SIEMENS

CardOS/M4.0

Chipcard Operating System

User's Manual **Edition 10/2001**

SIEMENS

CardOS/M4.0

User’s Manual

Contents

Edition 10/2001

1 Contents and Introduction

Contents

1.1 About this Manual

This CardOS/M4 User’s Manual is addressed to experienced designers of smart card applications.

Basic mechanisms and definitions for smart cards are not described in this manual therefore. Please refer to the smart card literature in the references chapter of this manual or additional literature.

This document shall give the application designer an overview of the abilities of CardOS/M4 and lead her or him to a rapid solution of the security application she or he wants to realize using CardOS/M4.

There exist two solutions for CardOS/M4 on different hardware platforms.
CardOS/M4.00 is available on the SLE66CX160S of Infineon.
CardOS/M4.01 is available on the SLE66CX320P of Infineon.

If no explicit version is specified or CardOS/M4 is mentioned in this manual this applies to CardOS/M4.00 and CardOS/M4.01.

Specialties for CardOS/M4.00 and CardOS/M4.01 are explained explicitly.

For additional information about CardOS/M4 refer to the newest releases of the

- ***CardOS/M4.00 Packages & Release Notes Manual*** respectively
- ***CardOS/M4.01 Packages & Release Notes Manual***.

1.2 References

[1] ISO 7812: 1987, Identification cards:
Numbering system and registration procedure for issuer identifiers

[2] ISO/IEC 7816–3: 1997, Identification cards:
Integrated circuit(s) cards with contacts -
Part 3: Electronic signals and transmission protocols

[3] ISO/IEC 7816–4: 1995, Identification cards:
Integrated circuit(s) cards with contacts -
Part 4: Interindustry commands for interchange

[4] ISO/IEC 7816–4: 1997, Amendment 1: Identification cards:
Integrated circuit(s) cards with contacts -
Impact of secure messaging on the structures of APDU messages

[5] ISO/IEC 7816–5: 1994, Identification cards:
Integrated circuit(s) cards with contacts -
Part 5: Numbering system and registration procedure for application identifiers

[6] ISO/IEC 7816–6: 199x, Identification cards:
Integrated circuit(s) cards with contacts -
Part 6: Interindustry data elements

[7] ISO/IEC 7816–8: 1999, FDIS, Identification cards:
Integrated circuit(s) cards with contacts -
Part 8: Security related interindustry commands

[8] ISO/IEC 7816–9: 1999, FCD, Identification cards:
Integrated circuit(s) cards with contacts -
Part 9: Additional interindustry commands and security attributes

[9] ISO 8825: 1987, Open systems interconnection:
Specification of basic encoding rules for abstract syntax notation one (ASN.1)

[10] ISO/IEC 9796: 1991, Security techniques:
Digital signature scheme giving message recovery

[11] ISO/IEC 9797: 1993, Security techniques:
Data integrity mechanisms using a cryptographic check function employing a block cipher algorithm

[12] ISO/IEC 9979: 1991, Data cryptographic techniques:
Procedures for the registration of cryptographic algorithms

[13] FIPS PUB 180-1: April 1995, (for SHA-1)

[14] ANSI X9.30: 1995, (for DSA)

[15] ANSI X3.92: 1981, Data Encryption Algorithm, (for DES)

[16] PKCS #1 v2.1: RSA Cryptography Standard, DRAFT 1, September 17, 1999.

[17] Bruce Schneier: Applied Cryptography (2nd edition), Wiley, 1995

[18] DIN NI-17.4, Version 1.0 (12/15/1998), German Signature law according to SigG und SigV.

1.3 Glossary

The following abbreviations and definitions are used in this manual:

Abbreviation / Definitions	Meaning
AC	Access Condition
AC Definition	For a specific access action on a file or object an AC definition defines, which access right must be granted in the security status, in order to perform the access action. (AC definition is equivalent to security attribute)
AID	Application Identifier (equal to DF name)
APDU	Application Protocol Data Unit
API	Application Programming Interface
ARA Counter	Access Right Applicability Counter
ATR	Answer to Reset
Backtracking	Mechanism of CardOS/M4 in order to find objects
BER	Basic Encoding Rules (see ISO 7816-4)
BS Object	Basic Security Object (BSO)
USECOUNT	Counter for usage of a BSO
CBC	Cipher Block Chaining
CC component	Component of a SE object or CSE referencing a PSO object for PSO_CCC or PSO_VCC
CHV	Card Holder Verification (equivalent to PIN)
CLA	Class Byte
CON component	Component of a SE object or CSE referencing a PSO object for PSO_DEC or PSO_ENC
CRT	Control Reference Template
CSE	Current Security Environment
Default SE object	SE object with fixed ID = 01h
DEK	Data Encryption Key (rfu)
DES	Data Encryption Standard
DF	Dedicated File
DF Name	Equal to AID
DS component	Component of a SE object or CSE referencing a PSO object for PSO_CDS or PSO_VDS
DSI	Digital Signature Information
ECB	Electronic Code Book
EF	Elementary File
EN	European Standard
FCI	File Control Information
FID	File ID
GSM	Global System for Mobile Communications
HASH component	Component of a SE object or CSE referencing a PSO object for PSO_H
IA component	Component of a SE object or CSE referencing a PSO object for INTERNAL AUTHENTICATE
ICC	Integrated Circuit Card

Abbreviation / Definitions	Meaning
INS	Instruction Byte
LSB	Least Significant Byte
LSBit	Least Significant Bit
MAC	Message Authentication Code
Modulus N	Key Component for Public RSA Keys and Private RSA Keys
MF	Master File
MSB	Most Significant Byte
MSBit	Most Significant Bit
MSE	MANAGE SECURITY ENVIRONMENT command
OCI	Object Control Information
P1-P2	Parameter Bytes
Personalization data	Structures of the operating system (e.g. objects, key data, ...) or a concatenation thereof
Phase	Life cycle Phase
PIN	Personal Identification Number (equivalent to CHV)
PK	Public Key
Private_Exponent D	Exponent for Private RSA Key
PSO	PERFORM SECURITY OPERATION command
PSO_CCC	PSO for a COMPUTE CRYPTOGRAPHIC CHECKSUM operation
PSO_CDS	PSO for a COMPUTE DIGITAL SIGNATURE operation
PSO_DEC	PSO for a DECIPHER operation
PSO_ENC	PSO for a ENCIPHER operation
PSO_H	PSO for a HASHING operation
PSO_VCC	PSO for a VERIFY CRYPTOGRAPHIC CHECKSUM operation
PSO_VDS	PSO for a VERIFY DIGITAL SIGNATURE operation
PSO Object	BS Object used for all PSO commands
PTS	Protocol Type Selection
Public_Exponent E	Exponent for Public RSA Key
Retail MAC	MAC built by the algorithms MAC3 and iMAC3
RFU, rfu	Reserved for Future Use
RPN	Reverse Polish Notation (i.e. internal format for logical test definition)
Security Attribute	Equivalent to AC Definition
SE	Security Environment
SE Object	Security Environment Object
SFI	Short File ID
SM	Secure Messaging
TEST component	Component of a SE object or CSE referencing a PSO object for EXTERNAL AUTHENTICATE or VERIFY
TLV	Tag, Length, Value

1.4 Differences between CardOS/M3 and CardOS/M4

Introduction

This section gives a brief overview of the differences between CardOS/M3 and CardOS/M4. These differences can be categorized in

- **General changes** (i.e. changes, which have impact on the general behavior of the smart card)
- **Personalization relevant changes** (i.e. changes, which have impact on the initialization and personalization of the smart card)
- **Application relevant changes** (i.e. changes, which have impact on the smart card application) and
- **Add-Ons** (i.e. changes, which are an add-on on the CardOS/M3 functionality)

General Changes

Unlike CardOS/M3 for CardOS/M4 all DES- and RSA-based algorithms are **DPA/SPA safe**. Therefore in CardOS/M4 the execution times for these algorithms are increased in comparison to CardOS/M3. Because of the advanced security features the SLE66P-series is offering it is recommended to use CardOS/M4.01 to meet the highest degree of security.

