p>B'0000 0001' ISO-9796 only.</p> <p>B'0000 0010' PKCS-1.0 only.</p> <p>B'0000 0011' PKCS-1.1 only.</p> <p>B'0000 0100' PKCS-PSS only.</p> <p>B'0000 0101' X9.31 only.</p> <p>B'0000 0110' ZERO-PAD only.</p> <p>All other values are reserved and undefined.</p>
052	002	Length of modulus: nnn bytes
054	002	Length of private exponent: ddd bytes
		End of Associated Data
056	048	<p>16 byte confounder + 32-byte Object Protection Key.</p> <p>OPK used as an AES key.</p> <p>encrypted with an AES KEK.</p>
104	016	<p>Key verification pattern</p> <ul style="list-style-type: none"> • For an encrypted private key, KEK verification pattern (KVP) • For a clear private key, binary zeros • For a skeleton, binary zeros
120	002	Reserved, binary zeros.
122	nnn	Modulus
122+nnn	ppp	<p>Payload starts here and includes:</p> <p>When this section is unencrypted:</p> <ul style="list-style-type: none"> • Clear private exponent d. • Length ppp bytes : ddd + 0 <p>When this section is encrypted:</p> <ul style="list-style-type: none"> • Private exponent d within the AESKW-wrapped payload. • Length ppp bytes : ddd + AESKW format overhead

RSA private key, 4096-bit chinese remainder Theorem format with AES encrypted OPK section (X'31') external form

This RSA private key token is supported on the Crypto Express3 Coprocessor and Crypto Express4 Coprocessor.

Table 87. 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: 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.
		Start of Associated Data
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'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 87. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form (continued)

Offset (bytes)	Length (bytes)	Description
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: uuu.
062	002	Length of modulus, nnn.
064	002	Reserved, binary zero.
066	002	Reserved, binary zero.
		End of Associated Data
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 ECC master key.
116	016	Key verification pattern <ul style="list-style-type: none"> • For an encrypted private key, KEK verification pattern (KVP) • For a clear private key, binary zeros • For a skeleton, binary zeros
132	002	Reserved, binary zeros
134	nnn	Modulus, n, n=pq, where p and q are prime.

Table 87. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external 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"> • Clear prime number p • Clear prime number q • Clear dp • Clear dq • Clear U • Length xxx bytes: ppp + qqg + rrr + sss +uuu + 0 <p>When this section is encrypted:</p> <ul style="list-style-type: none"> • prime number p • prime number q • dp • dq • U • within the AESKW-wrapped payload. <p>Length xxx bytes : ppp + qqg + rrr + sss +uuu + AESKW format overhead</p>

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. 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 88. RSA Private Internal Key Token Basic Record Format

Offset (Dec)	Number of Bytes	Description
Token Header (required)		
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.

Table 88. RSA Private Internal Key Token Basic Record Format (continued)

Offset (Dec)	Number of Bytes	Description
RSA Private Key Section and Secured Subsection (required)		
<ul style="list-style-type: none"> For 1024-bit X'02' Modulus-Exponent form, refer to “RSA private key token, 1024-bit X'02' modulus-exponent internal form” on page 301. For 1024-bit X'06' Modulus-Exponent form, refer to “RSA private key token, 1024-bit X'06' modulus-exponent internal form” on page 302. For 4096-bit X'08' Chinese Remainder Theorem form, refer to “RSA private key token, 4096-bit chinese remainder Theorem internal form” on page 308. For 4096-bit Modulus-Exponent form with AES OPK, refer to “RSA private key, 4096-bit modulus-exponent format with AES encrypted OPK section internal form” on page 304. For 4096-bit Chinese Remainder Theorem form with AES OPK, refer to Table 92 on page 306. 		
RSA Public Key Section (required)		
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).
Private Key Name (optional)		
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.
Internal Information Section (required)		
000	004	Eye catcher 'PKTN'.

Table 88. RSA Private Internal Key Token Basic Record Format (continued)		
Offset (Dec)	Number of Bytes	Description
004	004	PKA token type. 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 X'02' modulus-exponent internal form

Table 89. RSA Private Internal Key Token, 1024-bit X'02' ME Form		
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.

Table 89. RSA Private Internal Key Token, 1024-bit X'02' ME Form (continued)		
Offset (Dec)	Number of Bytes	Description
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.
050	001	Key use flag bits. Bit Meaning When Set On 0 Key management usage permitted. 1 Signature usage not permitted. All other bits reserved, set to binary zero.
051	009	Reserved; set to binary zero.
060	048	Object Protection Key (OPK) encrypted under the RSA-MK.
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 X'06' modulus-exponent internal form

Table 90. RSA Private Internal Key Token, 1024-bit X'06' ME Form		
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.

Table 90. RSA Private Internal Key Token, 1024-bit X'06' ME Form (continued)		
Offset (Dec)	Number of Bytes	Description
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. Bit Meaning When Set On 0 Key management usage permitted. 1 Signature usage not permitted. All other bits reserved, set to binary zeros.
054	006	Reserved; set to binary zero.
060	048	Object Protection Key (OPK) encrypted under the RSA-MK using the ede3 algorithm.
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	RSA master key verification 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.

Table 90. RSA Private Internal Key Token, 1024-bit X'06' ME Form (continued)		
Offset (Dec)	Number of Bytes	Description
		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 and newer Coprocessor.

Table 91. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section 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.
		Start of Associated Data
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
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

Table 91. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section internal form (continued)

Offset (bytes)	Length (bytes)	Description
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	<p>Key-usage flag:</p> <p>B'11xx xxxx' Only key unwrapping (KM-ONLY)</p> <p>B'10xx xxxx' Both signature generation and key unwrapping (KEY-MGMT)</p> <p>B'01xx xxxx' Undefined</p> <p>B'00xx xxxx' Only signature generation (SIG-ONLY)</p> <p>All other values are undefined.</p> <p>Translation control bit:</p> <p>B'xxxx xx1x' Private key translation is allowed (XLATE-OK)</p> <p>B'xxxx xx0x' Private key translation is not allowed (NO-XLATE)</p> <p>All other bits are reserved and must be zero.</p>
051	001	<p>Format restriction byte for digital-signature hash formatting method.</p> <p>Value:</p> <p>B'0000 0000' No format restriction.</p> <p>B'0000 0001' ISO-9796 only.</p> <p>B'0000 0010' PKCS-1.0 only.</p> <p>B'0000 0011' PKCS-1.1 only.</p> <p>B'0000 0100' PKCS-PSS only.</p> <p>B'0000 0101' X9.31 only.</p> <p>B'0000 0110' ZERO-PAD only.</p> <p>All other values are reserved and undefined.</p>
052	002	Length of modulus: nnn bytes
054	002	Length of private exponent: ddd bytes
		End of Associated Data
056	048	<p>16 byte confounder + 32-byte Object Protection Key.</p> <p>OPK used as an AES key.</p> <p>encrypted with the ECC master key.</p>

Table 91. RSA private key, 4096-bit Modulus-Exponent format with AES encrypted OPK section internal form (continued)		
Offset (bytes)	Length (bytes)	Description
104	016	Key verification pattern <ul style="list-style-type: none"> For an encrypted private key, ECC master-key verification pattern (MKVP) For a skeleton, binary zeros
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"> Clear private exponent d. Length ppp bytes : ddd + 0 When this section is encrypted: <ul style="list-style-type: none"> Private exponent d within the AESKW-wrapped payload. Length ppp bytes : ddd + AESKW format overhead

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 and newer Coprocessor.

RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') external form

Table 92. 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.
		Start of Associated Data
010	001	Associated Data Version: X'03' Version 3
011	001	Key format and security flag: X'08' Unencrypted RSA private-key subsection identifier

Table 92. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form (continued)

Offset (bytes)	Length (bytes)	Description
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.
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: uuu.
062	002	Length of modulus, nnn.
064	002	Reserved, binary zero.
066	002	Reserved, binary zero.
		End of Associated Data

Table 92. RSA private key, 4096-bit Chinese Remainder Theorem format with AES encrypted OPK section (X'31') internal form (continued)

Offset (bytes)	Length (bytes)	Description
068	048	16 byte confounder + 32-byte Object Protection Key. OPK used as an AES key. encrypted with the ECC-MK.
116	016	Key verification pattern <ul style="list-style-type: none"> For an encrypted private key, ECC master-key verification pattern (MKVP) For a skeleton, binary zeros
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"> Clear prime number p Clear prime number q Clear dp Clear dq Clear U Length xxx bytes: ppp + qqg + rrr + sss +uuu + 0 When this section is encrypted: <ul style="list-style-type: none"> prime number p prime number q dp dq U within the AESKW-wrapped payload. Length xxx bytes : ppp + qqg + rrr + sss +uuu + AESKW format overhead

RSA private key token, 4096-bit chinese remainder Theorem internal form

This RSA private key token (up to 2048-bit modulus) is supported on all cryptographic coprocessors. The 4096-bit modulus private key token is supported on the z9 EC, z9 BC, z10 EC, z10 BC, or IBM zEnterprise 196 with the Nov. 2007 or later version of the licensed internal code installed on the CCA Crypto Express coprocessor.

Table 93. 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.

Table 93. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format (continued)		
Offset (Dec)	Number of Bytes	Description
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'.
050	004	Key use flag bits: Bit Meaning When Set On 0 Key management usage permitted. 1 Signature usage not permitted. 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: qqg.
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	RSA master key verification pattern.
092	032	Object Protection Key (OPK) encrypted under the Asymmetric-Keys Master Key using the TDES (CBC outer chaining) algorithm.

Table 93. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format (continued)		
Offset (Dec)	Number of Bytes	Description
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 + qqq	rrr	$d_p = d \bmod(p - 1)$
132 + ppp + qqq + rrr	sss	$d_q = d \bmod(q - 1)$
132 + ppp + qqq + rrr + sss	uuu	$U = q^{-1} \bmod(p)$.
132 + ppp + qqq + rrr + sss + uuu	ttt	Random number r (used in blinding).
132 + ppp + qqq + rrr + sss + uuu + ttt	iii	Random number r^{-1} (used in blinding).
132 + ppp + qqq + 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 + qqq + rrr + sss + uuu + ttt + iii + xxx	nnn	Modulus, n. $n = pq$ where p and q are prime and $1 < n < 2^{4096}$.

ECC key token format

The following table presents the format of the ECC Key Token.

Table 94. ECC Key Token Format		
Offset (Dec)	Number of bytes	Description
Token header		
000	001	Token identifier. X'00' Null token X'1E' External token X'1F' Internal token; the private key is protected by the master key
001	001	Version, X'00'.

Table 94. ECC Key Token Format (continued)		
Offset (Dec)	Number of bytes	Description
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	<p>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).</p> <p>X'00' Clear – section is unencrypted.</p> <p>X'01' AESKW</p> <p>X'02' CBC Wrap - Other</p>
005	001	<p>Hash used for Wrapping</p> <p>X'01' SHA224</p> <p>X'02' SHA256</p> <p>X'04' Reserved.</p> <p>X'08 ' Reserved</p>
006	002	Reserved Binary Zero
008	001	<p>Key Usage:</p> <p>X'C0' Key Agreement</p> <p>X'80' Both signature generation and key agreement</p> <p>X'00' Signature generation only</p> <p>X'02' Translate allowed</p> <p>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.</p>

Table 94. ECC Key Token Format (continued)		
Offset (Dec)	Number of bytes	Description
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)
014	002	IBM Associated Data length. The length of this field must be greater than or equal to 16

Table 94. ECC Key Token Format (continued)		
Offset (Dec)	Number of bytes	Description
016	008	External Token: <ul style="list-style-type: none"> • Unencrypted – Reserved Binary 0x'00' • Encrypted – KVP of the AESKEK 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"> • 8 bytes = A6A6A6A6A6A60000 • 40 bytes = Confounder(8)/Key(32)
072	002	Associated data length, aa
074	002	Length of formatted section in bytes, bb
076	aa	Associated data See “Associated data format for ECC token” on page 314.
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 “AESKW wrapped payload format for ECC private key token” on page 315.
76 + aa + bb	End of formatted section	
ECC Token Public Section		
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 94. 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

Table 95 on page 314 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 95. 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.

<i>Table 95. Associated Data Format for ECC Private Key Token (continued)</i>		
Offset (Dec)	Number of Bytes	Description
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.

<i>Table 96. AESKW Wrapped Payload Format for ECC Private Key Token</i>		
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 blocks

A trusted block is an extension of CCA PKA key tokens using new section identifiers. 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 callable service. This service 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 service 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 service.

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:

Table 97. Trusted block sections		
Section	Reference	Usage
Header	Table 98 on page 317	Trusted block token header
X'11'	Table 99 on page 318	Trusted block public key
X'12'	Table 100 on page 319	Trusted block rule
X'13'	Table 107 on page 326	Trusted block name (key label)
X'14'	Table 108 on page 326	Trusted block information
X'15'	Table 112 on page 328	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; 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 can be 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 whether 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 98. 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” on page 317.

Following the header, in no particular order, are trusted block sections. There are five different sections that are defined, each identified by a one-byte section identifier (X'11' - X'15'). Two of the five sections

have subsections that are defined. A subsection is a tag-length-value (TLV) object, which is identified by a two-byte subsection tag.

Only sections X'12' and X'14' have subsections that are 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 that are 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 that are defined.

This section is defined in the following table:

Table 99. 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). Note: 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 317.

Trusted block section X'12'

Trusted block section X'12' contains information that defines a rule. A trusted block can 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 that is allowed to have multiple sections. Section X'12' is optional. Multiple sections are allowed.

Note: The overall length of the trusted block can not exceed its maximum size of 3500 bytes.

Five subsections (TLV objects) are defined.

This section is defined in the following table:

Table 100. Trusted block rule section (X'12')		
Offset (bytes)	Length (bytes)	Description
Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'12' 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). X'00000000' Generate new key. X'00000001' 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.

Table 100. Trusted block rule section (X'12') (continued)

Offset (bytes)	Length (bytes)	Description
017	001	<p>Key-check algorithm identifier (all others are reserved and must not be used):</p> <p>Value Meaning</p> <p>X'00' Do not compute key-check value. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to zero.</p> <p>X'01' 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.</p> <p>X'02' Compute the MDC-2 hash of the key. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to 16.</p>
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 by using the <i>sym_encrypted_key_identifier</i> parameter.</p> <p>Value Meaning</p> <p>X'00' Return an RKX key-token encrypted under a variant of the MAC key.</p> <p>Note: This is the only key format that is permitted when the flags value (offset 012) is set to generate a new key.</p> <p>X'01' Return a CCA DES key-token encrypted under a transport key.</p> <p>Note: This is the only key format that is permitted when the flags value (offset 012) is set to export an existing key.</p>
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> <p>Value Meaning</p> <p>X'00' Do not return an asymmetric key. Set the <i>asym_encrypted_key_length</i> variable to zero.</p> <p>X'01' Output in PKCS1.2 format.</p> <p>X'02' Output in RSAOAEP format.</p>
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 317.

Section X'12' has five rule subsections (tag-length-value objects) defined. These subsections are summarized in the following table:

Table 101. 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.

Table 101. Summary of trusted block rule subsection (continued)			
Rule subsection tag	TLV object	Optional or required	Comments
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. Note: 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.
X'0005'	Export key CCA token parameters	Optional; used for export of CCA DES key tokens only	Contains mask length, mask, and CV template to limit the usage of the exported key. Also contains the template length and template that defines which source key labels are allowed. 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.

Note: See “Number representation in trusted blocks” on page 317.

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 102. 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	Length of variant field in bytes (nnn). 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 before use.
008	nnn	Transport key variant. 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'. Note: A transport key is not used when the symmetric encrypted output key is in RKX key-token format.

Note: See “Number representation in trusted blocks” on page 317.

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 103. Transport key rule reference subsection (X'0002') of trusted block rule section (X'12')		
Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: X'0002' 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 wanted. 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 104. Common export key parameters subsection (X'0003') of trusted block rule section (X'12')		
Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: X'0003' 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 that uses this rule. If the variant is longer than the key, truncate it on the right to the length of the key before use. Note: The output key variant (offset 011) is not used if this length is zero.
011	xxx	Output key variant. The variant can be any value. Exclusive-OR this variant into the cleartext value of the output.

Table 104. Common export key parameters subsection (X'0003') of trusted block rule section (X'12') (continued)		
Offset (bytes)	Length (bytes)	Description
011+xxx	001	<p>CV length in bytes (yyy).</p> <ul style="list-style-type: none"> If the length is not 0, 8, or 16, return an error. 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. If a non-zero length is less than the length of the key that is identified by the <i>source_key_identifier</i> parameter, return an error. 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"> Ignore all CV bit definitions. If CCA DES key-token format, set the flag byte of the symmetric encrypted output key to indicate that a CV value is present. If the source key is 8 bytes, do not replicate the key to 16 bytes.
012+xxx	yyy	<p>CV.</p> <p>Place this CV into the output exported key-token, if the symmetric encrypted output key format selected (offset 018 in rule section) is CCA DES key-token.</p> <ul style="list-style-type: none"> 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. 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 that is identified by the <i>source_key_identifier</i> parameter whenever all of these conditions are met: <ol style="list-style-type: none"> The Replicate Key command (offset X'00DB') is enabled in the active role. The CV length (offset 011+xxx) is 16, and both CV halves are non-zero. The <i>source_key_identifier</i> parameter (contained in either a CCA DES key-token or RKX key-token) identifies an 8-byte key. The key-form bits (40 - 42) of this CV do not indicate a single-length key (are not set to zero) Key-form bit 40 of this CV does not indicate that the key is to have guaranteed unique halves (is not set to 1). <p>Note: 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 317.

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 that is 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 105. Source key rule reference subsection (X'0004' of trusted block rule section (X'12'))		
Offset (bytes)	Length (bytes)	Description
000	002	<p>Subsection tag:</p> <p>X'0004'</p> <p>Source key rule reference TLV object.</p>
002	002	Subsection length in bytes (14).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.

Table 105. Source key rule reference subsection (X'0004' of trusted block rule section (X'12') (continued)		
Offset (bytes)	Length (bytes)	Description
006	008	<p>Rule ID.</p> <p>Rule identifier for the rule that must have been used to create the source key.</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>

Note: See “Number representation in trusted blocks” on page 317.

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 106. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12')		
Offset (bytes)	Length (bytes)	Description
000	002	<p>Subsection tag:</p> <p>X'0005'</p> <p>Export key CCA token parameters TLV object.</p>
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	<p>Export key CV limit mask length in bytes (yyy).</p> <p>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:</p> <ul style="list-style-type: none"> • Must be 8 or 16. • Must not be less than the export key minimum length (offset 008 in subsection X'0003'). • Must be equal in length to the actual source key length of the key. <p>Example: An export key minimum length of 16 and an export key CV limit mask length of 8 returns an error.</p>
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>Examples: 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>

Table 106. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12') (continued)		
Offset (bytes)	Length (bytes)	Description
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"> 1. If the length of the key to be exported is less than yyy, return an error. 2. Logical AND the CV for the key to be exported with the export key CV limit mask. 3. Compare the result to the export key CV limit template. 4. Return an error if the comparison is not equal. <p>Examples: 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>Note: 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"> • The key label template must be 64 bytes. • The first character cannot be in the range X'00' - X'1F', nor can it be X'FF'. • The first character cannot be numeric (X'30' - X'39'). • 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. • The only special characters that are permitted are #, \$, @, and * (X'23', X'24', X'40', and X'2A'). • The wildcard X'2A' (*) is only permitted as the first character, the last character, or the only character in the template. • 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.

Note: See “Number representation in trusted blocks” on page 317.

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 that are defined. This section is defined in the following table:

Table 107. Trusted block key label (name) section X'13'		
Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'13' 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 317.

Trusted block section X'14'

Trusted block section X'14' contains control and security information that is 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 108. Trusted block information section X'14'		
Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'14' 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: X'00000000' Trusted block is in the inactive state. X'00000001' 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 317.

Section X'14' has two information subsections (tag-length-value objects) defined. These subsections are summarized in the following table:

Table 109. 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 by using MDC4).

Table 109. Summary of trusted block information subsections (continued)			
Rule subsection tag	TLV object	Optional or required	Comments
X'0002'	Activation and expiration dates	Optional	Contains flags indicating whether 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 317.

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 by using MDC4).

This subsection is defined in the following table:

Table 110. Protection information subsection (X'0001') of trusted block information section (X'14')		
Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: X'0001' 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: <div> Offset Description 00 - 07 Confounder. 08 - 15 Left key. 16 - 23 Middle key. 24 - 31 Right key. </div>
038	008	MAC. Contains the ISO-16609 TDES CBC message authentication code value.
046	016	MKVP. Contains the PKA master key verification pattern, computed by using MDC4, when the trusted block is in internal form, otherwise contains binary zero.

Note: See “Number representation in trusted blocks” on page 317.

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 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 111. Activation and expiration dates subsection (X'0002') of trusted block information section (X'14')		
Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: X'0002' 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: X'0000' The coprocessor does not check dates. X'0001' 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: YY Big-endian year (return an error if greater than 9999). M Month (return an error if any value other than X'01' - X'0C'). D 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 317.

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 not examined or used by CCA in any way.

Section X'15' is optional. No multiple sections are allowed. It has no subsections that are defined. This section is defined in the following table:

Table 112. Trusted block application-defined data section X'15'		
Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: X'15' Application-defined data.
001	001	Section version number (X'00').
002	002	Section length (6+xxx).

Table 112. Trusted block application-defined data section X'15' (continued)		
Offset (bytes)	Length (bytes)	Description
004	002	Application data length (xxx). The value of xxx can be from 0 bytes to a length that does not cause the trusted block to exceed its maximum size of 3500 bytes.
006	xxx	Application-defined data. Can be used to hold a public-key certificate for the trusted public key.

Note: See “Number representation in trusted blocks” on page 317.

Data areas

These topics present the format of the Cryptographic Communication Vector Table (CCVT) and the Cryptographic Communication Vector Table Extension (CCVE) data areas.

The Cryptographic Communication Vector Table (CCVT)

The CCVT is the ICSF base control block and contains addresses of common areas for use by ICSF components. Indicators in the CCVT also provide ICSF status information. The CCVT is getmained in subpool 245 under the line. The ICSF CCVT is anchored off of SCVTCCVT in the SCVT macro.

ONLY these fields are part of the programming interface:

- CCVTDACC
- CCVTRLVL
- CCVTCCVE
- CCVTHFLG
- CCVTSFLG
- CCVTPRPC
- CCVTINST
- CCVTINS2
- CCVTLNTH
- CCVTFMID
- CCVT_USERPARM

Table 113 on page 329 describes the contents of the Cryptographic Communication Vector Table. Any bits that are not described in the table are reserved.

Table 113. Cryptographic communication vector table			
Offset (Dec)	Number of bytes	Field name	Description
12	4	CCVTRLVL	ICSF level.

Table 113. Cryptographic communication vector table (continued)

Offset (Dec)	Number of bytes	Field name	Description
16	4	CCVTCCVE	Cryptographic Communication Vector Table Extension (CCVE) address. The address of a private area extension of the CCVT. You should place any fields not needed by other address spaces in the CCVE.
28	4	CCVTPRPC	Entry point for the pre-PC processing module, CSFARPC.
32	4	CCVTINST	For installation use.
56	8	CCVTINS2	An 8-byte area for installation use.
68	4	CCVTLNTH	Maximum installation data length.
80	1	CCVTHFLG	Flag bytes. Bit Meaning When Set On 0 Crypto assist instructions available. 1 Additional secure Crypto device available. 2 Support for 64-bit callers. 3 ICSF Cross-System Services environment is active for CKDS. 4 ICSF Cross-System Services environment is active for TKDS. 5 RSA 4096-bit function enabled and the RNGL service is available. 6 Secure key AES is available. 7 AES master key is active.
81	1	CCVTSFLG	Flag bytes. Bit Meaning When Set On 0 ICSF during initialization. 1 ICSF was able to complete cleanup, so no EOM cleanup is needed. 2 PKCS #11 operating in FIPS standard mode. 3 PKCS #11 operating in FIPS compatibility mode.
136	8	CCVTFMID	ICSF FMID.
144	8	CCVT_USERPARM	ICSF user parameter.
276	4	CCVTDACC	ICSF DAC instructions control block for RMF.

Table 113. Cryptographic communication vector table (continued)			
Offset (Dec)	Number of bytes	Field name	Description
484	44	CCVT_PKDSN	Name of the active PKDS. If no PKDS was specified, the first character will be an EBCDIC blank (X'40').
704	44	CCVT_CKDSN	Name of the active CKDS. If no CKDS was specified, the first character will be an EBCDIC blank (X'40').
748	44	CCVT_TKDSN	Name of the active TKDS.

The Cryptographic Communication Vector Table Extension (CCVE)

The CCVE is an extension of the CCVT that contains fields that can exist. The CCVE exists in ICSF extended private. It should contain any ICSF base control block fields that are not needed by other address spaces.

ONLY these fields are part of the programming interface:

- CCVEINPP
- CCVEINPL
- CCVESECC

Table 114 on page 331 describes the contents of the Cryptographic Communication Vector Table Extension. Any bits that are not described in the table are reserved.

Table 114. Cryptographic Communication Vector Table Extension			
Offset (Dec)	Number of Bytes	Field Name	Description
328	4	CCVEINPP	Pointer to installation optional parameter.
332	4	CCVEINPL	Length of the installation optional parameter.
372	8	CCVESECC	Reserved for security exit.

Generic Service Table (CSFMGST)

Table 115 on page 331 describes the format of the generic service table, a control block that is used to control the call of installation-defined services.

Table 115. Generic Service Table Block Format		
Offset (Dec)	Number of Bytes	Description
0	4	EBCDIC ID.
4	2	Version number.
6	2	Length of the MGST.
8	4	Number of entries in the array.
12	4	Subpool this table is in.
16	4	Reserved.

Table 115. Generic Service Table Block Format (continued)		
Offset (Dec)	Number of Bytes	Description
20	4	Reserved.
24	4	Reserved.
28	4	Reserved.
Variable Section of the MGST (Repeat for each entry in the array)		
0	8	IBM-assigned name.
8	8	Installation-assigned name.
16	4	Flags. Bit Meaning When Set On 0 Service has been requested by the installation. 1 Service has been loaded. 2 Service is active. 3 Service is required. 4 Service is UDX.
20	4	Address of the service.
24	4	Installation-assigned service number.
28	4	Reserved.

RMF measurements table

Table 116 on page 332 describes the contents of the performance measurements for RMF. The count fields are double-word length.

Table 116. RMF measurements record format			
Offset (Dec)	Number of bytes	Field name	Description
0	4	DACC_ID	The DACC ID.
4	4	DACC_VER	The version.
8	4	DACC_LEN	The control block length.
12	2	DACC_ENT_CNT	Number of entries.
14	2	DACC_ENT_LEN	Length of each entry.

Table 116. RMF measurements record format (continued)

Offset (Dec)	Number of bytes	Field name	Description
16	8	DACC_ENT_ID	Identifier of count array - character 'ENCSDDES'. The Encipher service will collect data as follows: <ul style="list-style-type: none"> Collection for single DES is done separately. The number of service calls, number of bytes of data enciphered, and the number of hardware instructions used to encipher the data will be collected.
24	8	DACC_ENT_SVC_CNT	Count of ENCSDDES service calls.
32	8	DACC_ENT_BYT_CNT	Count of ENCSDDES bytes processed.
40	8	DACC_ENT_INT_CNT	Count of ENCSDDES instructions.
48	8	DACC_ENT_ID	Identifier of count array - character 'ENCTDES'. The Encipher service will collect data as follows: <ul style="list-style-type: none"> Double and triple DES will be counted together. The number of service calls, number of bytes of data enciphered, and the number of hardware instructions used to encipher the data will be collected.
56	8	DACC_ENT_SVC_CNT	Count of ENCTDES service calls.
64	8	DACC_ENT_BYT_CNT	Count of ENCTDES bytes processed.
72	8	DACC_ENT_INT_CNT	Count of ENCTDES instructions.
80	8	DACC_ENT_ID	Identifier of count array - character DECSDES. The Decipher service will collect data as follows: <ul style="list-style-type: none"> Collection for single DES is done separately. The number of service calls, number of bytes of data deciphered, and the number of hardware instructions used to decipher the data will be collected.
88	8	DACC_ENT_SVC_CNT	Count of DECSDES service calls.
96	8	DACC_ENT_BYT_CNT	Count of DECSDES bytes processed.
104	8	DACC_ENT_INT_CNT	Count of DECSDES instructions.
112	8	DACC_ENT_ID	Identifier of count array - character DECTDES. The Decipher service will collect data as follows: <ul style="list-style-type: none"> Double and triple DES will be counted together. The number of service calls, number of bytes of data deciphered, and the number of hardware instructions used to decipher the data will be collected.
120	8	DACC_ENT_SVC_CNT	Count of DECTDES service calls.
128	8	DACC_ENT_BYT_CNT	Count of DECTDES bytes processed.
136	8	DACC_ENT_INT_CNT	Count of DECTDES instructions.

Table 116. RMF measurements record format (continued)

Offset (Dec)	Number of bytes	Field name	Description
144	8	DACC_ENT_ID	Identifier of count array - character MACGEN. The MAC Generate service will collect data as follows: <ul style="list-style-type: none"> Single and various double key MAC will be gathered together. The number of service calls, number of bytes of data MAC'd, and the number of instructions will be collected.
152	8	DACC_ENT_SVC_CNT	Count of MACGEN service calls.
160	8	DACC_ENT_BYT_CNT	Count of MACGEN bytes processed.
168	8	DACC_ENT_INT_CNT	Count of MACGEN instructions.
176	8	DACC_ENT_ID	Identifier of count array - character MACVER. The MAC Verify service will collect data as follows: <ul style="list-style-type: none"> Single and various double key MAC will be gathered together. The number of service calls, number of bytes of data MAC'd, and the number of instructions will be collected.
184	8	DACC_ENT_SVC_CNT	Count of MACVER service calls.
192	8	DACC_ENT_BYT_CNT	Count of MACVER bytes processed.
200	8	DACC_ENT_INT_CNT	Count of MACVER instructions.
208	8	DACC_ENT_ID	Identifier of count array - character OWH. The One Way Hash service will collect data as follows: <ul style="list-style-type: none"> For SHA-1, the number of service calls, number of bytes of bytes of data hashed, and the number of instructions will be collected.
216	8	DACC_ENT_SVC_CNT	Count of OWH service calls.
224	8	DACC_ENT_BYT_CNT	Count of OWH bytes processed.
232	8	DACC_ENT_INT_CNT	Count of OWH instructions.
240	8	DACC_ENT_ID	Identifier of count array - character PTR. The Encrypted PIN Translate and Encrypted PIN Translate Enhanced services will collect data as follows: <ul style="list-style-type: none"> Collect the number of service calls only.
248	8	DACC_ENT_SVC_CNT	Count of PTR and PTRE service calls.
256	16		Reserved.
272	8	DACC_ENT_ID	Identifier of count array - character PVR. The PIN Verify service will collect data as follows: <ul style="list-style-type: none"> Collect the number of service calls only.
280	8	DACC_ENT_SVC_CNT	Count of PVR service calls.
288	16		Reserved.

Table 116. RMF measurements record format (continued)

Offset (Dec)	Number of bytes	Field name	Description
304	8	DACC_ENT_ID	Identifier of count array - character OWH256. The One Way Hash service will collect data as follows: <ul style="list-style-type: none"> For SHA-224 and SHA-256, the number of service calls, number of bytes of data hashed, and the number of instructions will be collected.
312	8	DACC_ENT_SVC_CNT	Count of OWH service calls for SHA-224 and SHA-256.
320	8	DACC_ENT_BYT_CNT	Count of OWH bytes processed for SHA-224 and SHA-256.
328	8	DACC_ENT_INT_CNT	Count of OWH instructions for SHA-224 and SHA-256.
336	8	DACC_ENT_ID	Identifier of count array - character OWH512. The One Way Hash service will collect data as follows: <ul style="list-style-type: none"> For SHA-384 and SHA-512, the number of service calls, number of bytes of data hashed, and the number of instructions will be collected.
344	8	DACC_ENT_SVC_CNT	Count of OWH service calls for SHA-384 and SHA-512.
352	8	DACC_ENT_BYT_CNT	Count of OWH bytes processed for SHA-384 and SHA-512.
360	8	DACC_ENT_INT_CNT	Count of OWH instructions for SHA-384 and SHA-512.
368	8	DACC_ENT_ID	Identifier of count array - character 'ENCAES'. The Symmetric algorithm encipher service will collect data as follows: The number of service calls, number of bytes of data enciphered, and the number of instructions used to encipher the data will be collected.
376	8	DACC_ENT_SVC_CNT	Count of SAE service calls
384	8	DACC_ENT_BYT_CNT	Count of ENCAES bytes processed
392	8	DACC_ENT_INT_CNT	Count of ENCAES instruction
400	8	DACC_ENT_ID	Identifier of count array - character 'DECAES'. The Symmetric algorithm decipher service will collect data as follows: the number of service calls, number of bytes of data deciphered, and the number of instructions used to decipher the data will be collected.
408	8	DACC_ENT_SVC_CNT	Count of SAD service calls
416	8	DACC_ENT_BYT_CNT	Count of DECAES bytes processed
424	8	DACC_ENT_INT_CNT	Count of DECAES instruction
432	8	DACC_ENT_ID	Identifier of count array - character 'DSGRSA'. The Digital Signature Generate service will collect the number of service calls processed to generate a digital signature using an RSA private key.
440	8	DACC_ENT_SVC_CNT	Count of DSG service calls using an RSA private key
448	16		Reserved

Table 116. RMF measurements record format (continued)

Offset (Dec)	Number of bytes	Field name	Description
464	8	DACC_ENT_ID	Identifier of count array - character 'DSGECC'. The Digital Signature Generate service will collect the number of service calls processed to generate a digital signature using an ECC private key.
472	8	DACC_ENT_SVC_CNT	Count of DSG service calls using an ECC private key
480	16		Reserved
496	8	DACC_ENT_ID	Identifier of count array - character 'DSVRSA'. The Digital Signature Verify service will collect the number of service calls processed to verify a digital signature using an RSA private key.
504	8	DACC_ENT_SVC_CNT	Count of DSV service calls using an RSA private key
512	16		Reserved
528	8	DACC_ENT_ID	Identifier of count array - character 'DSVECC'. The Digital Signature Verify service collects the number of service calls processed to verify a digital signature using an ECC private key.
536	8	DACC_ENT_SVC_CNT	Count of DSV service calls using an ECC private key
544	16		Reserved
560	8	DACC_ENT_ID	Identifier of count array - character 'MACGEN2'. The MAC Generate2 service collects data as follows: <ul style="list-style-type: none"> • The number of service calls. • The number of bytes of data MACed. • The number of instructions used to MAC the data.
568	8	DACC_ENT_SVC_CNT	Count of MACGEN2 service calls.
576	8	DACC_ENT_BYT_CNT	Count of MACGEN2 bytes processed.
584	8	DACC_ENT_INT_CNT	Count of MACGEN2 instructions.
592	8	DACC_ENT_ID	Identifier of count array - character 'MACVER2'. The MAC Verify2 service collects data as follows: <ul style="list-style-type: none"> • The number of service calls. • The number of bytes of data MACed. • The number of instructions used to MAC the data.
600	8	DACC_ENT_SVC_CNT	Count of MACVER2 service calls.
608	8	DACC_ENT_BYT_CNT	Count of MACVER2 bytes processed.
616	8	DACC_ENT_INT_CNT	Count of MACVER2 instructions.
624	8	DACC_ENT_ID	Identifier of count array - character 'FPPE'. The FPE encipher service collects data as follows: The number of service calls, number of bytes of data encrypted, and the number of instructions used to encipher the data.
632	8	DACC_ENT_SVC_CNT	Count of FPPE service calls.

Table 116. RMF measurements record format (continued)

Offset (Dec)	Number of bytes	Field name	Description
640	8	DACC_ENT_BYT_CNT	Count of FPED bytes processed.
648	8	DACC_ENT_INT_CNT	Count of FPED instructions.
656	8	DACC_ENT_ID	Identifier of count array - character 'FPED'. The FPE decipher service collects data as follows: The number of service calls, number of bytes of data decrypted, and the number of instructions used to decrypt the data.
664	8	DACC_ENT_SVC_CNT	Count of FPED service calls.
672	8	DACC_ENT_BYT_CNT	Count of FPED bytes processed.
680	8	DACC_ENT_INT_CNT	Count of FPED instructions.
688	8	DACC_ENT_ID	Identifier of count array - character 'FPET'. The FPE translate service collects data as follows: The number of service calls, number of bytes of data translated, and the number of instructions used to translate the data.
696	8	DACC_ENT_SVC_CNT	Count of FPET service calls.
704	8	DACC_ENT_BYT_CNT	Count of FPET bytes processed.
712	8	DACC_ENT_INT_CNT	Count of FPET instructions.

Appendix B. ICSF SMF records

This topic contains the ICSF SMF records.