All CardOS/M3 commands using CLA bytes B0h/Bxh or A0h/Axh are available with CLA byte 80h/8xh in Cardos/M4 now. This has impact on the following commands:

- CHANGE KEY,
- DECREASE,
- DIRECTORY,
- ERASE FILES,
- FORMAT,
- GIVE RANDOM,
- INCREASE,
- INITIALIZE EEPROM,
- INITIALIZE END,
- LOAD EXECUTABLE,
- PERSONALIZE,
- PHASE CONTROL,
- RESET SECURITY STATE,
- UNINSTALL PACKAGE.

Data Encryption Keys (DEK) are not supported any more. Instead of using a DEK the corresponding SM mechanism shall be used. This is relevant for the commands:

- CHANGE REFERENCE DATA,
- RESET RETRY COUNTER.

Personalization Relevant Changes

The life cycle phase schemes of CardOS/M3 and CardOS/M4 are different. Therefore the following personalization commands are removed:

- OPEN,
- TEST EEPROM

and the command

- PERSONALIZE

is added.

The order of the ACs/SMs in the AC/SM definitions was changed. This has impact on the commands

- CREATE FILE,
- FORMAT,
- PUT DATA_FCI,
- PUT DATA_OCI.

The usage and hierarchy of objects needed for the CSE (current security environment) were homogenized. This has impact on the following personalization commands:

- PUT DATA_OCI,
- PUT DATA_SECI (new command in CardOS/M4).

Application Relevant Changes

The usage and hierarchy of objects needed for the CSE (current security environment) were homogenized. This has impact on the following application relevant commands:

- MANAGE SECURITY ENVIRONMENT (MSE),
- PERFORM SECURITY OPERATION (PSO).

Add-Ons

CardOS/M4 newly supports the so-called **Access Right Applicability COUNTER (ARA_COUNTER)**. Using this ARA_COUNTER the number of usages for a granted access right can be specified (unlimited or limited to a fixed number). This has impact on the internal representation of the security status and the command

- PUT DATA_OCI.

For the backtracking mechanism of CardOS/M4 it can be specified whether the search starts in the current DF or in the MF. This has impact on the commands:

- CHANGE REFERENCE DATA,
- EXTERNAL AUTHENTICATE,
- INTERNAL AUTHENTICATE,
- RESET RETRY COUNTER,
- VERIFY.

Using CardOS/M4 the **length of PIN data can be changed** in the life cycle phase *OPERATIONAL* via the commands

- CHANGE REFERENCE DATA and
- RESET RETRY COUNTER.

The object IDs for TEST objects and respectively the belonging ACs are in the range 01h ... 7Fh now (for CardOS/M3 these IDs were in the range 01h ... 20h).

The number of the nesting DF levels is limited by the available XRAM of the smart card only (Number of nesting DF levels is 8 for CardOS/M3).

2 Data Security

2.1 Introduction

General

CardOS/M4 uses many flexible and different protection mechanisms to ensure optimum data security:

- Control of commands via life cycle phases.
- Access protection for files and objects with up to 127 distinct access rights on each DF level.
- Logical combination of access rights with any desired complexity.
- Key management with any number of keys and distinction of keys with different usage.
- Secure messaging (SM) for protection of communication data between smart card and terminal/host.

Object oriented Approach

In CardOS/M4 the security architecture is based on objects. Using these objects the security design of the complete smart card can exactly be adapted to the needs of the customer.

Objects define,

- which authentication methods must be used (i.e. which tests must be performed successfully) before any access action on a file or object is permitted,
- how the security behavior of the smart card is

and objects

- store all information (e.g. keys) needed for cryptographic operations including SM.

Objects can be stored in any DF. Therefore the security architecture can be designed in each DF (e.g. application) separately and - if desired - independently of the security architecture of other DFs.

2.2 Security Architecture

2.2.1 Usage of the Smart Card

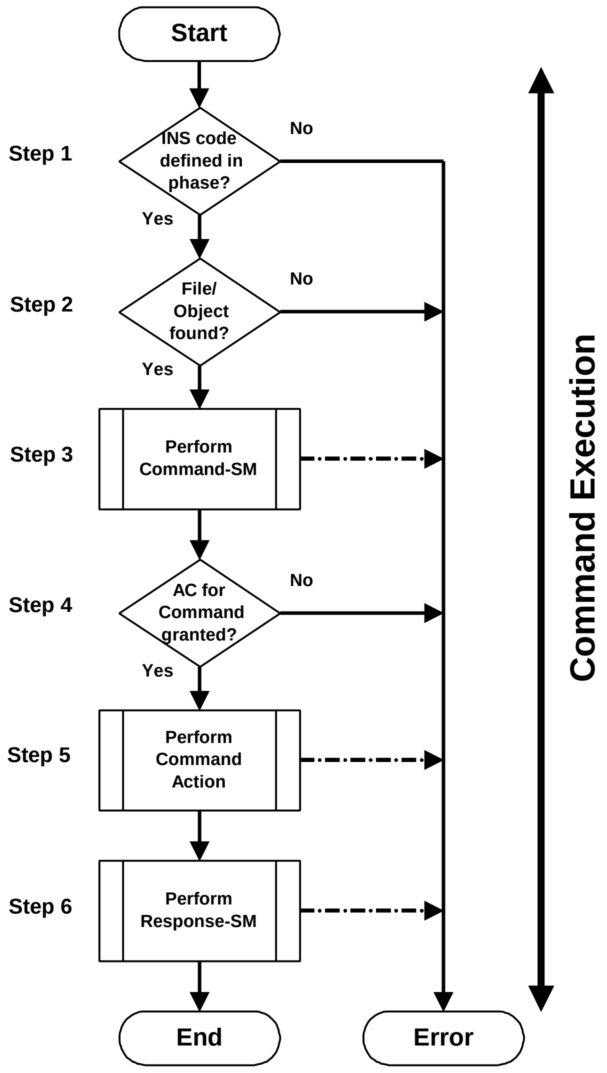
General

Smart cards are used via commands, which are sent from a terminal/host to the smart card. After this the smart card executes the received command and returns a response.

Security

In order to secure the usage of a smart card CardOS/M4 uses different protection mechanisms, which are **command-based** generally. Picture 2.2-1 shows the main steps of command execution, which can occur potentially. This means that for special commands and circumstances not all steps will be executed.

Picture 2.2.1-1 Main steps of Command Execution in CardOS/M4



Step 1

In Step 1 CardOS/M4 checks, whether the command is known (i.e. permitted) in the current life cycle phase (see section 2.2.3 also). If the command is known, then command execution is continued, otherwise command execution is aborted.

Step 1 is performed for **all** commands sent to the CardOS/M4 smart card.

Step 2

Most CardOS/M4 commands need internal information of the smart card in order to perform SM and the command action. This internal information is given in files or objects. In the commands the belonging file/object is referenced via a file/object ID. In Step 2 CardOS/M4 checks, whether the specified file/object ID is found.

Step 2 may only be performed in life cycle phases, in which a filesystem exists (i.e. the life cycle phases *ADMINISTRATION* and *OPERATIONAL*).

Step 3

If specified so in the referenced file/object (or specified for personalization commands), Command-SM (i.e. secure messaging for communication data from the terminal/host to the smart card) is performed.

Step 3 may be performed in any life cycle phase. In life cycle phases, in which no filesystem exists on the smart card (i.e. during personalization), for each personalization command it is specified, whether SM must/can be performed or not.

Step 4

In Step 4 CardOS/M4 checks, whether the AC (i.e. Access Condition or Access Right) necessary for the command is granted in the current security status of the smart card. If the AC is not granted command execution is aborted immediately with an error. The AC is defined in the AC definitions of the referenced file/object (see Step 2).