Record type 82 (52) – ICSF record

Record type 82 is used to record information about the events and operations of the Integrated Cryptographic Service Facility (ICSF) program product. Record type 82 is written to the SMF data set at the completion of certain cryptographic functions:

- **Subtype 1** — is written whenever ICSF is started or the options refresh is performed.
- **Subtype 3** — no longer written.
- **Subtype 4** — no longer written.
- **Subtype 5** — no longer written.
- **Subtype 6** — no longer written.
- **Subtype 7** — is written when an operational key is imported from a coprocessor.
- **Subtype 8** — is written whenever the in-storage copy of the CKDS is refreshed.
- **Subtype 9** — is written whenever the CKDS is updated by a dynamic CKDS update service or the KDS Metadata write service.
- **Subtype 10** — no longer written.
- **Subtype 11** — no longer written.
- **Subtype 12** — no longer written.
- **Subtype 13** — is written whenever the PKDS is updated by a dynamic PKDS update service or the KDS Metadata write service.
- **Subtype 14** — is written when a clear master key part is entered on a cryptographic coprocessor.
- **Subtype 15** — is written whenever a retained key is created or deleted.
- **Subtype 16** — is written for each request and reply from calls to the CSFPCI service by TKE.
- **Subtype 17** — no longer written.
- **Subtype 18** — is written when a coprocessor or accelerator comes online or offline.
- **Subtype 19** — is written periodically to record processing times for PCIXCC coprocessors.
- **Subtype 20** — is written periodically to record processing times for coprocessors or accelerators.
- **Subtype 21** — is written when ICSF issues IXCJOIN to join the ICSF sysplex group or issues IXCLEAVE to leave the sysplex group.
- **Subtype 22** — is written when the Trusted Block Create Callable services are invoked.
- **Subtype 23** — is written when the token data set (TKDS) is updated
- **Subtype 24** — is written when duplicate tokens are found.
- **Subtype 25** — is written when key store policy checking detects the unauthorized use of a key token.
- **Subtype 26** — is written whenever the in-storage copy of the PKDS is refreshed.
- **Subtype 27** — is written when key store policy PKA key extensions checking detects the unauthorized use of a key.
- **Subtype 28** — is written for information about High Performance Encrypted Key.
- **Subtype 29** — is written for each TKE workstation audit record received from a TKE workstation.
- **Subtype 30** — is written for each time an archived or inactive key data set record is referenced.
- **Subtype 40** — is written for lifecycle events related to symmetric CCA tokens. This replaces subtype 9.

Record Type 82

- **Subtype 41** — is written for lifecycle events related to asymmetric CCA tokens. This replaces subtype 13.
- **Subtype 42** — is written for lifecycle events related to PKCS#11 objects. This replaces subtype 23.
- **Subtype 43** — is written when there is a configuration change for a regional cryptographic server.
- **Subtype 44** — is written for usage events related to symmetric CCA tokens.
- **Subtype 45** — is written for usage events related to asymmetric CCA tokens.
- **Subtype 46** — is written for usage events related to PKCS#11 objects.
- **Subtype 47** — is written for supported PKCS #11 usage events which do not involve an object.

Macro to Symbolically Address Record Type 82: The SMF record mapping macro for ICSF type 82 record is CSFSMF82.

The mapping macro, CSFSMF82, resides in SYS1.MODGEN.

Record environment

The following conditions exist for the generation of each of the subtypes of this record:

Macro

Subtype Macro

1

SMFWTM (record exit: IEFU83)

3,4,5,6,7,8

SMFEWTM,BRANCH=YES,MODE=XMEM (record exit: IEFU85)

Record mapping

Two different record formats are produced by ICSF; one format that applies to subtypes smaller than 40 and another format that applies to subtypes 40 and higher.

For subtypes smaller than 40, the SMF record header is followed by the subtype-specific information and optionally, the audit header and audit section.

Table 117. Format of an SMF Type 82 record for subtypes smaller than 40	
SMF format	
SMF record header	
Subtype information	
Audit section header (optional)	
Audit section (optional)	

For subtypes 40 and higher, the SMF record header is followed by an ICSF header, a main section that contains the subtype-specific information, and optionally, the audit header and audit section.

Table 118. Format of an SMF Type 82 record for subtypes 40 and higher	
SMF format	
SMF record header	
ICSF header	
Main section with subtype information	
Audit section header (optional)	
Audit section (optional)	

SMF header

Table 119. SMF record header					
Offsets	Name	Length	Format	Description	
0	0 SMF82LEN	2	binary	Record length. This field and the next field (total of four bytes) form the RDW (record descriptor word).	
2	2 SMF82SEG	2	binary	Segment descriptor (see record length field).	
4	4 SMF82FLG	1	binary	System indicator: Bit Meaning When Set 0-2 Reserved 3-6 Version indicators 7 Reserved.	
5	5 SMF82RTY	1	binary	Record type 82 (X'52').	
6	6 SMF82TME	4	binary	Time since midnight, in hundredths of a second, that the record was moved into the SMF buffer.	
10	A SMF82DTE	4	packed	Date when the record was moved into the SMF buffer, in the form <i>OcyydddF</i> .	
14	E SMF82SID	4	EBCDIC	System identification (from the SID parameter).	
18	12 SMF82SSI	4	EBCDIC	Subsystem identification.	
22	16 SMF82STY	2	binary	Record subtype.	

ICSF header (for all subtypes 40 or greater)

Table 120. ICSF header (for all subtypes 40 or greater)					
Offsets		Name	Length	Format	Description
Dec	Hex				
0	0	SMF82IHDR_VER	1	binary	Version number of this record (X'01'). Incremented if a change is made to the record that is incompatible with the prior version.
1	1		1		Reserved.
2	2	SMF82IHDR_LEN	2	binary	Length of this header.
4	4		4		Reserved.
8	8	SMF82IHDR_MAIN_OFF	2	binary	Offset from SMF82IHDR to main section.
10	A	SMF82IHDR_MAIN_LEN	2	binary	Length of main section.
12	C	SMF82IHDR_AUD_OFF	2	binary	Offset from SMF82IHDR to audit section. If there is no audit section, this field is zero.
14	E	SMF82IHDR_AUD_LEN	2	binary	Length of audit section.
16	10	SMF82IHDR_END	0		End of ICSF header.

Main section (subtype information)

The data contained in the main section is specific to each subtype. The content of each subtype is described in later sections of this document.

Audit header and audit section

Provides optional server user or end user audit information. When auditing information is supplied, there will be a server user section and optionally, an end user section. The SMF82AUD_HDR_NUM_SECTIONS field of the Auditing Header section indicates whether only a server user section is provided or if an end user section is also provided. If both a server user section and an end user section are present, they can appear in either order.

Table 121. SMF type 82 server user or end user audit section					
Offsets	Name	Length	Format	Description	
0	0	SMF82AUD_SECTION_TYPE	4	EBCDIC	Type of the section that follows. Either: <ul style="list-style-type: none"> • 'SERV' (for server user) • 'USER' (for end user)
4	4	SMF82AUD_SECTION_NUM_FLDS	2	Binary	Number of triples in this section
6	6	SMF82AUD_SECTION_TOTAL_LEN	2	Binary	Overall length of this section, including this header
8	8	Tag-Length-Value (TLV) triplets start here and are defined in Table 122 on page 342 . These repeat as many times as the SMF82AUD_SECTION_NUM_FLDS field indicates.			

Each Tag-Length-Value (TLV) triplet is a structure that is called SMF82AUD_TRIPLET and is defined as follows. The values for the tags and the format and maximum length of the data are defined in [Table 123 on page 342](#).

Table 122. Tag-Length-Value (TLV) triplet structure (SMF82AUD_TRIPLET)					
Offsets	Name	Length	Format	Description	
0	0	SMF82AUD_TRIPL_TAG	2	Binary	Tag of the information in this TLV
2	2	SMF82AUD_TRIPL_LENGTH	2	Binary	Length of this TLV including these first two fixed fields
4	4	SMF82AUD_TRIPL_DATA	*	Varies	Data for this TLV

The tag values and their corresponding information are described in the following table. The tag value is defined in the constant SMF82AUD_TAG_XXX and the maximum length in SMF82AUD_MAXLEN_XXX. For example, the tag for X500_IDN is SMF82AUD_TAG_X500_IDN and maximum length of the associated data is SMF82AUD_MAXLEN_X500_IDN.

Table 123. TLV triplet tag values					
Tag Value	Name	Length	Format	Description	
1	1	X500_IDN	0-255	EBCDIC	X.500 Certificate Issuer's Distinguished Name (ACEEX5PR->IDN)
2	2	X500_SDN	0-255	EBCDIC	X.500 Certificate Subject's Distinguished Name (ACEEX5PR->SDN)
10	A	IDID_USRI	1-246	UTF-8	X.500 Distinguished Name of distributed client end user (ACEEIDID-> IDID1UDN)
11	B	IDID_USRF	1	Binary	Format of IDID_USRI (ACEEIDID->IDID1NMF) <ul style="list-style-type: none"> 0 Undetermined 1 Straight string 2 X.500 format
12	C	IDID_REG	1-255	UTF-8	Name of the registry that authenticated the user (ACEEIDID->IDID1RN)

Table 123. TLV triplet tag values (continued)					
Tag Value	Name	Length	Format	Description	
14 E	USRI	8	EBCDIC	RACF user ID (ACEEUSRI)	
15 F	GRPN	8	EBCDIC	Connect group (ACEEGRPN)	
16 10	TRM_USER	8	EBCDIC	Terminal ID (ACEETRM)	
17 11	JOB_JBN	8	EBCDIC	Job name (JMRJOB)	
18 12	JOB_RST	4	Binary	Job entry time (JMRENTY) in hundredths of a second that the reader recognized the JOB statement for this job. This field can be zero.	
26 1A	JOB_RSD	4	Binary	Job entry date (JMREDATE) that the reader recognized the JOB statement for this job in the form 0CYDDDF. This field can be zero.	
34 22	JOB_UID	8	Binary	User-defined identification field (JMRUSEID)	
42 2A	SEC	8	EBCDIC	Security label (TOKSCL)	

Tag-Length-Value (TLV) triplets

Where used within a subtype, Tag-Length-Value triplets are in the following format:

Table 124. Tag-Length-Value triplets					
Offsets		Name	Length	Format	Description
Dec	Hex				
0	0	SMF82_TRIPL_TAG	2	binary	The tag identifying the type of data that this triplet contains.
2	2	SMF82_TRIPL_LEN	2	binary	Length of this triplet including the tag and the length fields.
4	4	SMF82_TRIPL_VAL			The value for this triplet.

Subtype 1

Initialization/Options Refresh section

Table 125. Subtype 1 Initialization/Options Refresh

Offsets	Name	Length	Format	Description
0	0 SMF82VES	4	binary	<p>Cryptographic communication vector table extension (CCVE) status bits</p> <p>Bit</p> <p>Meaning When Set</p> <p>0 Special security mode allowed</p> <p>1 Reserved</p> <p>2 RNG Cache enabled</p> <p>3-5 Reserved</p> <p>6 RACF checking for authorized callers</p> <p>7-14 Reserved</p> <p>15 Reserved</p> <p>16 Default wrapping for internal tokens is the enhanced method</p> <p>17 Default wrapping for external tokens is the enhanced method</p> <p>18 Key archive reference message</p> <p>19-31 Reserved</p>
4	4 SMF82VTS	1	binary	<p>Cryptographic communication vector table (CCVT) status bits</p> <p>Bit</p> <p>Meaning When Set</p> <p>0-3 Reserved</p> <p>4 Compatible with CUSP and PCF</p> <p>5-7 Reserved.</p>
5	5 SMF82IDO	1	binary	Current crypto domain index.
6	6	6		Reserved.
12	C SMF82CKD	44	EBCDIC	Name of the cryptographic key data set (CKDS) that was read into storage.
56	38 SMF82IML	4	binary	Maximum length for data.
60	3C SMF82USR	8	EBCDIC	USERPARM specifies installation use in the installation options data set.
68	44 SMF82PKD	44	EBCDIC	PKDS name.
112	70 SMF82TKS	44	EBCDIC	TKDS name.

Subtype 7

Operational key load section

Table 126. Subtype 7 operational key entry				
Offsets	Name	Length	Format	Description
0	0 SMF82KPB	4	binary	Key part (KPART) bits Bit Meaning When Set 0 Key part verification pattern valid. 1 Coprocessor is a PCIXCC. (This has been deprecated; see SMF82KAP.) 2 Coprocessor is a CEX2C. (This has been deprecated; see SMF82KAP.) 3 Coprocessor is a CEX3C. (This has been deprecated; see SMF82KAP.) 4 Coprocessor is a CEX4C or higher. (This has been deprecated; see SMF82KAP.) 5-31 Reserved.
4	4 SMF82KV	8	binary	Key part verification pattern.
12	C SMF82KKS	1	binary	Coprocessor number.
13	D SMF82KDX	1	binary	Current crypto domain index.
14	E SMF82KAP	1	binary	Coprocessor type: X'05' Coprocessor is a PCIXCC. X'07' Coprocessor is a CEX2C. X'09' Coprocessor is a CEX3C. X'0A' Coprocessor is a CEX4C. X'0B' Coprocessor is a CEX5C or higher.
15	F	1		Reserved.
16	10 SMF82KCK	44	EBCDIC	Name of the CKDS containing the key part.
60	3C SMF82KCL	72	EBCDIC	CKDS entry being modified.

Subtype 8

Cryptographic key data set refresh section

Table 127. Subtype 8 Cryptographic key data set refresh				
Offsets	Name	Length	Format	Description
0	0 SMF82ROC	44	EBCDIC	Name of the CKDS being replaced.
44	2C SMF82RNC	44	EBCDIC	Name of the CKDS to replace the current CKDS.

Subtype 9

Dynamic CKDS update

Table 128. Subtype 9 Dynamic CKDS update				
Offsets	Name	Length	Format	Description
0	0 SMF82UCB	4	binary	Update CKDS bits Bit Meaning When Set 0 CKDS record added 1 CKDS record changed 2 CKDS record deleted 3 CKDS record archived 4 CKDS record recalled 5 CKDS record metadata changed 6-31 Reserved.
4	4 SMF82UCN	44	EBCDIC	CKDS name.
48	30 SMF82UCL	72	EBCDIC	CKDS entry being modified.

Subtype 13

Dynamic PKDS update

Table 129. Subtype 13 Dynamic PKDS update				
Offsets	Name	Length	Format	Description
0	0 SMF_PKDS_BITS	4	binary	Update PKDS bits Bit Meaning When Set 0 PKDS record added 1 PKDS record changed 2 PKDS record deleted 3 PKDS record archived 4 PKDS record recalled 5 PKDS record metadata changed 6-31 Reserved.
4	4 SMF_PKDS_NAME	44	EBCDIC	PKDS name.
48	30 SMF_PKDS_KEY_LABEL	72	EBCDIC	PKDS entry being modified.

Subtype 14

Cryptographic coprocessor master key entry

Table 130. Subtype 14 Cryptographic coprocessor master key entry

Offsets	Name	Length	Format	Description
0	0 SMF82AAB	4	binary	<p>Flag bytes</p> <p>Bit</p> <p>Meaning When Set</p> <p>0 DES NMK verification pattern is valid.</p> <p>1 RSA NMK verification pattern is valid.</p> <p>2 DES Key key part verification pattern is valid.</p> <p>3 RSA Key Key part verification pattern is valid.</p> <p>4 AES NMK verification pattern is valid.</p> <p>5 AES key part verification pattern is valid.</p> <p>6 ECC NMK verification pattern is valid.</p> <p>7 ECC key part verification pattern is valid.</p> <p>8 Always on.</p> <p>9 Coprocessor is a PCIXCC. (This has been deprecated; see SMF82AAP.)</p> <p>10 Coprocessor is a CEX2C. (This has been deprecated; see SMF82AAP.)</p> <p>11 Coprocessor is a CEX3C. (This has been deprecated; see SMF82AAP.)</p> <p>12 Coprocessor is a CEX4C or higher. (This has been deprecated; see SMF82AAP.)</p> <p>13-24 Reserved.</p> <p>25 DES NMK entered was 24-bytes long.</p> <p>26-31 Reserved.</p>
4	4 SMF82ANV	16	EBCDIC	New master key register verification pattern.
20	14 SMF82AKV	16	EBCDIC	Key part verification pattern.
36	24 SMF82APN	1	binary	Cryptographic Processor number.
37	25 SMF82ASN	8	EBCDIC	Cryptographic Processor serial number.
45	2D SMF82ADM	1	binary	Cryptographic Coprocessor domain.

Table 130. Subtype 14 Cryptographic coprocessor master key entry (continued)

Offsets	Name	Length	Format	Description
46	2E SMF82AAP	1	binary	Coprocessor type: X'05' Coprocessor is a PCIXCC. X'07' Coprocessor is a CEX2C. X'09' Coprocessor is a CEX3C. X'0A' Coprocessor is a CEX4C. X'0B' Coprocessor is a CEX5C or higher.
47	2F	1		Reserved.

Subtype 15

PCI Cryptographic coprocessor retained key create/delete

Table 131. Subtype 15 PCI Cryptographic coprocessor retained key create/delete

Offsets	Name	Length	Format	Description
0	0 SMF82RKF	4	binary	First flag byte Bit Meaning When Set 0 Retained key created. 1 Retained key deleted on coprocessor. 2 Retained key deleted from PKDS. 3-7 Reserved. 8 Always on. 9 Coprocessor is a PCIXCC. (This has been deprecated; see SMF82RAP.) 10 Coprocessor is a CEX2C. (This has been deprecated; see SMF82RAP.) 11 Coprocessor is a CEX3C. (This has been deprecated; see SMF82RAP.) 12 Coprocessor is a CEX4C or higher. (This has been deprecated; see SMF82RAP.) 13-31 Reserved.
4	4 SMF82RKN	64	EBCDIC	Label of Retained private key.
68	44 SMF82RKP	1	binary	Cryptographic Coprocessor number.
69	45 SMF82RKS	8	EBCDIC	Cryptographic Coprocessor serial number.
77	4D SMF82RDM	1	binary	Cryptographic Coprocessor domain.

Table 131. Subtype 15 PCI Cryptographic coprocessor retained key create/delete (continued)

Offsets	Name	Length	Format	Description
78	4E SMF82RAP	1	binary	Coprocessor type: X'05' Coprocessor is a PCIXCC. X'07' Coprocessor is a CEX2C. X'09' Coprocessor is a CEX3C. X'0A' Coprocessor is a CEX4C. X'0B' Coprocessor is a CEX5C or higher.
79	4F	1		Reserved.

Subtype 16

PCI Cryptographic coprocessor TKE

Table 132. Subtype 16 PCI Cryptographic Coprocessor TKE

Offsets	Name	Length	Format	Description
0	0 SMF82PFL	4	binary	Flag bytes Bit Meaning When Set 0 Request command. 1 Reply response. 2-7 Reserved. 8 Always on. 9 Coprocessor is a PCIXCC. (This has been deprecated; see SMF82PAP.) 10 Coprocessor is a CEX2C. (This has been deprecated; see SMF82PAP.) 11 Coprocessor is a CEX3C. (This has been deprecated; see SMF82PAP.) 12 Coprocessor is a CEX4 or higher. (This has been deprecated; see SMF82PAP.) 13-29 Reserved 30 Coprocessor is configured for CCA. 31 Coprocessor is configured for PKCS #11.
4	4 SMF82PPN	1	binary	Cryptographic Coprocessor number.
5	5 SMF82PSN	8	EBCDIC	Cryptographic Coprocessor serial number.
13	D SMF82PDM	1	binary	Cryptographic Coprocessor domain.

Table 132. Subtype 16 PCI Cryptographic Coprocessor TKE (continued)

Offsets	Name	Length	Format	Description
14	E SMF82PAP	1	binary	Coprocessor type: X'05' Coprocessor is a PCIXCC. X'07' Coprocessor is a CEX2C. X'09' Coprocessor is a CEX3C. X'0A' Coprocessor is a CEX4C. X'0B' Coprocessor is a CEX5 or higher. Note: For CEX4 and higher, bits 30 and 31 at offset 0 indicate whether the coprocessor is configured as a CCA or PKCS #11 coprocessor.
15	F	1		Reserved.
16	10 SMF82PBL	4	binary	Parameter block length, "xxx".
20	14 SMF82PDL	4	binary	Parameter data block length, "yyy".
24	18 SMF82PBK			Parameter block of length "xxx" followed by parameter data block of length "yyy".

Fixed length audit data – begins at offset 24 + xxx + yyy.

Table 133. Subtype 16 PCI Cryptographic Coprocessor TKE audit data

Offsets	Name	Length	Format	Description
0	0 SMF82P16		structure	Fixed length audit data
0	0 SMF82PAL	4	binary	Length of fixed audit data
4	4 SMF82PAD	4	binary	PKCS #11 Admin request ID. All zeros if not applicable
8	8 SMF82PFI	2	binary	Function ID
10	A SMF82PFR	4	binary	Function Return code 0 Success 4 Not authorized 8 Error
14	E SMF82PDE	256	EBCDIC	Function description
270	10E SMF82PUS	20	binary	Transaction Sequence Number (TSN) for commands or, for CCA coprocessor requests only, User ID Nonce (random number) for queries. All blanks if not applicable
290	122 SMP82PTA	8	EBCDIC	TKE Authority for CCA coprocessor requests. Blanks for PKCS #11 coprocessor requests

Subtype 18

Cryptographic processor configuration

Table 134. Subtype 18 Cryptographic Processor Configuration				
Offsets	Name	Length	Format	Description
0	0 SMF82CGB	4	binary	<p>Flag bytes</p> <p>Bit</p> <p>Meaning When Set</p> <p>0 A Cryptographic processor has been brought online.</p> <p>1 A Cryptographic processor has been taken offline.</p> <p>2-7 Reserved.</p> <p>8 Always on.</p> <p>9 Coproprocessor is a PCIXCC. (This has been deprecated; see SMF82CAP.)</p> <p>10 Coproprocessor is a CEX2C. (This has been deprecated; see SMF82CAP.)</p> <p>11 Coproprocessor is a CEX2A. (This has been deprecated; see SMF82CAP.)</p> <p>12 Coproprocessor is a CEX3C. (This has been deprecated; see SMF82CAP.)</p> <p>13 Coproprocessor is a CEX3A. (This has been deprecated; see SMF82CAP.)</p> <p>14 Coproprocessor is a CEX4 or higher. (This has been deprecated; see SMF82CAP.)</p> <p>15-28 Reserved.</p> <p>29 Configured as an accelerator</p> <p>30 Configured as a CCA coprocessor</p> <p>31 Configured as a PKCS #11 coprocessor</p>
4	4 SMF82CGX	1	binary	Cryptographic Coprocessor number.
5	5 SMF82CGS	8	EBCDIC	Cryptographic Coprocessor serial number.

Table 134. Subtype 18 Cryptographic Processor Configuration (continued)

Offsets	Name	Length	Format	Description
13	D SMF82CAP	1	binary	Coprocessor type: X'04' Coprocessor is a PCICA. X'05' Coprocessor is a PCIXCC. X'06' Coprocessor is a CEX2A. X'07' Coprocessor is a CEX2C. X'08' Coprocessor is a CEX3A. X'09' Coprocessor is a CEX3C. X'0A' Coprocessor is a CEX4. X'0B' Coprocessor is a CEX5 or higher. Note: For CEX4 and higher, bits 29, 30, and 31 at offset 0 indicate whether the coprocessor is configured as an accelerator, a CCA, or a PKCS #11 coprocessor.
14	E	2		Reserved.

Subtype 19

PCI X Cryptographic coprocessor timing

Table 135. Subtype 19 PCI X Cryptographic Coprocessor Timing

Offsets	Name	Length	Format	Description
0	0 SMF82XTN	8	EBCDIC	Time just before the PCI X Cryptographic Coprocessor operation begins.
8	8 SMF82XTD	8	EBCDIC	Time just after PCI X Cryptographic Coprocessor operation ends.
16	10 SMF82XTW	8	EBCDIC	Time just after results have been communicated to caller address space.
24	18 SMF82XTQ	4	binary	Number of processes waiting to submit work to the same PCI X Cryptographic Coprocessor and domain, using the same reference number.
28	1C SMF82XTF	2	EBCDIC	Function code of service.
30	1E SMF82XTX	1	binary	PCI X Cryptographic Coprocessor number.
31	1F SMF82XTS	8	EBCDIC	PCI X Cryptographic Coprocessor serial number.
39	27 SMF82XTM	1	binary	PCI X Cryptographic Coprocessor domain.
40	28 SMF82XTR	1	binary	PCI X Cryptographic Coprocessor reference number.
41	29	3		Reserved.

Subtype 20

Cryptographic processor processing times

Table 136. Subtype 20 Cryptographic Processor Processing Times				
Offsets	Name	Length	Format	Description
0	0 SMF82TFL	4	binary	<p>Flag bytes</p> <p>Bit</p> <p>Meaning When Set</p> <p>0 Processor is a PCIXCC or PCICA. (This has been deprecated; see SMF82TPT.)</p> <p>Note: The record is for a PCIXCC when bits 0 and 30 are on and for a PCICA with bits 0 and 29 are on.</p> <p>1 Coproprocessor is a CEX2C. (This has been deprecated; see SMF82TPT.)</p> <p>2 Coproprocessor is a CEX2A. (This has been deprecated; see SMF82TPT.)</p> <p>3 Coproprocessor is a CEX3C. (This has been deprecated; see SMF82TPT.)</p> <p>4 Coproprocessor is a CEX3A. (This has been deprecated; see SMF82TPT.)</p> <p>5 Coproprocessor is a CEX4 or higher. (This has been deprecated; see SMF82TPT.)</p> <p>6 Regional cryptographic server.</p> <p>7–28 Reserved.</p> <p>29 Configured as an accelerator.</p> <p>30 Configured as a CCA coprocessor.</p> <p>31 Configured as a PKCS #11 coprocessor.</p>
4	4 SMF82TNQ	8	binary	Coproprocessor time before NQAP.
12	C SMF82TDQ	8	binary	Coproprocessor time after DQAP.
20	14 SMF82TWT	8	binary	Coproprocessor time after WAIT.
28	1C SMF82TQU	4	binary	Coproprocessor queue length.
32	20 SMF82TSF	2	EBCDIC	Coproprocessor sub function code.
34	22 SMF82TIX	1	binary	Coproprocessor index.
35	23 SMF82TSN	8	EBCDIC	Coproprocessor serial number.
43	2B SMF82TDM	1	binary	Domain.
44	2C SMF82TRN	1	binary	Reference number.

Table 136. Subtype 20 Cryptographic Processor Processing Times (continued)				
Offsets	Name	Length	Format	Description
45	2D SMF82TPT	1	binary	Coprocessor type: X'04' Coprocessor is a PCICA. X'05' Coprocessor is a PCIXCC. X'06' Coprocessor is a CEX2A. X'07' Coprocessor is a CEX2C. X'08' Coprocessor is a CEX3A. X'09' Coprocessor is a CEX3C. X'0A' Coprocessor is a CEX4. X'0B' Coprocessor is a CEX5 or higher. Note: For CEX4 and higher, bits 29, 30, and 31 at offset 0 indicate whether the coprocessor is configured as an accelerator, a CCA, or a PKCS #11 coprocessor.
46	2E	2		Reserved.

Subtype 21

ICSF sysplex group change section

Table 137. Subtype 21 ICSF Sysplex Group Change				
Offsets	Name	Length	Format	Description
0	0 SMF82SXG	8	EBCDIC	Name of ICSF Sysplex group.
8	8 SMF82SXM	8	EBCDIC	Name of sysplex member.
16	F SMF82SXA	1	binary	ICSF Sysplex member status flags Bit Meaning When Set 0 Member joined the ICSF sysplex group. 1 Member left the ICSF sysplex group. 2–7 Reserved.
17	11 SMF82SXR	1	binary	ICSF Sysplex member conditions of status flags Bit Meaning When Set 0 Member joined or left the ICSF sysplex due to normal initialization/termination processing 1 Member left the ICSF sysplex due to error 2–7 Reserved.
18	12	2		Reserved.
20	14 SMF82SXT	8	EBCDIC	Time of ICSF sysplex join/leave index.
28	1C SMF82SXC	44	EBCDIC	Name of active CKDS.

Subtype 22

Trusted block create callable services section

Table 138. Subtype 22 Trusted Block Create Callable Services				
Offsets	Name	Length	Format	Description
0	0 SMF82TBF	4	binary	Process Flag bytes Bit Meaning When Set 0 Created Inactive Trusted Block. 1 Activate an Inactive Block. 2 Trusted Block has Public Key. 3–31 Reserved.
4	4 SMF82TBS	2	EBCDIC	ASID of caller.
6	6 SMF82TBN	64	EBCDIC	Label of Input Trusted Block.
70	46 SMF82TBO	64	EBCDIC	Label of Output Trusted Block.
134	86 SMF82TBX	64	EBCDIC	Label of Transport Key.

Subtype 23

Token data set update

Table 139. Subtype 23 Token Data Set Update				
Offsets	Name	Length	Format	Description
0	0 SMF82TKF	4	binary	TKDS bits Bit Meaning When Set 0 TKDS record added 1 TKDS record changed 2 TKDS record deleted 3 TKDS record archived 4 TKDS record recalled 5 TKDS record metadata changed 6–31 Reserved.
4	4 SMF82TKN	44	EBCDIC	TKDS name
48	30 SMF82TKH	44	EBCDIC	TKDS handle being processed

Subtype 24

Duplicate tokens found

Table 140. Subtype 24 Duplicate Tokens Found					
Offsets	Name	Length	Format	Description	
0	0 SMF82DCNTSTRT	4	binary	Start of duplicate labels.	
4	4 SMF82DCNTEND	4	binary	End of duplicate labels.	
8	8 SMF82DCNT	4	binary	Number of duplicate labels.	
12	C SMF82DRSVD	4	binary	Reserved.	
16	10 SMF82DNAM	44	binary	Name of key data set.	
The following field is repeated <i>count</i> (SMF82DCNT) number of times.					
60	3C SMF82_Label	64	EBCDIC	A key label.	

Subtype 25**Key store policy for key token authorization checking**

The key store policy must be activated before this SMF record subtype is logged. The subtype is logged when the callable service request fails the Key Token Authorization Checking key store policy check.

Table 141. Subtype 25 Key Store Policy Key Token Authorization Checking					
Offsets	Name	Length	Format	Description	
0	0 SMF82KDS	44	EBCDIC	Data set name.	
44	2C SMF82KLF	4	binary	<div>Key store policy flags:</div> <div><div>Bit</div><div>Meaning When Set</div></div> <div><div>0</div><div>Warning.</div></div> <div><div>1</div><div>List is incomplete.</div></div> <div><div>2</div><div>List is from CKDS.</div></div> <div><div>3</div><div>List is from PKDS.</div></div> <div><div>4</div><div>Authorization failures.</div></div> <div><div>5</div><div>Archived failures.</div></div> <div><div>6</div><div>Preactive failures.</div></div> <div><div>7</div><div>Deactivated failures.</div></div> <div><div>8-31</div><div>Reserved.</div></div>	
48	30 SMF82KLC	4	binary	Number of key labels following.	
The following field is repeated <i>count</i> (SMF82KLC) number of times.					
52	34 SMF82DKL	72	EBCDIC	Unauthorized duplicate key label and key type.	

Subtype 26

Public key data set refresh

Table 142. Subtype 26 Public Key Data Set Refresh					
Offsets	Name	Length	Format	Description	
0	0 SMF82PREF_FLAG	4	binary	Flags: Bit Meaning When Set 0 Data space was refreshed. 1-31 Reserved.	
4	4 SMF82_PREF_OLDDS	44	EBCDIC	Old PKDS Name.	
48	30 SMF82_PREF_NEWDS	44	EBCDIC	New PKDS Name.	

Subtype 27

PKA key management extensions

Table 143. Subtype 27 PKA Key Management Extensions					
Offsets	Name	Length	Format	Description	
24	18 SMF82PKE_FLAGS	4	binary	PKA Key Management Extension flags: Bit Meaning When Set 0 PKA token may not be used for requested function. 1 SYM token may not be exported by the provided PKA token. 2 PKA label list is incomplete. 3 SYM label list is incomplete. 24 Trusted certificate repository has changed. 25 PKA Key Management Extensions in WARNONLY mode. 26 An error was detected during processing. 27 Trusted cert repository was empty. 28 An error was detected while extracting APPLDATA. 29 The repository was not found. 30 One or more certs could not be parsed. Bits 0-3 are set during callable services. Bits 24-30 are set during repository parsing. Bits 4-23 and 31 are reserved.	
28	1C SMF82PKE_FUNCTION	8	EBCDIC	Name of the service that issued this SMF record. The name is in the form CSFzzz.	
36	24 SMF82PKE_APPLDATALEN	1	binary	Length of the enablement profile APPLDATA or current repository name.	

Record Type 82

Table 143. Subtype 27 PKA Key Management Extensions (continued)

Offsets	Name	Length	Format	Description
37 25	SMF82PKE_APPLDATA	247	EBCDIC	Enablement profile APPLDATA or current repository name.
284 11C	SMF82PKE_FUNCSPEC	0	binary	Function-specific section of the record.
284 11C	SMF82PKE_APPLDATA_PARSING	0	binary	APPLDATA parsing results section.
284 11C	SMF82PKE_SAF_RC	2	binary	SAF_RC or 'FFFF'X.
286 11E	SMF82PKE_SERV_RC	2	binary	RACF RC or ICSF RC.
288 120	SMF82PKE_SERV_RS	4	binary	RACF RS or ICSF RS.
284 11C	SMF82PKE_SERVICE_SECTION	0	binary	Callable services section.
284 11C	SMF82PKE_PKA_REC_CNT	4	binary	Number of PKA labels present in this record.
288 120	SMF82PKE_SYM_REC_CNT	4	binary	Number of SYM labels present in this record.
The following is repeated SMF82PKE_PKA_REC_CNT number of times.				
292 124	SMF82PKE_PKA_LABELS	64	EBCDIC	PKA key label.
The following is repeated SMF82PKE_SYM_REC_CNT number of times.				
292+ zzz	124+ zzz SMF82PKE_SYM_LABELS	72	EBCDIC	SYM key label.

Subtype 28

High performance encrypted key

Table 144. Subtype 28 High Performance Encrypted Key

Offsets	Name	Length	Format	Description
24 18	SMF82HPSK_FLAGS	4	binary	High Performance Encrypted Key flags: Bit Meaning When Set 0 Rewrapping operation is not permitted for this symmetric key. 1 Rewrapping operation was permitted for this symmetric key. 2 The list of labels is incomplete. 3 The key identifier was supplied as a key token, not as a label in the CKDS. Bits 4–31 are reserved.
28 1C	SMF82HPSK_FUNCTION	8	EBCDIC	Name of the service that issues this SMF record. The name is in the form of CSFzzzz.
36 24	SMF82HPSK_SYM_LABEL_CNT	4	binary	Number of SYM labels present in this record.
The following is repeated SMF82HPSK_SYM_LABEL_CNT number of times.				
40 28	SMF82HPSK_SYM_LABELS	72	EBCDIC	SYM key label and type.

Subtype 29

TKE workstation audit record

Table 145. Subtype 29 TKE Workstation Audit Record					
Offsets	Name	Length	Format	Description	
24	18 SMF82TKEAR_FLAGS	4	binary	Flags -- reserved	
28	1C SMF82TKEAR_NAMELEN	2	binary	TKE workstation name length	
30	24 SMF82TKEAR_RCDLEN	2	binary	TKE audit record data length	
32	20 SMF82TKEAR_NAME	VAR	EBCDIC	TKE workstation name	
VAR	VAR	VAR	EBCDIC	TKE audit record data	

Subtype 30**Key store policy archived and inactive KDS records**

Table 146. Subtype 30 Archived and inactive KDS records					
Offsets	Name	Length	Format	Description	
0	0 SMF_ARCH_FLAGS	4	binary	Flag bytes Bit Meaning When Set 0 CKDS 1 PKDS 2 TKDS 3-7 Reserved. 8 Record that is archived was referenced by service. By policy, service call failed. 9 Record that is archived was referenced by service. By policy, service call succeeded 10 Record that is pre-active was referenced by service. Service call failed. 11 Record that is inactive was referenced by service. Service call failed. 12-31 Reserved	
4	4 SMF_ARCH_DSNAME	44	EBCDIC	Key data set name.	
48	30 SMF_ARCH_KEY_LABEL	72	EBCDIC	Key data set entry.	

Subtype 40**CCA symmetric key lifecycle event**

This subtype consists of a number of tag-length-value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets is dependent on the type of event and the information that is available. This replaces subtype 9.

Table 147. Subtype 40 CCA symmetric key lifecycle event

Tag value		Name	Length	Format	Description
Dec	Hex				
256	100	KEY_EVENT	1	binary	<p>Key event. This field always occurs first in the record.</p> <p>X'10' Key token added to KDS.</p> <p>X'11' Key token updated in KDS.</p> <p>X'12' Key token deleted from KDS.</p> <p>X'13' Key token archived.</p> <p>X'14' Key token restored.</p> <p>X'15' Key token metadata changed.</p> <p>X'17' Key token pre-activated.</p> <p>X'18' Key token activated.</p> <p>X'19' Key token deactivated.</p> <p>X'1B' Key token exported.</p> <p>X'20' Key token generated.</p> <p>X'21' Key token imported.</p> <p>Note:</p> <ol style="list-style-type: none"> 1. When a key is exported, the key token that gets audited is the input or source token. 2. When a key is imported, the key token that gets audited is the output or target token.
257	101	KDS_LABEL	72	EBCDIC	The label in the KDS.
258	102	KDS_DSNAME	44	EBCDIC	The dataset name of the KDS associated with the event. If there is no associated KDS (for example, the event only involves a token), this field is not present.
259	103	KEY_NAME	64	EBCDIC	The key name from the token. Applies to variable-length CCA tokens only.