Step 4 may only be performed in life cycle phases, in which a filesystem exists (i.e. the life cycle phases *ADMINISTRATION* and *OPERATIONAL*).

Step 5

In this step the command action is performed.

Step 5 can be performed in any life cycle phase.

Step 6

If specified so in the referenced file/object, Response-SM (i.e. secure messaging for communication data from the smart card to the terminal/host) is performed.

As personalization commands do not return data, Step 6 may only be performed in life cycle phases, in which a filesystem exists (i.e. the life cycle phases *ADMINISTRATION* and *OPERATIONAL*).

2.2.2 Initialization and Personalization

Introduction

In the initialization and personalization process of smart cards several parties may normally be involved. These parties may be:

- **Chip Manufacturer,**
- **OS (Operating System) Manufacturer**
- **Card Issuer**
- **Application Provider(s)**.

Each party has own security interests and therefore wants to protect the information in the smart card and keep this information secret from all the other parties involved in the process. For this CardOS/M4 offers a concept for initialization and personalization.

Logical and physical personalization

CardOS/M4 provides a **logical personalization** with which the filesystem, the file data and the secret data can be loaded into the smart card using default ISO and proprietary commands (e.g. CREATE FILE). Logical personalization is performed in the life cycle phase *ADMINISTRATION*.

For mass production CardOS/M4 offers the **physical personalization**. Physical personalization is executed in the life cycle phases *INITIALIZATION*, where the initialization is performed, and *PERSONALIZATION*, where the personalization is performed. During the physical personalization certain parts of a consistent filesystem (A consistent filesystem contains all files, data and secret data.) are loaded into the smart card. After having installed all these parts of the filesystem the normal operation of the smart card is possible (i.e. in life cycle phase *OPERATIONAL*).

2.2.2.1 System Keys

System Keys

During personalization CardOS/M4 uses the following system keys:

- **StartKey**
  The initial *StartKey* is loaded into EEPROM by the chip manufacturer. It is a 16 byte key and will be used for SM mode x4h. The *StartKey* can be changed for each smart card, i.e. the personalization for each smart card may be secured by an own secret *StartKey*. **In order to prevent misuse of the CardOS/M4 smart cards and for security reasons, it is strongly recommended to change the *StartKey.***
- **PackageLoadKey**
  The initial *PackageLoadKey* is loaded into EEPROM by the chip manufacturer. It is a 16 byte key and will be used for protection only in order to ensure the data integrity of system and application packages, which shall be activated in the smart card. The *PackageLoadKey* can be changed for each smart card, i.e. activation of system and application packages may be secured by an own *PackageLoadKey* for each smart card.
- **PersonalizationKey(s)**
  For each application an initial *PersonalizationKey* is loaded into EEPROM during life cycle phase INITIALIZATION (e.g. by the card issuer). It is a 16 byte key and will be used for SM mode x4h. The *PersonalizationKey* can be changed, i.e. loading of personalization data may be secured by an own *PersonalizationKey* for each application on the smart card.

As system keys are 16 bytes in length, they are using a A-B-A key structure. Each system key has an error counter with a maximum of **10**.

- Each time a MAC-verification with a system key fails, the corresponding error counter is decreased. This means that after 10 times of unsuccessful usage any system key is blocked (i.e. error counter = 00h) irreversibly and all commands using that system key cannot be performed any more.

Each time the MAC-verification of the *StartKey* or any *PersonalizationKey* is successful the error counter of the system key is set to its maximum (i.e. 10). The error counter of the *PackageLoadKey* will never be increased.

The *StartKey* and the *PersonalizationKey(s)* are used with SM mode x4h only. As SM mode x4h has fixed algorithms only, this means, that system keys are used with the DES3_CBC and the MAC3 algorithm. One and the same key is used for encryption **and** MACing always.

In CardOS/M4.01 the version of the *StartKey* and the *PackageLoadKey can be changed* in addition to a new value of the key.

Usage of System Keys

Table 2.2.2.1-1 shows, which commands use which system key(s).

Table 2.2.2.1-1 Commands using System Keys

Command	Usable System Key(s)
CHANGE KEY	StartKey, PackageLoadKey, PersonalizationKey(s)
ERASE FILES	StartKey
FORMAT	StartKey
INITIALIZE EEPROM	StartKey
INITIALIZE END	StartKey
LOAD EXECUTABLE	PackageLoadKey
PERSONALIZE	PersonalizationKey(s)

Packages

Packages (i.e. OS extensions) can be loaded to EEPROM and activated during logical personalization (i.e. application packages) or during physical personalization (i.e. system packages). Packages are developed and controlled by the OS manufacturer.

Application packages are files whereas system packages are loaded as images (i.e. not in a file) inside the scope of the filesystem.

Loading of application packages may be secured by SM mode xCh with freely defined SM keys whereas loading of system packages may be secured by SM mode x4h using the *StartKey*.

For all packages (i.e. application and system packages) **activation** of the packages (via the LOAD EXECUTABLE command) is secured with the *PackageLoadKey*.

2.2.2.2 The Physical Personalization Scheme

Shared Secrets among parties

The card issuer may not want the chip manufacturer and the OS manufacturer to know which *StartKey* is used to initialize the filesystem. Therefore the card issuer can change the initial *StartKey* (loaded by the chip manufacturer) before any initialization command is sent to the smart card.

In order to change the *StartKey* via the CHANGE KEY command the current *StartKey* must be known. After the change the card issuer is independent from the chip manufacturer and the OS manufacturer. If this scheme does not fit the security requirements of the card issuer it is possible that the card issuer will get its own *StartKey* during chip manufacturing. For this a secure communication procedure between the chip manufacturer and the card issuer (e.g. secret letter with the StartKey shared among the chip manufacturer and the card issuer) must be established.

The card issuer and the application provider(s) may want to keep two parts of information separately secret:

- Initial filesystem (for the card issuer) and
- Personalization data (for the application provider(s)).

Therefore these data are loaded to the smart card at two or more different sites:

- the initialization site (card issuer site) and
- the personalization site(s) (application provider site(s)).

For the initialization site the *StartKey* is provided whereas an own *PersonalizationKey* is provided for each application provider site.

At the initialization site the initial filesystem is loaded to the smart card using the *StartKey*. Also the initial *PersonalizationKey(s)* is (are) loaded, which is (are) defined and shared between the card issuer and each application provider separately (i.e. same principle as for the *StartKey*).

At each personalization site the sensitive individual card data are loaded. In order to secure this each application provider can change his **initial** *PersonalizationKey* to his own **secret** *PersonalizationKey*, which will then be used to secure loading of all individual card data.

Using this scheme personalization data of different application providers can be loaded to the smart card independently secured by different *PersonalizationKeys*. This means that the personalization data of each application provider must be loaded to the correct locations in the filesystem.

Therefore the card issuer defines **separately with each** application provider the sequences of the personalization data and the locations where the personalization data are to be loaded. Additionally an initial *PersonalizationKey* and a **unique personalization number** for identifying the application provider are defined.

With this information so-called **placeholder(s)** and an application descriptor will be created and loaded together with the filesystem for each application provider.

Background Information

The following descriptions in this section are background information, which is **not necessarily needed to use CardOS/M4**.

Placeholders

During initialization a placeholder is loaded (via the INITIALIZE EEPROM command) exactly to that location in EEPROM, where the personalization data shall be loaded during personalization. This means, that for each entry of personalization data a placeholder is necessary.

All placeholders of an application are chained together as a **placeholder list**. The personalization data to be loaded must have the same order as in the placeholder list. Each placeholder consists of

- a **control byte**
- a **length byte** and
- **offset to the next placeholder** (2 bytes).

The **control byte** specifies, whether the personalization data are loaded unsecured or secured using MACing and/or ENCryption.

The **length byte** specifies the number of bytes of the personalization data.