Table 147. Subtype 40 CCA symmetric key lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
261	105	KEY_FPRINT	1 - 64	binary	<p>One or more key fingerprints.</p> <p>The first byte is the number (<i>n</i>) of fingerprints present for the key. Following that are <i>n</i> type-length-value triplets. Within each of these triplets is a 1-byte fingerprint type, followed by a 1-byte length for the triplet, followed by the fingerprint.</p> <p>Fingerprint types:</p> <p>X'01' Ciphertext obtained from encrypting a data block filled with binary zeros in ECB mode.</p> <p>X'03' SHA-256 algorithm. See Appendix E in <i>z/OS Cryptographic Services ICSF Application Programmer's Guide</i> for more information.</p> <p>X'04' SHA1P1 algorithm. See Appendix E in <i>z/OS Cryptographic Services ICSF Application Programmer's Guide</i> for more information.</p> <p>For example, X'010105010203' indicates that there is one fingerprint value (01) which is the ciphertext obtained from using the key to encrypt a data block of binary zeros in ECB mode (01). The fingerprint is 3 bytes in length (05 – 2) and the value is X'010203'.</p> <p>Note: If the event pertains to a record in the CKDS, the key fingerprint is only present when using a KDSR-format dataset.</p>
262	106	SERVICE	8	EBCDIC	The service associated with the event.
264	108	TOK_FMT	1	binary	<p>The format of the token.</p> <p>X'01' Fixed length CCA token.</p> <p>X'02' Variable length CCA token.</p> <p>X'03' TR-31 key block.</p> <p>X'04' RKX token.</p> <p>Note:</p> <ol style="list-style-type: none"> 1. When format is RKX token, no other key or token related fields are present. 2. When format is TR-31 key block, the only other key or token related field that may be present is the key fingerprint.
265	109	KEY_SEC	1	binary	<p>Key security.</p> <p>X'01' No key present.</p> <p>X'02' Clear key.</p> <p>X'03' Key encrypted under master key.</p> <p>X'04' Key encrypted under key encrypting key.</p>

Table 147. Subtype 40 CCA symmetric key lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
266	10A	KEY_ALG	1	binary	Key algorithm. X'02' DES. X'03' AES. X'04' HMAC.
267	10B	KEY_TYPE	2	binary	Key type. The key type from the token. Applies to variable-length CCA tokens only. See "Variable-length symmetric key token" in z/OS Cryptographic Services ICSF Application Programmer's Guide for the list of key types.
268	10C	KEY_CV	8	binary	Key control vector. The first eight bytes of the control vector from the token. Applies to fixed-length DES CCA tokens only. See Appendix C in z/OS Cryptographic Services ICSF Application Programmer's Guide for information on how to interpret the control vector.
269	10D	KEY_USAGE_CKDS	3 - 33	binary	Key usage fields. Consists of a 1 byte count followed by one or more 2-byte key usage fields. Applies to variable-length CCA tokens only. See Appendix B in z/OS Cryptographic Services ICSF Application Programmer's Guide for the list of key usage values for variable length tokens.
270	10E	KEY_LEN	2	binary	The length of the key (in bits). Applies to fixed-length CCA tokens only.
271	10F	KEY_CP	16	EBCDIC	Key crypto period. The start date followed by the end date for the record in YYYYMMDD format (therefore, YYYYMMDDYYYYMMDD). If a date is not set, that portion of the key crypto period will be blanks. Applies to records in the KDS which have at least one date set.

Subtype 41

CCA asymmetric key lifecycle event

This subtype consists of a number of tag-length-value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets is dependent on the type of event and the information that is available. This replaces subtype 13.

Table 148. Subtype 41 CCA asymmetric key lifecycle event

Tag value		Name	Length	Format	Description
Dec	Hex				
256	100	KEY_EVENT	1	binary	<p>Key event. This field always occurs first in the record.</p> <p>X'10' Key token added to KDS.</p> <p>X'11' Key token updated in KDS.</p> <p>X'12' Key token deleted from KDS.</p> <p>X'13' Key token archived.</p> <p>X'14' Key token restored.</p> <p>X'15' Key token metadata changed.</p> <p>X'17' Key token pre-activated.</p> <p>X'18' Key token activated.</p> <p>X'19' Key token deactivated.</p> <p>X'1B' Key token exported.</p> <p>X'20' Key token generated.</p> <p>X'21' Key token imported.</p> <p>Note:</p> <ol style="list-style-type: none"> 1. When a key is exported, the key token that gets audited is the input or source token. 2. When a key is imported, the key token that gets audited is the output or target token.
257	101	KDS_LABEL	72	EBCDIC	The 64-byte KDS label left-justified and padded on the right with blanks.
258	102	KDS_DSNAME	44	EBCDIC	The dataset name of the KDS associated with the event. If there is no associated KDS (for example, the event only involves a token), this field is not present.
259	103	KEY_NAME	64	EBCDIC	The key name from the token.
260	104	OBJ_TYPE	1	binary	<p>Object type.</p> <p>X'02' Public key.</p> <p>X'0B' Public/Private key pair.</p> <p>X'0D' Trusted block.</p> <p>Note: When the object type is trusted block, no other key or token related information is present.</p>

Table 148. Subtype 41 CCA asymmetric key lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
261	105	KEY_FPRINT	1 - 64	binary	<p>One or more key fingerprints.</p> <p>The first byte is the number (<i>n</i>) of fingerprints present for the key. Following that are <i>n</i> type-length-value triplets. Within each of these triplets is a 1-byte fingerprint type, followed by a 1-byte length for the triplet, followed by the fingerprint.</p> <p>Fingerprint types:</p> <p>X'02' SHA-1 hash of the public key.</p> <p>For example, X'010205010203' indicates that there is one fingerprint value (01) which is the SHA-1 hash of the public key (02). The fingerprint is 3 bytes in length (05 – 2) and the value is X'010203'.</p> <p>Note: If the event pertains to a record in the PKDS, the key fingerprint is only present when using a KDSR-format dataset.</p>
262	106	SERVICE	8	EBCDIC	The service associated with the event.
265	109	KEY_SEC	1	binary	<p>Key security.</p> <p>X'01' No key present.</p> <p>X'02' Clear key.</p> <p>X'03' Key encrypted under master key.</p> <p>X'04' Key encrypted under key encrypting key.</p>
266	10A	KEY_ALG	1	binary	<p>Key algorithm.</p> <p>X'07' RSA.</p> <p>X'08' DSA.</p> <p>X'09' ECC.</p> <p>Note: When the algorithm is DSA, the only other key or token information that is present is the object type.</p>
270	10E	KEY_LEN	2	binary	The length of the key (in bits). For RSA, this is the length of the modulus. For ECC, this is the length of the private value.
271	10F	KEY_CP	16	EBCDIC	Key crypto period. The start date followed by the end date for the record in YYYYMMDD format (therefore, YYYYMMDDYYYYMMDD). If a date is not set, that portion of the key crypto period will be blanks. Applies to records in the KDS which have at least one date set.

Table 148. Subtype 41 CCA asymmetric key lifecycle event (continued)					
Tag value		Name	Length	Format	Description
Dec	Hex				
272	110	KEY_USAGE_PKDS	4	binary	Key usage for private keys. Bit Meaning when set 0 Undefined. 1 Key management usage permitted. 2 Signature usage permitted. 3 Key translation permitted. 4 Key agreement usage permitted. 5-31 Reserved.
274	112	KEY_EC_CURVE	1	binary	ECC curve type. X'01' Prime curve. X'02' Brainpool curve.

Subtype 42

PKCS#11 object lifecycle event

This subtype consists of a number of tag-length-value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets is dependent on the type of event and the information that is available. This replaces subtype 23.

Table 149. Subtype 42 PKCS#11 object lifecycle event					
Tag value		Name	Length	Format	Description
Dec	Hex				
256	100	KEY_EVENT	1	binary	Object event. This field always occurs first in the record. X'10' Object added to KDS. X'11' Object updated in KDS. X'12' Object deleted from KDS. X'13' Object archived. X'14' Object restored. X'15' Object metadata changed. X'17' Object pre-activated. X'18' Object activated. X'19' Object deactivated. X'1B' Object wrapped by another key.

Table 149. Subtype 42 PKCS#11 object lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
257	101	KDS_LABEL	72	EBCDIC	The 44-byte key handle left-justified and padded on the right with blanks. If the sequence number of the handle is 'FFFFFFF', this was a raw object.
258	102	KDS_DSNAME	44	EBCDIC	The dataset name of the KDS.
259	103	KEY_NAME	1 - 513	EBCDIC	The CKA_LABEL attribute from the object. If the CKA_Label is greater than 512 characters, the plus (+) symbol is placed at the 513th character to indicate truncation.
260	104	OBJ_TYPE	1	binary	Object type. X'01' Symmetric key. X'02' Public key. X'03' Private key. X'05' Certificate. X'06' Domain parameters. X'07' Data object. X'0C' PKCS #11 token.
261	105	KEY_FPRINT	1 - 64	binary	One or more key fingerprints. The first byte is the number (<i>n</i>) of fingerprints present for the key. Following that are <i>n</i> type-length-value triplets. Within each of these triplets is a 1-byte fingerprint type, followed by a 1-byte length for the triplet, followed by the fingerprint. Fingerprint types: X'01' Ciphertext obtained from encrypting a data block filled with binary zeros in ECB mode. X'02' SHA-1 hash of the public key. For example, X'010105010203' indicates that there is one fingerprint value (01) which is the ciphertext obtained from using the key to encrypt 8 bytes of binary zeros in ECB mode (01). The fingerprint is 3 bytes in length (05 – 2) and the value is X'010203'. Note: If the event pertains to a record in the TKDS, the key fingerprint is only present when using a KDSR-format dataset.
262	106	SERVICE	8	EBCDIC	The service associated with the event.
265	109	KEY_SEC	1	binary	Key security. X'02' Clear key. X'03' Key encrypted under master key.

Table 149. Subtype 42 PKCS#11 object lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
266	10A	KEY_ALG	1	binary	<p>Key algorithm.</p> <p>X'01' Generic symmetric.</p> <p>X'02' DES.</p> <p>X'03' AES.</p> <p>X'05' RC4.</p> <p>X'06' Blowfish.</p> <p>X'07' RSA.</p> <p>X'08' DSA.</p> <p>X'09' ECC.</p> <p>X'0A' Diffie-Hellman.</p>
270	10E	KEY_LEN	2	binary	The length of the key (in bits).
271	10F	KEY_CP	16	EBCDIC	Key crypto period. The start date followed by the end date for the record in YYYYMMDD format (therefore, YYYYMMDDYYYYMMDD). If a date is not set, that portion of the key crypto period will be blanks. Applies to records in the KDS which have at least one date set.
273	111	KEY_USAGE_TKDS	4	binary	<p>Key usage for private keys.</p> <p>Bit Meaning when set</p> <p>0 Data encryption allowed.</p> <p>1 Data decryption allowed.</p> <p>2 Key derivation allowed.</p> <p>3 Sign allowed where signature is appendix.</p> <p>4 Verify allowed where signature is appendix.</p> <p>5 Sign allowed where data is recovered from signature.</p> <p>6 Verify allowed where data is recovered from signature.</p> <p>7 Key wrapping allowed.</p> <p>8 Key unwrapping allowed.</p> <p>9 Key usage must be FIPS-compliant.</p> <p>10-31 Reserved.</p>

Table 149. Subtype 42 PKCS#11 object lifecycle event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
274	112	KEY_EC_CURVE	1	binary	ECC curve type. X'01' Prime curve. X'02' Brainpool curve.
279	117	FIPS_INFO	4	binary	FIPS information related to the event. Bit Meaning when set 0 FIPSMODE(YES) in effect. 1 FIPSMODE(COMPAT) in effect. 2 Request was evaluated for FIPS-compliance due to system settings. (Either FIPSMODE(YES) is in effect or FIPSMODE(COMPAT) is in effect, but the request was not exempt from FIPS-compliance.) 3 Request was evaluated for FIPS-compliance at user request. (Either the object involved had the FIPS compliance flag on or FIPS-compliance was requested via a parameter on the service call.) 4 Request passed FIPS evaluation. 5-31 Reserved.

Subtype 43

Regional cryptographic server configuration

Table 150. Subtype 43 Regional cryptographic server configuration

Offsets	Name	Length	Format	Description
0	0 SMF_RCS_CONFIG_FLAGS	4	binary	Regional cryptographic server configuration bits. Bit Meaning When Set 0 Regional cryptographic server brought online. 1 Regional cryptographic server taken offline. 2-31 Reserved.
4	4 SMF_RCS_CONFIG_INDEX	1	binary	Regional cryptographic server index.
5	5 SMF_RCS_CONFIG_SN	8	EBCDIC	Regional cryptographic server serial number.
13	D SMF_RCS_CONFIG_Port	5	EBCDIC	Regional cryptographic server port number.
18	12 SMF_RCS_CONFIG_HostLen	2	binary	Length of the regional cryptographic server host name.
20	14 SMF_RCS_CONFIG_Host	256	EBCDIC	Regional cryptographic server host name.
276	114 SMF_RCS_CONFIG_API	4	binary	Regional cryptographic server API level - VVRRxxxx.
280	118 SMF_RCS_CONFIG_Geo	2	binary	Regional cryptographic server geography.
282	11A SMF_RCS_CONFIG_GenMin	1	binary	Regional cryptographic server minimum compatible generation.

Table 150. Subtype 43 Regional cryptographic server configuration (continued)

Offsets	Name	Length	Format	Description
283 11B	SMF_RCS_CONFIG_GenCur	1	binary	Regional cryptographic server current generation.
284 11C		4		Reserved.

Subtype 44

CCA symmetric key usage event

This subtype consists of a number of tag-length-value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets is dependent on the type of event and the information that is available.

Table 151. Subtype 44 CCA symmetric key usage event

Tag value		Name	Length	Format	Description
Dec	Hex				
257	101	KDS_LABEL	72	EBCDIC	The label in the KDS.
259	103	KEY_NAME	64	EBCDIC	The key name from the token. Applies to variable-length CCA tokens only.
261	105	KEY_FPRINT	1 - 64	binary	<p>One or more key fingerprints.</p> <p>The first byte is the number (<i>n</i>) of fingerprints present for the key. Following that are <i>n</i> type-length-value triplets. Within each of these triplets is a 1-byte fingerprint type, followed by a 1-byte length for the triplet, followed by the fingerprint.</p> <p>Fingerprint types:</p> <p>X'01' Ciphertext obtained from encrypting a data block filled with binary zeros in ECB mode.</p> <p>X'03' SHA-256 algorithm. See Appendix E in z/OS Cryptographic Services ICSF Application Programmer's Guide for more information.</p> <p>X'04' SHAVP1 algorithm. See Appendix E in z/OS Cryptographic Services ICSF Application Programmer's Guide for more information.</p> <p>For example, X'010105010203' indicates that there is one fingerprint value (01) which is the ciphertext obtained from using the key to encrypt a data block of binary zeros in ECB mode (01). The fingerprint is 3 bytes in length (05 – 2) and the value is X'010203'.</p>
262	106	SERVICE	8	EBCDIC	The service associated with the event.
264	108	TOK_FMT	1	binary	<p>The format of the token.</p> <p>X'01' Fixed length CCA token.</p> <p>X'02' Variable length CCA token.</p> <p>X'03' TR-31 key block.</p> <p>X'04' RKX token.</p> <p>Note:</p> <ol style="list-style-type: none"> When format is RKX token, no other key or token related fields are present. When format is TR-31 key block, the only other key or token related field that may be present is the key fingerprint.

Table 151. Subtype 44 CCA symmetric key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
265	109	KEY_SEC	1	binary	Key security. X'01' No key present. X'02' Clear key. X'03' Key encrypted under master key. X'04' Key encrypted under key encrypting key.
266	10A	KEY_ALG	1	binary	Key algorithm. X'02' DES. X'03' AES. X'04' HMAC.
267	10B	KEY_TYPE	2	binary	Key type. The key type from the token. Applies to variable-length CCA tokens only. See “Variable-length symmetric key token” in z/OS Cryptographic Services ICSF Application Programmer's Guide for the list of key types.
268	10C	KEY_CV	8	binary	Key control vector. The first eight bytes of the control vector from the token. Applies to fixed-length DES CCA tokens only. See Appendix C in z/OS Cryptographic Services ICSF Application Programmer's Guide for information on how to interpret the control vector.
269	10D	KEY_USAGE_CKDS	3 - 33	binary	Key usage fields. Consists of a 1 byte count followed by one or more 2-byte key usage fields. Applies to variable-length CCA tokens only. See Appendix B in z/OS Cryptographic Services ICSF Application Programmer's Guide for the list of key usage values for variable length tokens.
270	10E	KEY_LEN	2	binary	The length of the key (in bits). Applies to fixed-length CCA tokens only.
275	113	START_TOD	16	binary	Start time of the interval in STCKE format.
276	114	END_TOD	16	binary	End time of the interval in STCKE format.
277	115	USG_COUNT	4	binary	Number of usages accounted for in this record.
278	116	KEY_OLD	0	binary	The key is internal, but not wrapped under the current master key. Additionally, if key store policy is enabled for CKDS, the key is wrapped under the old master key. Applies to token usage only.

The following tags may be present in the end user audit section:

- X500_IDN
- X500_SDN
- IDID_USRI
- IDID_USRF
- IDID_REG

- USRI

See [“Audit header and audit section”](#) on page 342 for more details.

Subtype 45

CCA asymmetric key usage event

This subtype consists of a number of tag-length-value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets is dependent on the type of event and the information that is available.

Table 152. Subtype 45 CCA asymmetric key usage event						
Tag value			Name	Length	Format	Description
Dec	Hex					
257	101	KDS_LABEL		72	EBCDIC	The 64-byte KDS label left-justified and padded on the right with blanks.
259	103	KEY_NAME		64	EBCDIC	The key name from the token.
260	104	OBJ_TYPE		1	binary	Object type. X'02' Public key. X'0B' Public/Private key pair. X'0D' Trusted block. Note: When the object type is trusted block, no other key or token related information is present.
261	105	KEY_FPRINT		1 - 64	binary	One or more key fingerprints. The first byte is the number (<i>n</i>) of fingerprints present for the key. Following that are <i>n</i> type-length-value triplets. Within each of these triplets is a 1-byte fingerprint type, followed by a 1-byte length for the triplet, followed by the fingerprint. Fingerprint types: X'02' SHA-1 hash of the public key. For example, X'010105010203' indicates that there is one fingerprint value (01) which is the ciphertext obtained from using the key to encrypt 8 bytes of binary zeros in ECB mode (01). The fingerprint is 3 bytes in length (05 – 2) and the value is X'010203'.
262	106	SERVICE		8	EBCDIC	The service associated with the event.
265	109	KEY_SEC		1	binary	Key security. X'01' No key present. X'02' Clear key. X'03' Key encrypted under master key. X'04' Key encrypted under key encrypting key.

Table 152. Subtype 45 CCA asymmetric key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
266	10A	KEY_ALG	1	binary	Key algorithm. X'07' RSA. X'08' DSA. X'09' ECC. Note: When the algorithm is DSA, the only other key or token information present is the object type.
270	10E	KEY_LEN	2	binary	The length of the public key (in bits).
272	110	KEY_USAGE_PKDS	4	binary	Key usage for private keys. Bit Meaning when set 0 Undefined. 1 Key management usage permitted. 2 Signature usage permitted. 3 Key translation permitted. 4 Key agreement usage permitted. 5-31 Reserved.
274	112	KEY_EC_CURVE	1	binary	ECC curve type. X'01' Prime curve. X'02' Brainpool curve.
275	113	START_TOD	16	binary	Start time of the interval in STCKE format.
276	114	END_TOD	16	binary	End time of the interval in STCKE format.
277	115	USG_COUNT	4	binary	Number of usages accounted for in this record.
278	116	KEY_OLD	0	N/A	The key is internal, but not wrapped under the current master key. Applies to token usage only.

The following tags may be present in the end user audit section:

- X500_IDN
- X500_SDN
- IDID_USRI
- IDID_USRF
- IDID_REG
- USRI

See [“Audit header and audit section”](#) on page 342 for more details.