The **offset to the next placeholder** specifies the location of the next placeholder in placeholder list. The offset of the first placeholder in the placeholder list is part of the application descriptor. If the offset value of the application descriptor or the placeholder is 0000h, then the end of the placeholder list is reached. The offset is calculated from the beginning of the filesystem to the location of the placeholder.

Application Descriptor

One application descriptor is necessary for each application provider and his personalization data. The application descriptor contains

- the **personalization number** (1 byte),
- the **retry counter for the *PersonalizationKey*** (1 byte),
- the ***PersonalizationKey*** (16 bytes),
- the **offset to first placeholder** (2 bytes),
- the **sequence counter** (2 bytes), and
- the **working area** (8 bytes), which must be set to 00h.

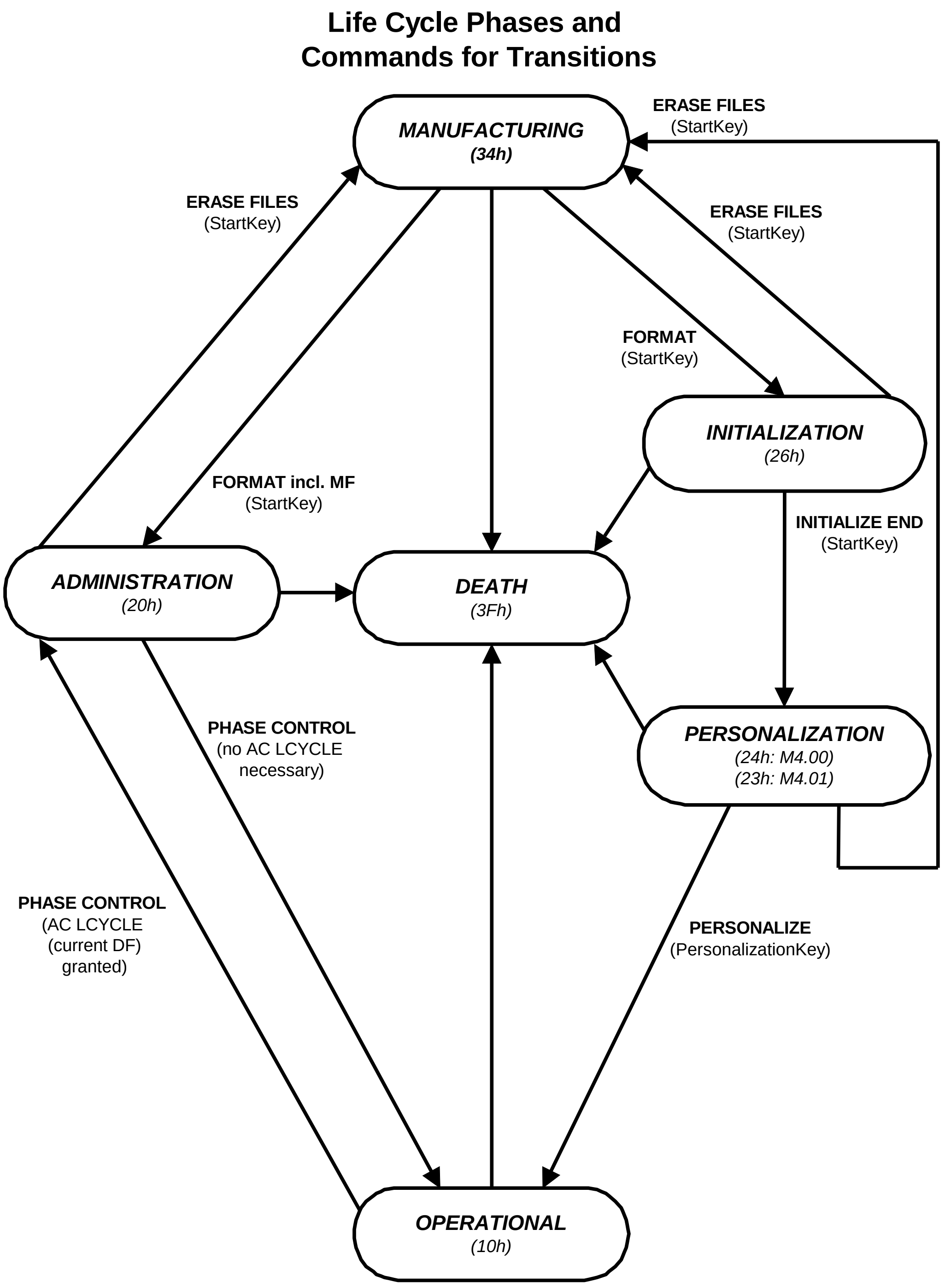
The **sequence counter** will be incremented each time when a placeholder shall be replaced by the personalization data via the PERSONALIZE command in life cycle phase *PERSONALIZATION*. Before the placeholder is replaced the sequence number sent with the personalization data is compared with the sequence counter of the application descriptor. If both values are identical, then the placeholder is replaced by the personalization data.

Each application descriptor is loaded to a location outside the filesystem of the smart card via the INITIALIZE EEPROM command in the life cycle phase *INITIALIZATION.* Each application descriptor will be erased, if **all** placeholders are replaced by the personalization data (i.e. all personalization data of one application provider).

2.2.3 Life Cycle Phases

Description In order to secure the personalization of a CardOS/M4 smart card different so-called life cycle phases are provided. Picture 2.2.3-1 shows, which life cycle phases are provided

Picture 2.2.3-1 Life Cycle Phases and Transitions between them



The current life cycle phase of the smart card (i.e. the corresponding hexadecimal notation given in brackets) can be read using the GET DATA command. Therefore GET DATA can be performed in any life cycle phase.

Transitions between the life cycle phases are possible using the specified commands and system keys / AC. All transitions shown in picture 2.2.3-1 except the transition from the *OPERATIONAL* to the *ADMINISTRATION* phase are permanent.

A permanent transition changes the current life cycle phase permanently. This means that the current life cycle phase is not affected by a reset of the smart card.

The transition from the *OPERATIONAL* to the *ADMINISTRATION* phase sets the current life cycle phase from *OPERATIONAL* to *ADMINISTRATION* temporarily. This means that after a reset of the smart card the current life cycle phase will be *OPERATIONAL* again.

MANUFACTURING

This is the life cycle phase after chip production

INITIALIZATION

This life cycle phase is used for initialization of the CardOS/M4 smart card. During initialization basic structures and data are loaded to the smart card as well as data, which are needed for the succeeding *PERSONALIZATION* life cycle phase.

Using the INITIALIZE EEPROM command in this life cycle phase it is possible to load the following data to the smart card:
- **filesystem structures**,
- **data structures**,
- **application descriptors,**
- **placeholders** and
- eventually **file data**.

The INITIALIZE EEPROM command can be executed with or without secure messaging (SM) using the *StartKey*.

Using the LOAD EXECUTABLE command secured with the *PackageLoadKey* so-called
- **system packages**

can be loaded as an image inside the filesystem of the CardOS/M4 smart card.

PERSONALIZATION

This life cycle phase is used for personalization of the CardOS/M4 smart card. During personalization the placeholders, which were loaded to the smart card in INITIALIZATION phase, are replaced by the personalization data.

Using the PERSONALIZE command a placeholder is replaced by real personalization data. Each PERSONALIZE command provides a sequence number, which will be compared with the sequence counter of the belonging application descriptor. If the values are identical the replacement will be performed by the PERSONALIZE command.

If the last placeholder of the placeholder list is replaced the belonging application descriptor will be erased as well. If the last application descriptor is erased the life cycle phase will be changed from PERSONALIZATION to MANUFACTURING automatically.

The personalization data, all filesystem information, and the system packages may be deleted via the ERASE FILES command using the *StartKey*. Additionally the life cycle phase will change from PERSONALIZATION to MANUFACTURING.

ADMINISTRATION In this life cycle phase the logical personalization of the CardOS/M4 smart card takes place. This means that in this phase all files, objects and data needed for the use of the smart card are installed and activated in the smart card.

After the logical personalization this life cycle phase is left using the PHASE CONTROL command.