Subtype 46

PKCS#11 key usage event

This subtype consists of a number of tag-length-value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets is dependent on the type of event and the information that is available.

Table 153. Subtype 46 PKCS#11 key usage event					
Tag value		Name	Length	Format	Description
Dec	Hex				
257	101	KDS_LABEL	72	EBCDIC	The 44-byte key handle left-justified and padded on the right with blanks. If the sequence number of the handle is 'FFFFFFF', this was a raw object.
259	103	KEY_NAME	1 - 513	EBCDIC	The CKA_LABEL attribute from the object. If the CKA_Label is greater than 512 characters, the plus (+) symbol is placed at the 513th character to indicate truncation.
260	104	OBJ_TYPE	1	binary	Object type. X'01' Symmetric key. X'02' Public key. X'03' Private key. X'05' Certificate. X'06' Domain parameters. X'07' Data object. X'0C' PKCS #11 token.
261	105	KEY_FPRINT	1 - 64	binary	One or more key fingerprints. The first byte is the number (<i>n</i>) of fingerprints present for the key. Following that are <i>n</i> type-length-value triplets. Within each of these triplets is a 1-byte fingerprint type, followed by a 1-byte length for the triplet, followed by the fingerprint. Fingerprint types: X'01' Ciphertext obtained from encrypting a data block filled with binary zeros in ECB mode. X'02' SHA-1 hash of the public key. For example, X'010105010203' indicates that there is one fingerprint value (01) which is the ciphertext obtained from using the key to encrypt 8 bytes of binary zeros in ECB mode (01). The fingerprint is 3 bytes in length (05 – 2) and the value is X'010203'.
262	106	SERVICE	8	EBCDIC	The service associated with the event.
265	109	KEY_SEC	1	binary	Key security. X'02' Clear key. X'03' Key encrypted under master key.

Table 153. Subtype 46 PKCS#11 key usage event (continued)

Tag value		Name	Length	Format	Description
Dec	Hex				
266	10A	KEY_ALG	1	binary	Key algorithm. X'01' Generic symmetric. X'02' DES. X'03' AES. X'05' RC4. X'06' Blowfish. X'07' RSA. X'08' DSA. X'09' ECC. X'0A' Diffie-Hellman.
270	10E	KEY_LEN	2	binary	The length of the key (in bits). For RSA, this is the modulus length. For other asymmetric keys, this is the length of the public key.
273	111	KEY_USAGE_TKDS	4	binary	Key usage. Bit Meaning when set 0 Data encryption allowed. 1 Data decryption allowed. 2 Key derivation allowed. 3 Sign allowed where signature is appendix. 4 Verify allowed where signature is appendix. 5 Sign allowed where data is recovered from signature. 6 Verify allowed where data is recovered from signature. 7 Key wrapping allowed. 8 Key unwrapping allowed. 9 Key usage must be FIPS-compliant. 10-31 Reserved.
274	112	KEY_EC_CURVE	1	binary	ECC curve type. X'01' Prime curve. X'02' Brainpool curve.
275	113	START_TOD	16	binary	Start time of the interval in STCKE format.

Table 153. Subtype 46 PKCS#11 key usage event (continued)					
Tag value		Name	Length	Format	Description
Dec	Hex				
276	114	END_TOD	16	binary	End time of the interval in STCKE format.
277	115	USG_COUNT	4	binary	Number of usages accounted for in this record.
279	117	FIPS_INFO	4	binary	<p>FIPS information related to the event.</p> <p>Bit</p> <p>Meaning when set</p> <p>0 FIPSMODE(YES) in effect.</p> <p>1 FIPSMODE(COMPAT) in effect.</p> <p>2 Request was evaluated for FIPS-compliance due to system settings. (Either FIPSMODE(YES) is in effect or FIPSMODE(COMPAT) is in effect, but the request was not exempt from FIPS-compliance.)</p> <p>3 Request was evaluated for FIPS-compliance at user request. (Either the object involved had the FIPS compliance flag on or FIPS-compliance was requested via a parameter on the service call.)</p> <p>4 Request passed FIPS evaluation.</p> <p>5-31 Reserved.</p>

The following tags may be present in the end user audit section:

- X500_IDN
- X500_SDN
- IDID_USRI
- IDID_USRF
- IDID_REG
- USRI

See [“Audit header and audit section” on page 342](#) for more details.

Subtype 47

PKCS#11 no key usage event

This subtype consists of a number of tag-length-value (TLV) triplets. The following triplets may be contained in the record. The specific set of triplets is dependent on the type of event and the information that is available.

Table 154. Subtype 47 PKCS#11 no key usage event					
Tag value		Name	Length	Format	Description
Dec	Hex				
262	106	SERVICE	8	EBCDIC	The service associated with the event. Service is either CSF1PPRF or CSF1POWH.
275	113	START_TOD	16	binary	Start time of the interval in STCKE format.
276	114	END_TOD	16	binary	End time of the interval in STCKE format.
277	115	USG_COUNT	4	binary	Number of usages accounted for in this record.

Table 154. Subtype 47 PKCS#11 no key usage event (continued)					
Tag value		Name	Length	Format	Description
Dec	Hex				
279	117	FIPS_INFO	4	binary	FIPS information related to the event. Bit Meaning when set 0 FIPSMODE(YES) in effect. 1 FIPSMODE(COMPAT) in effect. 2 Usage was evaluated for FIPS-compliance due to system settings. (Either FIPSMODE(YES) is in effect or FIPSMODE(COMPAT) is in effect, but usage was not exempt from FIPS-compliance.) 3 Usage was evaluated for FIPS-compliance at user request. (Either the object used had the FIPS compliance flag on or FIPS-compliance was requested via a parameter on the service call.) 4 Usage passed FIPS evaluation. 5-31 Reserved.

The following tags may be present in the end user audit section:

- X500_IDN
- X500_SDN
- IDID_USRI
- IDID_USRF
- IDID_REG
- USRI

See [“Audit header and audit section” on page 342](#) for more details.

Appendix C. CICS-ICSF Attachment Facility

The purpose of the CICS-ICSF Attachment Facility is to enhance the performance of CICS transactions in the same region as a transaction using long-running ICSF services such as the PKA services and CKDS or PKDS update services.

Without the CICS-ICSF Attachment Facility, the application that requests a long-running ICSF service is placed into an OS WAIT. With the CICS-ICSF Attachment Facility, a long running service is transferred to an L8, and the CICS application is placed into a CICS WAIT, rather than an OS WAIT, for the duration of the operation.

Note: The CICS-ICSF Attachment Facility can only be used by 31-bit assembler stub functions. The CICS-ICSF Attachment Facility cannot be used when invoking ICSF APIs with C linkage or 64-bit assembler stub functions. See *Combining C or C++ and Assembler in z/OS XL C/C++ Programming Guide* for information on invoking the 31-bit assembler stubs from a C/C++ program.

Installing the CICS-ICSF Attachment Facility

Before you can use the CICS-ICSF Attachment Facility, the ICSF system programmer, or the CICS administrator needs to install it. This involves:

- Relinking the ICSF enabling routine, CSFATREN, and the ICSF TRUE, CSFATRUE, if ICSF was previously installed in an environment without the CICS-ICSF Attachment Facility
- Installing the proper load libraries in the PROC used to start CICS
- Updating the CICS System Definitions (CSD) data set to define the programs to CICS
- Enabling these programs

For information about CICS TRUE programs, refer to *CICS Transaction Server for z/OS, Version 5 Release 1* (www.ibm.com/support/knowledgecenter/SSGMCP_5.2.0/com.ibm.cics.ts.devtrial.doc/topics/devtrial.htm).

Steps for installing the CICS-ICSF attachment facility

1. If ICSF was previously installed in an environment without the CICS-ICSF Attachment Facility (i.e., without being linked with the CICS SDFHLOAD data set), the ICSF system programmer will need to relink the ICSF TRUE, CSFATRUE, and the ICSF enabling routine, CSFATREN. This would be the case if, for example, (a) the DDDEF entries for ICSF do not have the SDFHLOAD DDDEF pointing to the CICS SDFHLOAD data set but instead have it pointing to an empty data set, or (b) z/OS (and hence ICSF) was installed using a ServerPac.

To relink the ICSF modules, first manually update the ICSF DDDEF for SDFHLOAD to point to the CICS SDFHLOAD data set. (Refer to ICSF sample CSFDDDEF shipped in SAMPLIB.) Then submit a job to relink the ICSF modules. This is an example of job control language for the relink.

```
//STEP01      EXEC PGM=IEWL,
//  PARM='LIST,XREF,LET,DCBS,AMODE(31),RMODE(24)'
//SYSLMOD DD DISP=SHR,DSN=yyy.SCSFSTUB (the ICSF load library)
//SYSLIB DD DISP=SHR,DSN=xxxxxx.SDFHLOAD
//SDFHLOAD DD DISP=SHR,DSN=xxxxxx.SDFHLOAD
//SCSFMOD0 DD DISP=SHR,DSN=yyy.SCSFMOD0 (the ICSF load library)
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(10,10))
//SYSPRINT DD SYSOUT=*
//SYSLIN DD *
            INCLUDE SDFHLOAD(DFHEAI)
            REPLACE CSFDHEAI(DFHEAI),CSF0EAI
            INCLUDE SCSFMOD0(CSFATREN)
            ENTRY DFHEAI
            NAME CSFATREN(R)
            INCLUDE SDFHLOAD(DFHEAI)
            REPLACE CSFDHEAI(DFHEAI),CSF0EAI
            INCLUDE SCSFMOD0(CSFATRUE)
```

```

        ENTRY DFHEAI
        NAME CSFATRE(R)
/*

```

2. Include the ICSF load module data set in the CICS startup job control language as shown in this example.

```

//DFHRPL DD DISP=SHR,DSN=xxxxx.SDFHLOAD
//      DD DISP=SHR,DSN=yyy.SCSFSTUB (ICSF callable service stubs)
//      DD DISP=SHR,DSN=yyy.SIEALNKE (ICSF shared libraries)
//      DD ...
...
//SYSIN DD DISP=SHR,DSN=xxxxx.SYSIN(DFH$SIPx)
...

```

In the previous sample code, DFH\$SIPx includes the entry:

```

PLTPI=yy,

```

3. Customize the Program Load Table (PLT), to include the ICSF enabling routine CSFATREN in second stage initialization.

This is an example input deck for compiling a PLT for automatic enablement of the CICS-ICSF link. This is ASM code. Assemble it with the CICS macro library, but **without** the CICS translator.

```

//SYSIN DD *
*
* List of programs to be executed sequentially during system
* initialization. Required system initialization parm: PLTPI=yy
* DFHPLTCS should be defined in the CSD by CEDA or DFHCSDUP job
*
DFHPLT TYPE=INITIAL,SUFFIX=yy
*
* ----- Second stage of initialization -----
*
DFHPLT TYPE=ENTRY,PROGRAM=CSFATREN (Run enable of CSFATRE)
*
* ----- Delimiter between Stages 2 and 3 -----
*
DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
*
* ----- Third stage of initialization -----
* (none)
*
DFHPLT TYPE=FINAL
END
/*

```

The previous code is an example only. Your CICS administrator can use it as a guide in customizing the PLT. For more information about coding the PLT, refer to [CICS Transaction Server for z/OS, Version 5 Release 1 \(www.ibm.com/support/knowledgecenter/SSGMCP_5.2.0/com.ibm.cics.ts.devtrial.doc/topics/devtrial.htm\)](http://www.ibm.com/support/knowledgecenter/SSGMCP_5.2.0/com.ibm.cics.ts.devtrial.doc/topics/devtrial.htm).

4. Link edit the PLT with these controls:

```

INCLUDE OBJLIB(DFHPLTyy)
NAME DFHPLTyy(R)

```

5. The CICS administrator should customize the system CSD to include:

- CSFATRE
- CSFATREN
- A PLT to indicate that initialization is to call CSFATREN to enable the ICSF TRUE, CSFATRE

This is an example of the job control language and input. In this example, xxxxx represents the local CICS prefix, and zzzzzzz represents the PLT entry that was compiled previously.

```

//UPDATE JOB ...
//*- - - - -
//DEFINES EXEC PGM=DFHCSDUP,REGION=2M
//STEPLIB DD DISP=SHR,DSN=xxxxxx.SDFHLOAD
//      DD DISP=SHR,DSN=zzzzzzzz

```

```
//DFHCSD DD DISP=SHR,DSN=xxxxxx.DFHCSD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*
DEFINE PROGRAM(CSFATREN) GROUP(ICSF)
                        DESCRIPTION(TRUE enablement routine)
                        LANGUAGE(ASSEMBLER)
*
DEFINE PROGRAM(CSFATRUE) GROUP(ICSF)
                        DESCRIPTION(ICSF interface TRUE)
                        LANGUAGE(ASSEMBLER)
                        CONCURRENCY(THREADSAFE)
                        API(OPENAPI)
*
DEFINE PROGRAM(DFHPLTy) GROUP(ICSF)
                        DESCRIPTION(PLT Program Init for CSFATRUE)
                        LANGUAGE(ASSEMBLER)
```

The PLT in the example runs the program CSFATREN during CICS initialization. CSFATREN automatically enables the ICSF TRUE, CSFATRUE. If CICS is already started, use a CICS Command Level Interpreter Transaction (CECI) to enable CSFATRUE. To do this, go into CECI and issue this statement:

```
ENABLE PROGRAM('CSFATRUE') TALENGTH(250) LINKEDITMODE START
```

You can also do this in a single step with this statement:

```
CECI ENABLE PROGRAM('CSFATRUE') TALENGTH(250) LINKEDITMODE START
```

6. If you have any existing CICS applications which invoke any of the ICSF services in the Wait List, then these applications must be re-linked with the current ICSF stubs.

Implementing the CICS wait list

The CICS Wait List can be implemented by means of a customer modifiable data set, pointed to by the Installation Options Data Set (WAITLIST parameter). The default WAITLIST includes all services which can complete asynchronously (for example, those services which perform I/O to a key data set and those services which are routed to a cryptographic processor). If the option is not specified, the default CICS Wait List will be utilized by ICSF when a CICS application invokes an ICSF callable service. If WAITLIST is specified, the data set specified by this parameter will be used to determine the names of the services to be placed on the CICS Wait List. A sample data set is provided by ICSF via member CSFWTL01 of SYS1.SAMPLIB. The sample data set contains the same entries as the default ICSF CICS Wait List -- for example, the data set contains the names of all ICSF callable services which, by default, will be driven through the CICS TRUE.

The WAITLIST option should be added to the Installation Options data set under these conditions.

- CICS customers who want to use the default CICS Wait List shipped with ICSF do not need to specify a WAITLIST keyword. If you have any existing CICS applications which invoke any of the ICSF services in the Wait List, then these applications must be re-linked with the current ICSF stubs.
- CICS customers who do not want to make use of CICS TRUE must either not enable the TRUE or specify a WAITLIST keyword and point to an empty wait list data set or you can specify WAITLIST(DUMMY) in the Installation Options data set.
- CICS customers who wish to modify the ICSF default CICS Wait List should modify the sample Wait List data set supplied in member CSFWTL01 of SYS1.SAMPLIB. The WAITLIST keyword in the Installation Options Data Set should be set to point to this data set. If you have any existing CICS applications which invoke any of the ICSF services in the Wait List, then these applications must be re-linked with the current ICSF stubs.

To ensure maximum performance, any existing CICS applications which invoke any of the ICSF services in the Wait List that were linked with ICSF stubs prior to HCR7770 should be re-linked with the current ICSF stubs.

If you already have the CICS-ICSF Attachment facility installed, there are a number of callable services which may potentially be routed to a coprocessor or may perform other asynchronous processing. If you

have a modified CICS Wait List, you should ensure that the wait list data set includes all such services, and any CICS applications which invoke any of these services are re-linked with the current ICSF stubs. As a model, you can use the default CICS Wait List that is shipped with ICSF which includes all services which have an asynchronous interface to ICSF or you can use a sample Wait List data set that is also shipped with ICSF. The sample CICS Wait List data set is contained in member CSFWTL01 of SYS1.SAMPLIB. The sample data set contains the same entries as the default ICSF CICS Wait List. If you have an application which invokes a UDX while running under CICS, then the name of the UDX generic service should be added to the CICS Wait List.

If you use a CICS Wait List data set, you need to identify the data set to ICSF through the WAITLIST(data_set_name) option in the ICSF Installation Options data set. The data set can be a member of a PARMLIB, a member of a partitioned data set, or a sequential data set. The data set should be allocated on a permanently resident volume and should adhere to:

- The format of each record in the data set must be fixed length or fixed block length.
- A physical line in the data set must be a LRECL of 80 characters long. The system ignores any characters in positions 73 to 80 of the line.
- You can delimit comments by "/" and "*" and include them anywhere in the text. A comment cannot span physical records.
- Only one service may be specified on a logical line.

Note: You can use the WAITLIST(DUMMY) parameter to specify a null CICS Wait List data set, or you can disable the CICS TRUE if you do not want to utilize the CICS TRUE. See [“Parameters in the installation options data set” on page 30](#) for additional information.

Appendix D. Helpful hints for ICSF first time startup

The purpose of this topic is to provide some helpful hints and resolutions for the problems that you may encounter when starting ICSF for the first time.

See [Appendix F, “Systems without Cryptographic features,” on page 391](#) if you're running in this environment.

Checklist for first-time startup of ICSF

This is a checklist for the first-time startup of ICSF.

Note: ALL crypto coprocessors cards must be loaded with the same level of code. Otherwise, unpredictable results can occur.

Step 1. Hardware setup

Note: The CP Assist for Cryptographic Functions feature is required for selection of the coprocessor in the activation profiles.

Process

LIC installed for CP Assist for Cryptographic Functions

Note: If using TKE, you must Permit each coprocessor for TKE Commands.

Responsible

CE or Client Operator Representative

Where

Support Element

Verify

Via CPC details

- CP Assist for Cryptographic Functions is 'Installed'
- CP Assist for Cryptographic Functions DES/TDES enablement (feature 3863) is 'Installed'

Via PCI Cryptographic Configuration Task

- Status for each coprocessor or accelerator is 'Configured'

Note: If using TKE, the status for each Coprocessor is "Permitted".

References

Support Element Operations Guide

Completed

Step 2. LPAR activation profiles

Process

PCI Crypto Page Setup

Responsible

CE or Client Operator Representative

Where

Support Element

Verify

Control Domain Index

Usage Domain Index

PCI Cryptographic Candidate List includes all CCA Crypto Express coprocessors and accelerators that CAN be online

PCI Cryptographic Online List includes all CCA Crypto Express coprocessors and accelerators that WILL be online when activation is complete (Selections in the Online List MUST be selected in the Candidate List)

References

Support Element Operations Guide

z/OS Cryptographic Services ICSF TKE Workstation User's Guide (LPAR Considerations)

zSeries PR/SM Planning Guide

Completed

Note: If TKE is to be used, ALL cryptographic coprocessors that you want TKE to be able to control MUST be defined in the Online and Candidate Lists. Also, the Usage Domain for the TKE Host LPAR **MUST** be unique. While the same domain may be used by other LPARs as long as these LPARs do not share any of the same cards, the TKE Host domain must have access to all the cards so that prohibits any other LPAR from using the same domain.

Step 3. ICSF setup

Process

Install and Customize ICSF

Responsible

System Programmer and ICSF Administrator

Where

TSO and ISPF Panels

Verify

Customize SYS1.PARMLIB

- Add CSF.SCSFMODE and CSF.SCSFSTUB to the LNKLIST concatenation
- Update PROGxx to APF authorize CSF.SCSFMODE and CSF.SCSFSTUB
- Update IKJTSoxx for ICSF by adding CSFDAUTH and CSFDPKDS to the AUTHPGM and AUTHTSF parameter lists. To change the active IKJTSoxx member of SYS1.PARMLIB, use the PARMLIB UPDATE command.

CKDS and PKDS created

ICSF Startup Procedure created

Installation Options Dataset created

- The DOMAIN parameter in the installation options data set is optional. It is required if more than one domain is specified as the usage domain on the PR/SM panels or if running in native mode.
- CKDS and PKDS names specified
- COMPAT(NO)

Access provided to the ICSF panels

References

Chapter 2, "Installation, initialization, and customization," on page 9

Completed

Step 4. TKE setup

If you are not using TKE, proceed to the next step.

Process

Initialize the TKE Workstation.

Configure TCP/IP on the Host and the TKE Workstation.

Perform passphrase or smart card setup.

Setup the TKE Host Transaction Program:

- Create JCL to start the TKE Host Transaction Program.
- RACF Security Setup.
- Start the TKE Host Transaction Program.

Responsible

Network Programmer, System Programmer and TKE Administrator.

Where

ISPF Panels, TKE Workstation.

Verify

CSFTTKE is authorized in the AUTHCMD list of IKJTSoxx in SYS1.PARMLIB.

TKE Host Transaction Program (CSFTTCP) is defined in the RACF STARTED class (Note: If your installation has a Generic Userid associated to all started procedures, this is not necessary).

CSFTTKE profile is defined in the RACF FACILITY and RACF APPL classes.

The userid associated with the TKE Host Transaction Program (CSFTTCP) must be authorized to the CSFCRC, CSFKIM, CSFKRC, CSFKRD, CSFKRR, CSFKRW, CSFKYT, CSFKYT2, CSFPCI, CSFPKRC, CSFPKRW, and CSFPKI profiles in the CSFSERV class.

References

[*z/OS Cryptographic Services ICSF TKE Workstation User's Guide*](#) (See Topics: TKE Workstation Setup and Customization and TKE TCP/IP and Host Considerations.)

Completed

Step 5. ICSF startup

Process

Start ICSF

Responsible

Client Operator Representative or System Programmer

Where

Operator Console

References

[Chapter 2, "Installation, initialization, and customization," on page 9](#)

Completed

Step 6. Loading master keys and initializing the CKDS through ICSF panels

Note: When defining a master key by specifying master key parts, make sure that the key parts are recorded and saved in a secure location. When you are entering the key parts for the first time, be aware that you might need to reenter these same key values at a later date to restore master key values that have been cleared. If defining a master key by using a pass phrase, realize that the same pass phrase always produces the same master key values and is therefore as critical and sensitive as the master key values themselves. Make sure that you save the pass phrase so that you can later reenter it if needed. Because of the sensitive nature of the pass phrase, make sure that you secure it in a safe place.

If you are using TKE, proceed to the next step.

Process

Passphrase Initialization to load and SET master keys and initialize CKDS and PKDS

- OR -

Clear Master Key Entry

Note: Using the Coprocessor Management panel, the master keys can be loaded into all the coprocessors at the same time.

- Load DES New Master Key (optional)
- Load RSA New Master Key (optional)
- Load New AES master key if running on z10 or newer servers with a CCA Crypto Express coprocessor and the Nov. 2008 or newer licensed internal code. (optional)
- Load New ECC master key if running on z10 or newer servers with a CCA Crypto Express coprocessor and the Sept. 2011 or newer licensed internal code. (optional)
- Initialize CKDS
- Initialize the PKDS
- Enable PKA Callable Services control

Note: The PKA Callable Services control is disabled if the system has a CEX3C or newer with the Sept. 2011 or newer licensed internal code.

Responsible

ICSF Administrator and Key Officers

Where

ICSF Panels

Verify

In System Log (Systems with CCA Crypto Express coprocessors and accelerators):

```
CSFM608I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM608I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES
SUCCESSFUL.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS5 ACCELERATOR 5Axx, SERIAL NUMBER N/
A.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS5 COPROCESSOR 5Czz, SERIAL NUMBER
ssssssss.
CSFM133I THERE ARE NO ACTIVE PKCS11 COPROCESSORS.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CSFCKDS IS NOT
INITIALIZED.
CSFM101E PKA KEY DATA SET, CSF.CSFPKDS IS NOT
INITIALIZED.
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE
AVAILABLE.
CSFM001I ICSF INITIALIZATION COMPLETE
```

Message CSFM111I is issued for each active Crypto Express coprocessor and accelerator.

Message CSFM122I is not issued when your system has any CEX3C coprocessors (with the Sept. 2011 or later LIC) online. The PKA callable services control will not be active. The availability of RSA callable services depend on the status of the RSA master key. CSFM130I is issued when the RSA master key is active and RSA callable services are available.

In System Log (without coprocessors and accelerators):

```
CSFM608I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM608I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES SUCCESSFUL.
CSFM505I CRYPTOGRAPHY - THERE ARE NO ACTIVE CRYPTOGRAPHIC COPROCESSORS.
CSFM133I THERE ARE NO ACTIVE PKCS11 COPROCESSORS.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CSFCKDS IS NOT INITIALIZED.
CSFM101E PKA KEY DATA SET, CSF.CSFPKDS IS NOT INITIALIZED.
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
```



```
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.  
CSFM001I ICSF INITIALIZATION COMPLETE
```

References

For information on using the Pass Phrase Initialization Utility and managing master keys, refer to [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Completed

Step 7. Customizing TKE and loading master keys

If you are not using TKE, proceed to the next step.

Process

TKE Administrator's and Key Officers

- Define host IDs
- Define roles
- Define coprocessor authorities
- Load new DES master key (optional)
- Load new RSA master key (optional)
- Load new AES master key (optional)
- Load new ECC master key if running on z10 or newer servers with a CCA Crypto Express coprocessor and the Sept. 2011 or later licensed internal code. (optional)

Note: If you have more than one coprocessor, repeat the process for each, unless groups have been defined.

Responsible

ICSF Administrator

- Initialize CKDS and SET the DES and AES (if applicable) master keys
- Initialize PKDS and SET the RSA and ECC (if applicable) master keys
- Enable PKA Callable Services control

Note: The PKA Callable Services control is disabled if the system has a CEX3C or newer with the Sept. 2011 or newer licensed internal code.

Where

TKE Workstation and ICSF Panels

Verify

In System Log (Systems with Crypto Express coprocessors and accelerators):

```
CSFM608I A CKDS KEY STORE POLICY IS NOT DEFINED.  
CSFM608I A PKDS KEY STORE POLICY IS NOT DEFINED.  
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.  
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.  
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.  
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.  
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.  
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES  
SUCCESSFUL.  
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS5 ACCELERATOR 5Axx, SERIAL NUMBER N/  
A.  
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS5 COPROCESSOR 5Czz, SERIAL NUMBER  
ssssssss.  
CSFM133I THERE ARE NO ACTIVE PKCS11 COPROCESSORS.  
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CSFCKDS IS NOT  
INITIALIZED.  
CSFM101E PKA KEY DATA SET, CSF.CSFPKDS IS NOT  
INITIALIZED.  
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE  
AVAILABLE.  
CSFM001I ICSF INITIALIZATION COMPLETE
```

Message CSFM111I is issued for each active Crypto Express coprocessors and accelerators.

Message CSFM122I will not be issued when your system has any CEX3C or newer coprocessors (with the Sept. 2011 or later LIC) online. The PKA callable services control will not be active. The availability of RSA callable services will depend on the status of the RSA master key. CSFM130I is issued when the RSA master key is active and RSA callable services are available.

In System Log (Systems without coprocessors or accelerators):

```
CSFM608I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM608I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES SUCCESSFUL.
CSFM505I CRYPTOGRAPHY - THERE ARE NO ACTIVE CRYPTOGRAPHIC COPROCESSORS.
CSFM131E CRYPTOGRAPHY - DES SERVICES ARE NOT AVAILABLE.
CSFM131E CRYPTOGRAPHY - RSA SERVICES ARE NOT AVAILABLE.
CSFM131E CRYPTOGRAPHY - ECC SERVICES ARE NOT AVAILABLE.
CSFM131E CRYPTOGRAPHY - AES SERVICES ARE NOT AVAILABLE.
CSFM133I THERE ARE NO ACTIVE PKCS11 COPROCESSORS.
CSFM131E CRYPTOGRAPHY - SECURE KEY PKCS11 SERVICES ARE NOT AVAILABLE.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CSFCKDS IS NOT INITIALIZED.
CSFM101E PKA KEY DATA SET, CSF.CSFPKDS IS NOT INITIALIZED.
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
CSFM001I ICSF INITIALIZATION COMPLETE
```

References

For information on managing master keys, refer to [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Completed

Step 8. CICS-ICSF Attachment Facility setup

If you are not using CICS, proceed to the next topic.

Process

Follow the instructions in [Appendix C, “CICS-ICSF Attachment Facility,”](#) on page 377 if desired.

Responsible

System Programmer

Where

Sample Jobs

References

[Appendix C, “CICS-ICSF Attachment Facility,”](#) on page 377

Completed

Step 9. Complete ICSF initialization

See [“Steps for initializing ICSF”](#) on page 28

Responsible

System Programmer

Where

Operator Console

Completed

Commonly encountered ICSF first time setup/initialization messages

These ICSF messages are commonly encountered during initialization and first time startup of ICSF.

- **CSFM124I MASTER KEY** *mk* ON *coprocessor-name cii*, **SERIAL NUMBER** *nnnnnnnn*, **NOT INITIALIZED** - The cryptographic coprocessor does not have the master key. When a master key is not set, then the cryptographic coprocessor may not be used for operations with the master key until the system administrator has provided the master key. This may be a normal situation for your installation. Have the system administrator enter the correct master key if appropriate.
- **CSFM410E ERROR IN OPTIONS DATA SET** - ICSF could not interpret the options data set. Check the CSF job output for diagnostic messages.

Appendix E. Using AMS REPRO encryption

This appendix provides information on using IDCAMS REPRO ENCIPHER and DECIPHER options with ICSF.

Steps for setting up ICSF

Perform these tasks to use the ENCIPHER and DECIPHER parameters with ICSF:

1. Define the key value that is used to encrypt and decrypt the data key. To define the key value, use one of these ICSF key administrative options:
 - Trusted Key Entry (TKE) workstation. For information about how to define the key value by using the TKE workstation, see [z/OS Cryptographic Services ICSF TKE Workstation User's Guide](#).
 - Key generator utility program (KGUP). Use the KGUP panel ICSF - Create ADD, UPDATE, or DELETE Key Statement to define the key value. For more information about how to use KGUP panels, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

Be aware of the following restrictions:

- The length of the data encryption key is limited to 8 bytes, or 56-bit DES. Triple DES support is not available.
- Key labels are limited to 8 characters because of the fixed size of REPRO storage areas.
- The REPRO command's encryption algorithm variables are not documented, so you cannot use them to write decryption applications on another system. Therefore, cross-platform exchange is not possible.

2. Refresh ICSF's cryptographic key data set (CKDS) so that the key value can be used by REPRO.
3. Ensure that ICSF can support PCF macro calls by specifying COMPAT(YES) in the ICSF installation options. For more information about how to specify ICSF installation options, see [Chapter 2, "Installation, initialization, and customization,"](#) on page 9.

If you had to change the ICSF installation options, you must restart ICSF.

4. Run the REPRO ENCIPHER or DECIPHER job.

Restrictions: The REPRO command's encryption algorithm variables are not documented, so you cannot use them to write decryption applications on another system. Therefore, cross-platform exchange is not possible.

Recommendation: Do not specify the REPRO parameter PRIVATEKEY because it exposes the clear data key value. Instead, specify either EXTERNALKEY or INTERNALKEY, and STOREDATAKEY.

Appendix F. Systems without Cryptographic features

This topic describes the processing of ICSF without a cryptographic coprocessor or accelerator.

Applications and programs

Applications requiring secure cryptography using encrypted keys will not be able to execute without a cryptographic coprocessor or accelerator. All cryptographic keys must be clear keys.

These applications and programs are not supported:

- Access Method Services Cryptographic option.
- CICS attachment facility.
- CKDS Conversion program.
- CSFEUTIL program for CKDS reencipher, refresh, and change master key functions.
- CSFPUTIL program for PKDS reencipher and refresh functions.
- Distributed Key Management System (DKMS).
- Key Generation Utility Program (KGUP) – Clear key can be generated.
- PCF applications.
- UDX (User Defined Extension) support.
- VTAM Session Level Encryption.
- If only the CPACF feature is installed, you will not be able to:
 1. Set master keys.
 2. Initialize the PKDS.
 3. Store keys in the PKDS.

Callable services

These services are available when there are no cryptographic coprocessors or accelerators:

- Character/Nibble Conversion (CSNBXBC and CSNBXCB)
- Code Conversion (CSNBXEA and CSNBXAE)
- Control Vector Generate (CSNBCVG)
- Decode (CSNBDCO)
- Encode (CSNBECO)
- Field level decipher (CSNBFLD)
- Field level encipher (CSNBFLE)
- ICSF Query Facility (CSFIQF and CSFIQF6) - The only rule available without a coprocessor is ICSFSTAT.
- ICSF Query Facility2 (CSFIQF2 and CSFIQF26)
- ICSF Query Algorithm (CSFIQA)
- MDC Generate (CSNBMDG and CSNBMDG1)
- One-Way Hash Generate (CSNBOWH and CSNBOWH1)
- PKA Key Token Build (CSNDPKB)
- PKA Public Key Extract (CSNDPKX)
- PKCS #11 Derive multiple keys (CSFPDMK)

- PKCS #11 Derive key (CSFPDVK)
- PKCS #11 Get attribute value (CSFPGAV)
- PKCS #11 Generate key pair (CSFPGKP)
- PKCS #11 Generate secret key (CSFPGSK)
- PKCS #11 Generate MAC (CSFPHMG)
- PKCS #11 Verify MAC (CSFPHMV)
- PKCS #11 One-way hash generate (CSFPOWH)
- PKCS #11 Private key sign (CSFPPKS)
- PKCS #11 Public key verify (CSFPPKV)
- PKCS #11 Pseudo-random function (CSFPPRF)
- PKCS #11 Set attribute value (CSFPSAV)
- PKCS #11 Secret key decrypt (CSFPSKD)
- PKCS #11 Secret key encrypt (CSFPSKE)
- PKCS #11 Token record create (CSFPTRC)
- PKCS #11 Token record delete (CSFPTRD)
- PKCS #11 Token record list (CSFPTRL)
- PKCS #11 Unwrap key (CSFPUWK)
- PKCS #11 Wrap key (CSFPWPK)
- Random Number Generate (CSNBRNG) and Random Number Generate Long (CSNBRNGL)
- Symmetric Key Decipher (CSNBSYD and CSNBSYD1) - Only clear keys are supported.
- Symmetric Key Encipher (CSNBSYE and CSNBSYE1) - Only clear keys are supported.
- Symmetric MAC Generate (CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1)
- Symmetric MAC Verify (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1)
- X9.9 Data Editing (CSNB9ED)

These services are available when there are no cryptographic coprocessors and there are accelerators:

- Digital Signature Verify (CSNDDSV)
- PKA Decrypt (CSNDPKD)
- PKA Encrypt (CSNDPKE) ZERO-PAD formatting only

Note:

1. Installation defined callable services are supported only if you're using clear keys and using one of the supported callable services.
2. If running without an active PKCS #11 Cryptographic coprocessor, the PKCS #11 callable services are limited to clear keys only.

ICSF setup and initialization

If starting ICSF without any cryptographic features:

```
CSFM608I A CKDS KEY STORE POLICY IS DEFINED.
CSFM608I A PKDS KEY STORE POLICY IS DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES SUCCESSFUL.
CSFM505I CRYPTOGRAPHY - THERE ARE NO ACTIVE CRYPTOGRAPHIC COPROCESSORS.
CSFM133I THERE ARE NO ACTIVE PKCS11 COPROCESSORS.
```



```
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CKDS IS NOT INITIALIZED.  
CSFM101E PKA KEY DATA SET, CSF.PKDS IS NOT INITIALIZED.  
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.  
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.  
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.  
CSFM001I ICSF INITIALIZATION COMPLETE
```

If starting ICSF with a cryptographic accelerator:

```
CSFM608I A CKDS KEY STORE POLICY IS DEFINED.  
CSFM608I A PKDS KEY STORE POLICY IS DEFINED.  
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.  
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.  
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.  
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.  
CSFM654I KEY ARCHIVING USE CONTROL IS DISABLED.  
CSFM015I FIPS 140 SELF CHECKS FOR PKCS11 SERVICES SUCCESSFUL.  
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS4 ACCELERATOR 4A00,  
SERIAL NUMBER N/A.  
CSFM505I CRYPTOGRAPHY - THERE ARE NO ACTIVE CRYPTOGRAPHIC COPROCESSORS.  
CSFM133I THERE ARE NO ACTIVE PKCS11 COPROCESSORS.  
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CKDS IS NOT INITIALIZED.  
CSFM101E PKA KEY DATA SET, CSF.PKDS IS NOT INITIALIZED.  
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.  
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.  
CSFM001I ICSF INITIALIZATION COMPLETE
```

Secure Sockets Layer (SSL)

System SSL applications are supported. SSL defines methods for data encryption, server authentication, message integrity, and client authentication for a TCP/IP connection. Security is provided on the link and callable services have been enhanced for DES, TDES and SHA-1 services.

TKE workstation

The Trusted Key Entry (TKE) workstation is not available with this hardware configuration.

Appendix G. Accessibility

Accessible publications for this product are offered through [IBM Knowledge Center \(www.ibm.com/support/knowledgecenter/SSLTBW/welcome\)](http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

If you experience difficulty with the accessibility of any z/OS information, send a detailed email message to mhvrdfs@us.ibm.com.

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- [*z/OS TSO/E Primer*](#)
- [*z/OS TSO/E User's Guide*](#)
- [*z/OS ISPF User's Guide Vol I*](#)

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element

The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element

The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE (KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

*** indicates an optional syntax element that is repeatable**

The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.

3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services that are offered in the USA 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 grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America*

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.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for the Knowledge Centers. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites 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:

*IBM Corporation
Site Counsel
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 document 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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, JES2, JES3, and MVS™, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS \(www.ibm.com/software/support/systemsz/lifecycle\)](http://www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com)[®] 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 \(www.ibm.com/legal/copytrade.shtml\)](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.