OPERATIONAL In this life cycle phase the smart card is issued to the customer.

DEATH In this life cycle phase all smart card commands except the GET DATA command are disabled and no command exists to leave the life cycle phase *DEATH*, i.e. the smart card is **blocked irreversibly**.

The life cycle phase *DEATH* will be reached if special events occur in the smart card (e.g. EEPROM weakness, filesystem or EEPROM corrupted or special mechanisms against fraud).

ERASE IN PROGRESS This life cycle phase is passed during the run of the ERASE FILES command.

Command Set Each life cycle phase has a specific command set (i.e. a subset of all CardOS/M4 commands) shown on the next page in table 2.2.3-1.

Table 2.2.3-1 Command Sets for different Life Cycle Phases

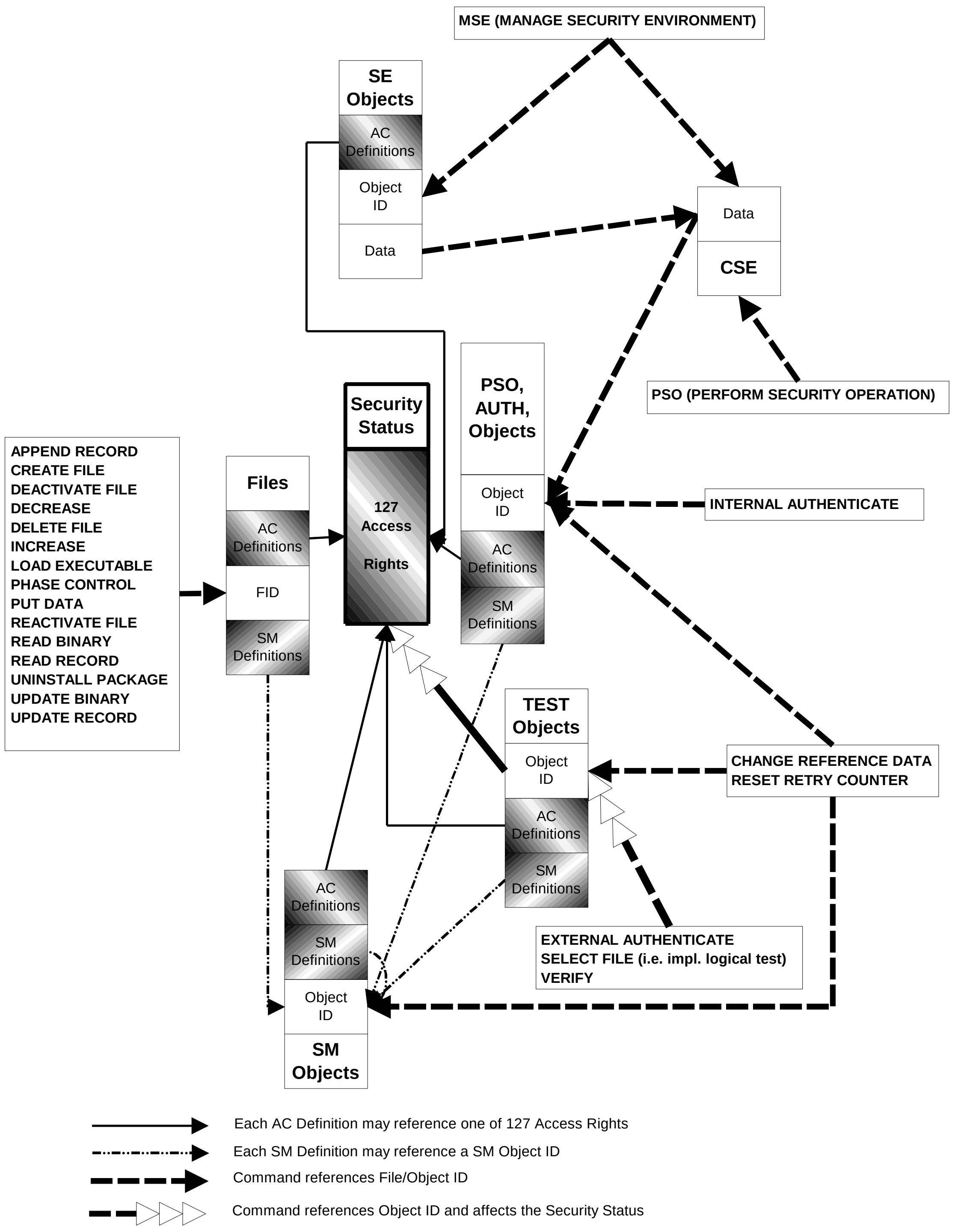
Command	CLA [hex]	INS [hex]	MANUFACTURING	INITIALIZATION	PERSONALIZATION	ADMINISTRATION	OPERATIONAL	DEATH
ACTIVATE FILE	00h	44h				X	X	
APPEND RECORD	0xh	E2h				X	X	
CHANGE KEY	84h	24h	X	X	X			
CHANGE REFERENCE DATA	0xh	24h				X	X	
CREATE FILE	0xh	E0h				X		
DEACTIVATE FILE	00h	04h				X	X	
DECREASE	8xh	30h				X	X	
DELETE FILE [01]	0xh	E4h				X		
DIRECTORY [01]	80h	16h				X	X	
ERASE FILES	84h	06h	X	X	X	X		
EXTERNAL AUTHENTICATE	0xh	82h				X	X	
FORMAT	84h	40h	X					
GENERATE KEY PAIR [01]	0xh	46h				X		
GET CHALLENGE	00h	84h				X	X	
GET DATA	00h	CAh	X	X	X	X	X	X
GIVE RANDOM	80h	86h				X	X	
INCREASE	8xh	32h				X	X	
INITIALIZE EEPROM	80h / 84h	02h		X				
INITIALIZE END	84h	00h		X				
INTERNAL AUTHENTICATE	0xh	88h				X	X	
LOAD EXECUTABLE	80h / 84h [02]	20h		X		X		
MANAGE CHANNEL [01]	00h	70h				X	X	
MANAGE SECURITY ENVIRONMENT (MSE)	00h	22h				X	X	
PERFORM SECURITY OPERATION (PSO)	0xh / 1xh	2Ah				X	X	
PERSONALIZE	80h / 84h	08h			X			
PHASE CONTROL	80h	10h				X	X	
PUT DATA	0xh	DAh				X		
READ BINARY	0xh	B0h				X	X	
READ RECORD	0xh	B2h				X	X	
RESET RETRY COUNTER	0xh	2Ch				X	X	
RESET SECURITY STATUS [01]	80h	EAh				X	X	
SELECT FILE	00h	A4h				X	X	
UNINSTALL PACKAGE [01]	80h / 84h	E4h		X		X		
UPDATE BINARY	0xh	D6h				X	X	
UPDATE RECORD	0xh	DCh				X	X	
VERIFY	0xh	20h				X	X	

(01)	CardOS/M4.00:	Available, if belonging package was loaded and activated before.
	CardOS/M4.01:	Available always
(02)	CardOS/M4.00:	80h
	CardOS/M4.01:	84h

2.2.4 Protection Mechanisms

Picture 2.2.4-1 Protection Mechanisms

Protection Mechanisms and their Usage via Commands



Description

Picture 2.2.4-1 gives an overview of the non-life cycle related protection mechanisms of CardOS/M4. These mechanisms are available in the life cycle phases *ADMINISTRATION* and *OPERATIONAL*.

The picture shows, which commands work with files and which work with objects. All commands reference the belonging file/object via its ID except the PSO command , which needs a CSE always.

All files/objects have AC definitions and SM definitions. The mechanisms of using AC definitions and SM (i.e. secure messaging) definitions are identical for files and objects (even for SM objects themselves).

The protection mechanisms can be distinguished in

- Protection of File/Object data and
- Protection of Communication Data (i.e. SM)

Both protection mechanisms are described in the following sections and are referencing picture 2.2.4-1.

2.2.4.1 Access Protection

Access Actions

Access actions on files or objects are performed during command execution. Any access action on a file or object (e.g. UPDATE BINARY or PUT DATA_OCI command for an update action) can be protected using the so-called *security status* and AC definitions.

Security Status

Each DF of the smart card filesystem has a *DF-specific security status*. For all commands of CardOS/M4 only the *security status* of the ***current DF*** is used for execution (if defined so in the AC definitions).

The *security status* consists of up to 127 access rights. Only TEST objects affect the *security status* of the DF, where they are defined, and all child-DFs. Other objects **do not affect** the *security status*.

AC Definitions

For files and objects AC definitions (i.e. security attributes) define, which access right in the *security status* must be granted in order to permit an access action on a file respectively object.

For files the AC definitions are installed during file creation via the FORMAT respectively CREATE FILE command. Using the PUT DATA_FCI command the AC definitions for files can be administrated.

For objects the AC definitions are installed during object creation via the PUT DATA_OCI or PUT DATA_SECI command. Using the PUT DATA_OCI or PUT DATA_SECI command the BS object's respectively the SE object's AC definitions cannot be administrated but overwritten. This means that all settings (incl. AC definitions) of a BS object or a SE object must be fixed during installation or overwriting as if all settings were **one unit**.

Access Rights

An access right can be granted (i.e. set to binary '1') or can be not granted (i.e. set to binary '0').

After a reset of the smart card no access right is granted. Access rights can be acquired performing a security test. 3 different kinds of security tests are provided in CardOS/M4:
- Using an entity authentication with password (i.e. a PIN test using the VERIFY command) or
- using an entity authentication with key (i.e. C/R test using the EXTERNAL AUTHENTICATION command) or
- using a predefined logical test, which is performed implicitly (i.e. at the end of any VERIFY, EXTERNAL AUTHENTICATE or SELECT FILE command).

Access rights may be withdrawn
- after an unsuccessful PIN test or
- after an unsuccessful C/R test or
- after having left the test context
  (i.e. the DF, in which the necessary TEST object is stored).

All information for the test to be performed is stored in a TEST object. The object ID of the TEST object specifies exactly the access right, which is granted after a successful test execution (Therefore the range of object IDs for TEST objects is in between 01h and 7Fh according to the 127 possible access rights in the security status.).

Access Right Applicability COUNTER

CardOS/M4 supports the so-called **Access Right Applicability COUNTER (ARA_COUNTER)**. Using this ARA_COUNTER the number of usages for a granted access right can be specified (unlimited or limited to a fixed number, which is specified in the belonging TEST object).

Scope of validity

Access rights and their validity are subject to the following rules:

1st Access rights can be acquired in any DF, from which - according to the backtracking mechanism of CardOS/M4 - the corresponding TEST object can be found.

2nd Access rights are granted (i.e. set to binary '1') in the DF, where the corresponding TEST object is stored, and in all child-DFs (i.e. in all DFs below this DF). The scope of validity for access rights is the DF, where the corresponding TEST object is stored, and all child-DFs.

3rd Access rights are withdrawn (i.e. set to binary '0'), when the scope of validity is exited.

4th Access rights are withdrawn in the scope of validity, when execution of the corresponding test is not successful or, at the latest, when the test has been blocked.

5th No access rights are granted after a reset of the smart card (i.e. after startup of CardOS/M4).

2.2.4.2 Access Definitions for Files

Description The following tables 2.2.4.2-1 to 2.2.4.2-3 show for a MF, a DF and an EF, which command(s) on files use which AC definition(s). This means, it is shown, which AC definition must be used in order to protect a specific access action (i.e. command) on a file.

Table 2.2.4.2-1 AC Definitions for the MF

Byte No.	AC definitions (possible values)	Meaning for MF	Commands protected with access right referenced by AC definition
1	00h ... 7Fh, FFh	AC LCYCLE	PHASE CONTROL
2	00h ... 7Fh, FFh	AC UPDATE (Objects)	PUT DATA_OCI, PUT DATA_ATR, PUT DATA_DSI
3	00h ... 7Fh, FFh	AC APPEND (Objects)	PUT DATA_OCI, PUT DATA_ATR, PUT DATA_DSI
4	00h ... 7Fh, FFh	AC DEACTIVATE (MF)	DEACTIVATE FILE
5	00h ... 7Fh, FFh	AC ACTIVATE (MF)	ACTIVATE FILE
6	00h ... 7Fh, FFh	AC ERASE (MF) (*)	ERASE FILES (*)
7	00h ... 7Fh, FFh	AC ADMIN (MF)	PUT DATA_FCI
8	00h ... 7Fh, FFh	AC CREATE (Files)	CREATE FILE
9	00h ... 7Fh, FFh	AC EXECUTE	LOAD EXECUTABLE

(*) rfu for CardOS/M4.00, active for CardOS/M4.01 and higher

Table 2.2.4.2-2 AC Definitions for the DF

Byte No.	AC definitions (possible values)	Meaning for DF	Commands protected with access right referenced by AC definition
1	00h ... 7Fh, FFh	AC LCYCLE	PHASE CONTROL
2	00h ... 7Fh, FFh	AC UPDATE (Objects)	PUT DATA_OCI, PUT DATA_DSI
3	00h ... 7Fh, FFh	AC APPEND (Objects)	PUT DATA_OCI, PUT DATA_DSI
4	00h ... 7Fh, FFh	AC DEACTIVATE (DF)	DEACTIVATE FILE
5	00h ... 7Fh, FFh	AC ACTIVATE (DF)	ACTIVATE FILE
6	00h ... 7Fh, FFh	AC DELETE (DF)	DELETE FILE
7	00h ... 7Fh, FFh	AC ADMIN (DF)	PUT DATA_FCI
8	00h ... 7Fh, FFh	AC CREATE (Files)	CREATE FILE
9	00h ... 7Fh, FFh	rfu	-

Table 2.2.4.2-3 AC Definitions for the EF

Byte No.	AC definitions (possible values)	Meaning for EF	Commands protected with access right referenced by AC definition
1	00h ... 7Fh, FFh	AC READ (Data)	READ BINARY, READ RECORD
2	00h ... 7Fh, FFh	AC UPDATE (Data)	UPDATE BINARY, UPDATE RECORD
3	00h ... 7Fh, FFh	AC APPEND (Record) / rfu	APPEND RECORD
4	00h ... 7Fh, FFh	AC DEACTIVATE (EF)	DEACTIVATE FILE
5	00h ... 7Fh, FFh	AC ACTIVATE (EF)	ACTIVATE FILE
6	00h ... 7Fh, FFh	AC DELETE (EF)	DELETE FILE
7	00h ... 7Fh, FFh	AC ADMIN (EF)	PUT DATA_FCI
8	00h ... 7Fh, FFh	AC INCREASE / rfu	INCREASE
9	00h ... 7Fh, FFh	AC DECREASE / rfu	DECREASE

Meaning of possible values:

00h:	Access action referenced by AC can **always** be performed (i.e. right is granted).
01h ... 7Fh	Access action referenced by AC can be performed, if right 01h ... 7Fh is granted.
FFh:	Access action referenced by AC can **never** be performed.

2.2.4.3 Access Definitions for BS Objects

Description Table 2.2.4.3-1 shows for BS objects, which command(s) on BS objects use which AC definition(s). This means, it is shown, which AC definition must be used in order to protect a specific access action (i.e. command) on a BS object.