Index

A

- abends [126](#)
- access control checking
 - udx [187](#)
- Access Method Services Cryptographic Option and ICSF [113](#)
- accessibility
 - contact IBM [395](#)
 - features [395](#)
- activity report
 - defining on a DD statement [209](#)
 - description [210](#)
- adding a regional cryptographic server
 - SETICSF command [112](#)
- addressing mode
 - no restrictions on ICSF's caller [113](#)
- AMS DEFINE CLUSTER command [13](#), [16](#), [18](#)
- AMS IMPORT/EXPORT commands [13](#), [16](#), [18](#)
- AMS REPRO command [13](#), [16](#), [18](#)
- AMS REPRO encryption [199](#)
- assistive technologies [395](#)

B

- BEGIN installation option [35](#)

C

- callable services exit
 - CSF_SERVICE_EXIT [157](#)
- changing coprocessor or accelerator status
 - SETICSF command [112](#)
- changing parameters in installation options data set
 - specifying option parameters and values [35](#)
- changing the master key in compatibility or coexistence mode [201](#)
- CHECKAUTH installation option [36](#)
- choosing compatibility modes during migration [201](#)
- CICS
 - WAITLIST installation option [51](#)
- CICS wait list [83](#)
- CICS-ICSF Attachment Facility
 - installing [377](#)
- CIPHER macro
 - SVC description [7](#)
- CKDS
 - create [12](#)
 - primary space required [12](#)
 - secondary space required [13](#)
- CKDS (cryptographic key data set)
 - conversion from PCF CKDS to ICSF CKDS [202](#)
 - creating [13](#)
 - description [5](#)
 - header record format [215–217](#)
 - record format [217–219](#), [286](#)
- CKDS entry retrieval installation exit

- CKDS entry retrieval installation exit (*continued*)
 - environment [159](#)
 - input [160](#)
 - installing [160](#)
 - purpose and use [159](#)
 - return codes [161](#)
- CKDS refresh
 - SMF record type 82 [116](#)
- CKDSN installation option [36](#)
- CKTAUTH [36](#)
- coexistence mode
 - changing the master key [200](#), [201](#)
 - description [199](#), [200](#)
- coexistence, definition [54](#)
- command
 - syntax diagrams [87](#)
- command syntax notation [87](#)
- common record format (KDSR)
 - format [250](#)
- COMPAT installation option [36](#), [199](#)
- compatibility mode
 - and the Access Method Services Cryptographic Option [113](#)
 - changing the master key [199](#), [201](#)
 - description [199](#)
- COMPENC installation option [37](#)
- component trace [125](#)
- configure on/off cryptographic coprocessors [107](#)
- contact
 - z/OS [395](#)
- controlling access to CSFDUTIL [123](#)
- controlling access to secure tokens [124](#)
- controlling access to the callable services [123](#)
- controlling access to the cryptographic keys [124](#)
- controlling access to the key generator utility program [123](#)
- controlling the program environment [123](#)
- conversion program
 - activity report [210](#)
 - bypassing entries [205](#)
 - converting key types [207](#)
 - data sets [209](#)
 - including information in a key entry [206](#)
 - installation exit [203](#)
 - JCL for submitting [209](#)
 - override file [204](#)
 - running [209](#)
- conversion program installation exit
 - PCF
 - purpose and use [162](#)
 - return codes [164](#)
- converting a PCF CKDS [202](#)
- Converting to KDSR format [69](#)
- CP Assist for Cryptographic Functions
 - description [1](#)
- Creating an
 - creating an ICSF CTRACE Configuration Data Set [23](#)
 - Creating an ICSF CTRACE Configuration Data Set [23](#)

- creating the CKDS
 - allocating space for the CKDS [12](#)
 - reading the CKDS into storage [28](#)
 - using the AMS DEFINE CLUSTER command [13](#)
- creating the installation options data set
 - guidelines [21](#)
- creating the PKDS
 - allocating space for the PKDS [16](#)
- creating the startup procedure
 - specifying the installation options data set [25](#)
- creating the TKDS
 - allocating space for the TKDS [18](#)
- crypto education [xxi](#)
- Crypto Express2 Coprocessor
 - description [1](#)
- cryptographic communication vector table [329](#)
- cryptographic communication vector table extension [331](#)
- Cryptographic Coprocessor clear master key entry
 - SMF record type [82](#) [117](#)
- cryptographic coprocessor retained key create or delete
 - SMF record type [82](#) [117](#)
- cryptographic coprocessor timing
 - SMF record type [82](#) [118](#)
- cryptographic coprocessor TKE command request or reply
 - SMF record type [82](#) [117](#)
- cryptographic coprocessors
 - bringing offline [107](#)
 - bringing online [107](#)
 - disabling [107](#)
- csf [25](#)
- CSF_SERVICE_EXIT [157](#)
- CSFAPRPC processing routine [188](#)
- CSFCKDS exit [159](#)
- CSFCONVX exit [161](#)
- CSFESECI exit [168](#)
- CSFESECK exit [168](#)
- CSFESECS exit [168](#)
- CSFESECT exit [168](#)
- CSFEXIT1 exit [136](#)
- CSFEXIT2 exit [136](#)
- CSFEXIT3 exit [136](#)
- CSFEXIT4 exit [136](#)
- CSFEXIT5 exit [136](#)
- CSFKGUP exit [172](#)
- CSFPARM data set [25](#)
- CSFPRM00 [21](#)
- CSFSRRW exit [164](#)
- CSFVINP data set [210](#)
- CSFVNEW data set [210](#)
- CSFVOVR data set [210](#)
- CSFVRPT data set [210](#)
- CSFVSRC data set [210](#)
- CTRACE installation option [37](#)

D

- DEFAULTWRAP installation option [37](#)
- DEFINE CLUSTER command [13](#), [16](#), [18](#)
- defining conversion program data sets [209](#)
- DES external key token format [258](#)
- disabling cryptographic coprocessors [107](#)
- DISPLAY command
 - displaying cryptographic coprocessor status [112](#)
- Display ICSF command [90](#)

- displaying cryptographic coprocessor status
 - DISPLAY command [112](#)
- DOMAIN installation option [37](#)
- duplicate key tokens
 - SMF record type [82](#) [119](#)
- dynamic CKDS update
 - SMF record type [82](#) [117](#)
- dynamic PKDS update
 - SMF record type [82](#) [117](#)

E

- ECC token
 - associated data format for [314](#)
- EMK macro
 - SVC description [7](#)
- END installation option [38](#)
- ENF signals [129](#)
- event recording [114](#)
- exit
 - CKDS entry retrieval installation exit [132](#), [159](#)
 - description [131](#)
 - entry and return specifications [133](#)
 - identifier on ICSF [38](#)
 - invocation on ICSF [38–43](#)
 - key generator utility program installation exit [133](#), [172](#)
 - mainline installation exits [131](#), [136](#)
 - PCF conversion program installation exit [132](#), [161](#)
 - security installation exits [168](#)
 - service installation exits [132](#), [144](#)
 - single-record, read-write installation exit [132](#), [164](#)
- EXIT installation option [38](#)
- exit name table [141](#)
- external key token
 - DES [258](#)
 - PKA
 - RSA private [288](#)

F

- feedback [xxiii](#)
- FIPSMODE installation option [43](#)
- FMID
 - applicable z/OS releases [4](#)
 - hardware [4](#)
 - servers [4](#)
- formatting control blocks
 - using IPCS [126](#)
- functions not supported [83](#)

G

- GENKEY macro
 - SVC description [7](#)

H

- hash services [58](#)

I

- ICSF
 - dispatching priority [51](#), [113](#)

- ICSF (Integrated Cryptographic Service Facility)
 - CSFSMF82 mapping macro [339](#)
 - record type 82 [339](#)
- ICSF CTRACE Configuration Data Set [23](#)
- ICSF initialization
 - SMF record type 82 [116](#)
- ICSF installation options data set
 - deprecated parameters [57](#)
- ICSF migration actions [54](#)
- ICSF operator commands
 - Display ICSF [90](#)
 - SETICSF [95](#)
- icsf sysplex group
 - SMF record type 82 [119](#)
- ICSFMIG77A1_COPROCESSOR_ACTIVE [54](#)
- ICSFMIG77A1_TKDS_OBJECT [55](#)
- initializing ICSF
 - creating the CKDS [13](#)
 - creating the PKDS [16](#)
 - creating the TKDS [18](#)
 - creation of [13](#), [16](#), [18](#)
 - selecting ICSF startup options
 - creating the installation options data set [21](#)
 - creating the startup procedure [25](#)
 - starting ICSF [28](#)
- installation option keyword
 - CHECKAUTH [36](#)
 - CKDSN [36](#)
 - CKTAUTH [36](#)
 - COMPAT [36](#), [199](#)
 - COMPENC [37](#)
 - CTRACE [37](#)
 - DEFAULTWRAP [37](#)
 - DOMAIN [37](#)
 - EXIT [38](#)
 - FIPSMODE [43](#)
 - KEYAUTH [46](#)
 - MASTERKCVLEN [46](#)
 - MAXLEN [46](#)
 - MAXSESSOBJECTS [46](#)
 - PKDSCACHE [46](#)
 - PKDSN [46](#)
 - REASONCODES [47](#)
 - SERVICE [48](#)
 - SSM [48](#)
 - SYSPLEXCKDS [49](#)
 - SYSPLEXTKDS [50](#)
 - TKDSN [50](#)
 - UDX [50](#)
 - USERPARM [50](#)
 - WAITLIST [51](#)
- installation option parameter
 - BEGIN [35](#)
 - END [38](#)
- installation options
 - performance considerations [112](#)
- installation options data set
 - changing option parameters and values [35](#)
 - creating [21](#)
 - example [21](#)
 - specifying the installation options data set [25](#)
- installation steps [9](#)
- installation-defined service
 - access control checking [187](#)

- installation-defined service (*continued*)
 - defining [188](#)
 - description [185](#)
 - entry and exit code example [187](#)
 - executing [189](#)
 - link editing [187](#)
 - parameter checking [187](#)
 - writing [185](#)
- Integrity [317](#)
- internal key token
 - aes; [253](#)
 - DES [255](#)
 - PKA
 - RSA private [299](#), [301–304](#), [310](#), [314](#), [315](#)

K

- KDSR
 - format [250](#)
- key generator utility program exit parameter block [174–182](#)
- key generator utility program installation exit
 - calling points [172](#)
 - environment [173](#)
 - installing [173](#)
 - processing [173](#)
 - purpose and use [172](#)
 - return codes [182](#)
 - SET statement [182](#)
- key part entry
 - SMF record type 82 [116](#)
- key store policy
 - SMF record type 82 [119](#)
- key token
 - aes; internal [253](#)
 - DES
 - null [260](#)
 - DES internal [255](#)
 - PKA
 - null [287](#)
 - RSA 1024-bit private internal [301–304](#)
 - RSA 2048-bit Chinese remainder theorem private internal [308–310](#)
 - RSA private external [288](#)
 - RSA private internal [299](#), [310](#), [314](#), [315](#)
 - RSA public [287](#)
- KEYAUTH installation option [46](#)
- keyboard
 - navigation [395](#)
 - PF keys [395](#)
 - shortcut keys [395](#)

L

- link editing
 - callable services [187](#)

M

- mainline installation exit
 - environment [136](#)
 - exit parameter block [138](#)
 - input [137](#)
 - installing [137](#)

- mainline installation exit (*continued*)
 - parameters [139, 143](#)
 - purpose and use [136](#)
- mapping macro
 - CSFSMF82 (ICSF) [340](#)
- MASTERKCVLEN installation option [46](#)
- MAXLEN installation option [46](#)
- MAXSESSOBJECTS installation option [46](#)
- message recording [122](#)
- migrating from PCF [199](#)
- Migrating to the common record format (KDSR) key data set [69](#)
- migration
 - terminology [54](#)
- migration actions
 - Cryptographic Services [54](#)
- MODIFY command [87](#)
- modifying ICSF [87](#)

N

- navigation
 - keyboard [395](#)
- noncompatibility mode
 - description [199, 201](#)
- null key token
 - format [260, 287](#)

O

- object ion key (OPK) [325](#)
- operator commands
 - ICSF [90](#)
- OPK, object protection key [325](#)
- override file
 - defining on a DD statement [209](#)

P

- panels
 - accessing [26](#)
 - ICSF Coprocessor Management [107](#)
- parameter checking
 - callable services [187](#)
- PCF
 - application [199–201](#)
 - macro [199](#)
 - migration to ICSF [199](#)
- PCF conversion program installation exit
 - environment [162](#)
 - input [163](#)
 - installing [162](#)
 - purpose and use [162](#)
- PCI Cryptographic Coprocessor configuration
 - SMF record type [82 118](#)
- PCI X Cryptographic Coprocessor timing
 - SMF record type [82 118](#)
- performance
 - problems [51, 113](#)
- PKA key token
 - record format
 - RSA 1024-bit private internal [301–304](#)

- PKA key token (*continued*)
 - record format (*continued*)
 - RSA 2048-bit Chinese remainder theorem private internal [308–310](#)
 - RSA private external [288](#)
 - RSA private internal [299, 310, 314, 315](#)
 - RSA public [287](#)
- PKDS (public key data set)
 - creating [16](#)
 - description [6](#)
 - header record format [219](#)
 - record format [220, 221](#)
- PKDSCACHE installation option [46](#)
- PKDSN installation option [46](#)
- private external key token
 - RSA [288](#)
- private internal key token
 - RSA [299, 301–304, 310, 314, 315](#)
- public key data set
 - improving security and reliability for the PKDS [16](#)
- public key data set refresh
 - SMF record type [82 120](#)
- public key token
 - RSA [287](#)

R

- read-write exit parameter block [166, 167](#)
- REASONCODES installation option [47](#)
- record metadata [67](#)
- recording events [114](#)
- regional cryptographic server configuration
 - SMF record type [82 121](#)
- regional cryptographic servers
 - adding [108, 112](#)
 - adding and removing [108](#)
 - configuring ICSF to use TCP/IP [109, 110](#)
 - give access to the ICSF address space [110](#)
 - removing [109](#)
 - setup AT-TLS [110](#)
 - setup the ICSF address space [110](#)
- RETKEY macro
 - SVC description [7](#)
- return codes
 - from PCF macros
 - migration consideration [199](#)
- RKX key-token [259](#)
- RMF
 - header record format [332](#)
- RSA 1024-bit private internal key token [301–304](#)
- RSA key token formats [287](#)
- RSA private external key token [288](#)
- RSA private internal Chinese remainder theorem key token [308–310](#)
- RSA private internal key token [299, 310, 314, 315](#)
- RSA public token [287](#)
- running ICSF
 - in coexistence mode [200](#)
 - in compatibility mode [199](#)
 - in noncompatibility mode [201](#)
- running the conversion program
 - creating a job to run the conversion program [209](#)
 - defining conversion program data sets [209](#)

S

- scheduling changes for cryptographic keys [124](#)
- secondary parameter block [154](#)
- section sequence, trusted block [316](#)
- security considerations [122](#)
- security installation exit
 - environment [169](#)
 - input [170](#)
 - installing [169](#)
 - purpose and use [168](#)
 - return codes [171](#)
- selecting ICSF startup options
 - creating the installation options data set [21](#)
 - creating the startup procedure [25](#)
- sending to IBM
 - reader comments [xxiii](#)
- service installation exit
 - environment [144](#)
 - exit parameter block [151](#)
 - input [150](#)
 - installing [145](#)
 - parameters [156](#)
 - purpose and use [144](#)
 - return codes [156](#)
- SERVICE installation option
 - syntax [188](#)
- service stub
 - description [185](#)
 - example [190](#)
 - linking [189](#)
 - writing [188](#)
- SETICSF command
 - adding a regional cryptographic server [112](#)
 - changing coprocessor or accelerator status [112](#)
- shortcut keys [395](#)
- single-record, read-write installation exit
 - conversion program invocation [203](#)
 - input [166](#)
 - installing [165](#)
 - purpose and use [165](#)
 - return codes [167](#)
- SMF record type 82
 - subtype 1 [116](#)
 - subtype 13 [117](#)
 - subtype 14 [117](#)
 - subtype 15 [117](#)
 - subtype 16 [117](#)
 - subtype 18 [118](#)
 - subtype 19 [118](#)
 - subtype 20 [118](#)
 - subtype 21 [119](#)
 - subtype 22 [119](#)
 - subtype 23 [119](#)
 - subtype 24 [119](#)
 - subtype 25 [119](#)
 - subtype 26 [120](#)
 - subtype 43 [121](#)
 - subtype 7 [116](#)
 - subtype 8 [116](#)
 - subtype 9 [117](#)
- SMF recording [114](#), [182](#)
- specifying the installation options data set [25](#)
- SSM installation option [48](#)

- START command [28](#)
- starting ICSF
 - creating the startup procedure [25](#)
 - entering the ICSF START command [28](#), [85](#)
 - IPL-time [86](#)
- startup procedure [9](#), [25](#)
- steps in installation [9](#)
- stopping ICSF [85](#)
- SVC [143](#) [7](#)
- syntax diagrams
 - how to read [87](#)
- SYS1.PARMLIB
 - customizing [10](#)
 - description [9](#)
- SYS1.PROCLIB
 - description [9](#)
 - storing startup procedure [25](#)
- SYS1.SAMPLIB
 - CSFPRM00 [21](#)
 - description [9](#)
- SYSPLFXCKDS installation option [49](#)
- SYSPLFXPKDS installation option [49](#)
- SYSPLXTKDS installation option [50](#)

T

- testing ICSF [201](#)
- TKDS
 - SMF record type 82 [119](#)
- TKDS (public key data set)
 - creating [18](#)
- TKDS (token data set)
 - description [68](#)
 - format [221](#)
- TKDS (token key data set)
 - description [6](#)
- TKDSN installation option [50](#)
- token data set (TKDS)
 - description [68](#)
 - format [221](#)
- token key data set
 - improving security and reliability for the TKDS [18](#)
- token validation value (TVV) [254](#)
- TRACEENTRY option and ICSF [59](#)
- trusted block create
 - SMF record type 82 [119](#)
- trusted blocks [315](#)

U

- udx
 - access control checking [187](#)
- UDX installation option [50](#)
- user interface
 - ISPF [395](#)
 - TSO/E [395](#)
- USERPARM installation option [50](#)
- using different configurations [105](#)
- using the conversion program override file [204](#)

V

- V2R1 changed information FMID HCR77A1 [xxix](#)

V2R1 changed information FMID HCR77B0 [xxvii](#)
V2R1 deleted information FMID HCR77A1 [xxix](#)
V2R1 deleted information FMID HCR77B0 [xxviii](#)
V2R1 new information FMID HCR77A1 [xxviii](#)
V2R1 new information FMID HCR77B0 [xxvii](#)
V2R2 changed information FMID HCR77B1 [xxvi](#)
V2R2 changed information FMID HCR77C0 [xxv](#)
V2R2 deleted information FMID HCR77B1 [xxvii](#)
V2R2 deleted information FMID HCR77C0 [xxvi](#)
V2R2 new information FMID HCR77B1 [xxvi](#)
V2R2 new information FMID HCR77C0 [xxv](#)
virtual storage constraint relief
 for the caller of ICSF [113](#)
VSAM data set
 creating [13](#)
VTAM session-level encryption
 and ICSF [113](#)

W

WAITLIST installation option [51](#)



SC14-7507-06