Table 2.2.4.3-1 AC Definitions for BS Objects

Byte No.	AC definitions (possible values)	Meaning	Commands protected with access right referenced by AC definition
1	00h ... 7Fh, FFh	AC USE (BS Object)	EXTERNAL AUTHENTICATE, INTERNAL AUTHENTICATE, PSO_CCC, PSO_CDS, PSO_DEC, PSO_ENC, PSO_VCC, PSO_VDS, VERIFY
2	00h ... 7Fh, FFh	AC CHANGE (BS Object Data)	CHANGE REFERENCE DATA
3	00h ... 7Fh, FFh	AC UNBLOCK (BS Object)	RESET RETRY COUNTER
4	00h ... 7Fh, FFh	rfu	
5	00h ... 7Fh, FFh	rfu	
6	00h ... 7Fh, FFh	rfu	
7	00h ... 7Fh, FFh	AC GENKEY (BS Object)	GENERATE KEY PAIR

Meaning of possible values:

00h:	Access action referenced by AC can **always** be performed (i.e. right is granted).
01h ... 7Fh	Access action referenced by AC can be performed, if right 01h ... 7Fh is granted.
FFh:	Access action referenced by AC can **never** be performed.

2.2.4.4 Access Definitions for SE Objects

Description Table 2.2.4.4-1 shows for SE objects, which command(s) on SE objects use which AC definition(s). This means, it is shown, which AC definition must be used in order to protect a specific access action (i.e. command) on a SE object.

Table 2.2.4.4-1 AC Definitions for SE Objects

Byte No.	AC definitions (possible values)	Meaning	Commands protected with access right referenced by AC definition
1	00h ... 7Fh, FFh	AC RESTORE (SE Object)	MSE (RESTORE mode) (*)
2	00h ... 7Fh, FFh	AC STORE (SE Object) rfu	rfu

Meaning of possible values:

00h:	Access action referenced by AC can **always** be performed (i.e. right is granted).
01h ... 7Fh	Access action referenced by AC can be performed, if right 01h ... 7Fh is granted.
FFh:	Access action referenced by AC can **never** be performed.

(*) Each time a SELECT FILE command is used CardOS/M4.0 performs an implicit MSE (RESTORE mode). Therefore the ARA_COUNTER for the corresponding TEST object shall be set to unlimited, i.e. to 00h or to FFh.

2.2.4.5 Communication Protection

Secure Messaging Secure messaging (SM) is used to protect communication data between smart card and terminal/host.

SM Modes For personalization and for normal use of the smart card two SM modes are provided.

SM Mode x4h SM mode x4h is a proprietary SM mode used for personalization commands as long as no filesystem exists on the smart card (except ERASE FILE). SM mode x4h does not use SM definitions nor SM objects but proprietary methods (see section 2.5.1). This SM mode is designed for a fast and secure personalization of a smart card but less for scalability. This SM mode is **not** related to picture 2.2.4-1.

SM Mode xCh This SM mode is related to picture 2.2.4-1.

SM mode xCh uses SM definitions and SM objects (see section 2.5.2) and guarantees a high degree of scalability. Therefore SM in mode xCh can exactly meet the needs of the customer.

SM Definitions For any possible access action and communication direction on a file/object a SM definition exists (see sections 2.5.2.1 and 2.5.2.2). SM definitions reference SM objects. The referenced SM object determines exactly, how SM shall be performed for the belonging access action and communication direction.

As a result of this the SM for reading and updating data of a file may be completely different using different keys and algorithms.

2.3 Objects

Description

The security architecture of CardOS/M4 is based on different objects, which are classified as

- **BS Objects** (Basic Security Objects) and
- **SE Objects** (Security Environment Objects).

BS Objects

Any command or mechanism of CardOS/M4 is **always** based on BS objects.

BS objects can be used

- **directly**
  (by all commands referencing objects except all PSO commands) or
- **indirectly**.

The following BS Objects exist:

- **TEST Objects,**
- **AUTH Objects,**
- **SM Objects** and
- **PSO Objects.**

Except the TEST objects, which may contain other data as well, **all BS objects contain keys**.

TEST Objects

TEST objects contain all information necessary for execution of a test including the behavior of the smart card during and after test execution.

A TEST object can define

- a **PIN TEST** (Object Data = PIN)
  used by the VERIFY command or
- a **C/R TEST** (i.e. challenge/response test; Object Data = Key)
  used by the EXTERNAL AUTHENTICATE command or
- a **LOGICAL TEST** (Object Data = logical expression in RPN)
  used implicitly by VERIFY, EXTERNAL AUTHENTICATE and SELECT FILE.

Only TEST objects affect the *security status* of the DF, in which they are defined, and all child-DFs. Other objects do not affect the security status.

PIN TEST

When PIN TESTs are involved, test execution is performed with the VERIFY command by comparing the TEST object data with the PIN presented in the command. This means that the TEST object data contain the PIN in plain text.

C/R TEST

A dynamic authentication (i.e. C/R test) is performed using a random number the so-called challenge, which is built in the smart card and which is requested by the terminal/host using a GET CHALLENGE command. Then the challenge is MACed/signed by the terminal/host using the symmetric key/asymmetric private key. The MACed/signed challenge is called the response. The response is sent to the smart card for verification using the EXTERNAL AUTHENTICATE command.

A requested challenge can be used once only and need not to be requested immediately before the EXTERNAL AUTHENTICATE command but during the same smart card session.

- The minimum length of the challenge to be requested from the smart card is specified in the *MINLEN* byte of the TEST object (see PUT DATA_OCI).
- The host/terminal must build the response using the complete challenge (i.e. using all bytes of the challenge incl. padding bytes). If the length of the challenge requested is less than the length specified in *MINLEN* then an error code is issued during command execution of EXTERNAL AUTHENTICATE.

The response is verified differently for symmetric keys and asymmetric keys.

- For symmetric keys the response is verified with a response internally calculated for the challenge sent with the GET CHALLENGE command before.
- For asymmetric keys first the challenge is internally recalculated using the response and the asymmetric public key. Then the internally calculated challenge and the challenge sent with the GET CHALLENGE command are verified.

The symmetric key (MAC key) / asymmetric public key is stored in the TEST object data.

LOGICAL TEST Logical tests are evaluated implicitly by the card each time an access right is changed.

This type of test can be used to grant rights when a logical expression whose operands contain numbers of other access rights supplies the result TRUE (i.e. bit is set to ‘1’). During execution, granted rights are considered TRUE and not granted rights are considered FALSE (i.e. bit is set to ‘0’).

The TEST object data contain the logical expression in ”Reversed Polish Notation” (RPN). For this notation the number of an access right is directly coded with 01h to 7Fh. The operators AND and OR are specified with 00h and 0FFh respectively.

Reserve Polish Notation
The Polish logician Lukasiewicz (1878-1956) developed a method of notation for mathematical and logical expressions in which no rules of precedence need to be defined for operators within the expression. Since the operator precedes its operands in this type of notation, it is also referred to as prefix notation.

However, when a computer or smart card processor evaluates expressions, it is better to have the operator follow the operands so the "Polish" method of prefix notation was simply "reversed" (i.e. reverse Polish notation or RPN).

What does this notation look like after this reverse? The algebraic

expression	3 + 5 * 7
becomes	3 5 7 * +
in RPN, or simply	5 7 * 3 +

when the operands are reordered.
This technique usually uses a stack or LIFO storage (i.e. last-in first-out) which results in the following algorithm.

- If operand, put it into the stack

- If operator, take as many operands from stack as required by the operator. Apply the operator to them and return the result to the stack.

The operands are followed by the operator as can be seen in the example below. The intermediate results are retained in the stack for further processing by subsequent operands.

		7		
	5	5	35	
3	3	3	3	38
3	5	7	*	+

The two possible operators are AND (00h) and OR (FFh). The operands are rights 01 to 127 (01h to 7Fh).

If a test is to be TRUE when one of the rights 01 or 02 and one of the rights 03 or 04 are granted, this is represented in algebraic notation as
(01 OR 02) AND (03 OR 04).
The equivalent in RPN is 01 02 FFh 03 04 FFh 00h

Assumed the current security bits are set as follows

01h: 1
02h: 0
03h: 1
04h: 1

				1		
	0		1	1	1	
1	1	1	1	1	1	1
01h	02h	FFh	03h	04h	FFh	00h

Access rights stack size: max. 128

A correct RPN expression leaves exactly one entry in the stack. This entry is then the result of the expression.

Two errors can occur within an RPN evaluation (i.e., stack underflow and stack overflow).

A stack underflow occurs, when an operator requires more operands than are currently in the stack. This cannot occur when the RPN expression is noted correctly.

A stack overflow occurs, when an attempt is made to enter more operands in the stack than it can hold. For CardOS, the size of the stack is > 100.

Many stack overflows can be avoided by reordering the operands. Other expressions are too complex for the given stack size.

NOTE: If the syntax of the expression in RPN is not correct, then eventually the belonging access right will not be granted.

AUTH, SM Objects These BS Objects contain keys for different purposes.

PSO Objects PSO objects are used by all PSO commands **indirectly**. PSO objects contain keys always.

SE Objects One SE object contains references to the following BS objects:

- 4 **PSO objects** (i.e.
  - **SE object's CC component**,
  - **SE object's DS component**,
  - **SE object's HASH component,**
  - **SE object's CON component**).
- 1 **TEST object** (i.e. SE object's TEST component), and
- 1 **AUTH object** (i.e. SE object's IA component).

Using the MSE command in RESTORE mode the references of a SE object, which is referenced by its ID, are copied to the so-called *current security environment* (CSE).

- Subsequent PSO commands **must** use the CSE references to the PSO objects.
- Subsequent EXTERNAL AUTHENTICATE or VERIFY commands **may** use the CSE reference to the TEST object.
- Subsequent INTERNAL AUTHENTICATE commands **may** use the CSE reference to the AUTH object.

When performing a PSO command CardOS/M4 searches for a CSE first. If the CSE is found, CardOS/M4 searches for the relevant PSO object referenced by the found CSE. When the PSO object is found the PSO command works with the PSO object. The described search represents Step 2 of picture 2.2.1-1.

Before using a PSO command a MSE command in RESTORE mode or eventually a SELECT FILE command must be executed in order to create a CSE.

Creation BS Objects can be created using the PUT DATA_OCI command whereas SE objects can be created using the PUT DATA_SECI command. The objects are stored in DFs and can be addressed using the object address parameters described in section 2.3.1.

Search Objects are searched for according to the backtracking mechanism of CardOS/M4 described in section 2.3.2.

2.3.1 Format of BS Objects

Description

This section is an addition to the BS object format description of the PUT DATA_OCI command. Some of the mechanisms for BS objects are described in more detail now.

Table 2.3.1-1 gives an overview of the BS object format, i.e. all the OCI (Object Control Information) parameters, which can be specified, when installing a BS object via the PUT DATA_OCI command.

Table 2.3.1-1 Overview over OCI Parameters

T	L	V (Value)	m /o	Description
83h	02h	Byte 1: Object Class Byte 2: Object ID	m	Object Address
85h	08h	Byte 1: OPTIONS Byte 2: FLAGS Byte 3: ALGO Byte 4: ERRCNT Byte 5: USECOUNT Byte 6: DEK Byte 7: ARA_COUNTER Byte 8: MINLEN	m	Object Parameters
86h	q	Definitions for Access Conditions	m	AC Definitions
8Bh	r	Definitions for Secure Messaging	o	SM Definitions
8Fh	s	Object Data	m	Object Data

m/o:	mandatory / optional
q:	Length of AC definitions in bytes
r:	Length of SM definitions in bytes
s:	Length of Object Data in bytes

This section describes each byte of

- **Object Address** and
- **Object Parameters**.

The other OCI parameters are described in the following sections:

- **AC Definitions** in section **2.2.4.3**,
- **SM Definitions** in section **2.5.2.2** and
- **Object Data** in the sections **2.4.1.2** and **0** for keys and in section **2.3** for TEST objects.

Object Class Byte The Object Class Byte is shown in table 2.3.1-2

Table 2.3.1-2 Object Class Byte

Bit No.								**Description**		
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**			
								Object Class Byte		
0								**BS Objects:**	fixed to ‘0’	
	p							**BS Objects:**	p=’0’	
		u	u	u				**BS Objects:**	uuu = Object Usage	Class Byte for kkk=’000’
								uuu=’000’:	TEST Object	(00h)
								uuu=’001’:	AUTH Object	(08h)
								uuu=’010’:	SM Object	(10h)
								uuu=’100’:	PSO Object	(20h)
					k	k	k	**BS Objects:**	kkk = Object Component	
								ALGO	*ALGO-ID*	*Meaning*
								kkk=’000’:		
								DES_ECB	(00h)	Single Component
								DES_CBC	(01h)	Single Component
								DES3_CBC	(03h)	Single Component
								Logical	(7Fh)	Single Component
								MAC	(80h)	Single Component
								iMAC	(81h)	Single Component
								MAC3	(82h)	Single Component
								iMAC3	(83h)	Single Component
								PIN (CHV)	(87h)	Single Component
								DSA (*)	(89h)	Prime P
								DSA_SHA-1 (*)	(C9h)	Prime P
								RSA	(08h)	Modulus N
								RSA_PURE	(0Ch)	Modulus N
								RSA_SIG	(88h)	Modulus N
								RSA_PURE_SIG	(8Ch)	Modulus N
								RSA_SIG_SHA-1	(C8h)	Modulus N
								RSA_PURE_SIG_SHA-1	(CCh)	Modulus N
								kkk=’001’:		
								DSA (*)	(89h)	Prime Q
								DSA_SHA-1 (*)	(C9h)	Prime Q
								RSA	(08h)	Exponent E or D
								RSA_PURE	(0Ch)	Exponent E or D
								RSA_SIG	(88h)	Exponent E or D
								RSA_PURE_SIG	(8Ch)	Exponent E or D
								RSA_SIG_SHA-1	(C8h)	Exponent E or D
								RSA_PURE_SIG_SHA-1	(CCh)	Exponent E or D
								kkk=’010’:		
								DSA (*)	(89h)	Base G
								DSA_SHA-1 (*)	(C9h)	Base G
								kkk=’011’:		
								DSA (*)	(89h)	Factor X or Y
								DSA_SHA-1 (*)	(C9h)	Factor X or Y)

(*): DSA algorithms are available only, if DSA_Package was loaded and activated before.

CardOS/M4 uses the Object Address (i.e. Object Class and Object ID) in order to search for an object, which is necessary for a command. When searching for a key, which may consist of different object components, CardOS/M4 searches for the first component (i.e. object component with kkk = ‘000’) always. CardOS/M4 uses the options of the component kkk=’000’ during usage of the belonging object only.

Note that a TEST object affects the *security status* of the DF, in which the TEST object is stored and in all child-DFs.

Object ID Table 2.3.1-3 shows the Object ID Byte

Table 2.3.1-3 Object ID Byte

Bit No.								Description
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
								Object ID Byte
v	v	v	v	v	v	v	v	vvvv vvvv = ‘1111 1111’ (FFh) not permitted other valid values depend on the type of object: *Object* — *Value Range* **BS Objects:** **TEST Object** used for: — 01h ... 7Fh EXTERNAL AUTHENTICATE SELECT FILE (i.e. for implicit logical test) VERIFY **AUTH Object** — 01h ... FEh **SM Object** — 01h ... FEh **PSO Object** — 01h ... FEh

For one object class all object IDs in one DF level must be unique.

For **BS objects** the object ID **can freely be defined** by the customer.

TEST objects reference the access rights they affect by their object ID. This means for instance, that a TEST object with object ID 07h affects the access right 07h in the *security status*. As 127 access rights exist in each DF the range of possible values for the TEST object IDs is from 01h to 7Fh.

All other BS objects do not affect the *security status* and therefore the range of values for the object IDs is from 01h to FEh. The value FFh is reserved for special purposes (e.g. if **FFh** is specified in the SM definitions of files/objects this means that **no** SM shall be used).

OPTIONS Byte Table 2.3.1-4 shows the OPTIONS Byte

Table 2.3.1-4 OPTIONS Byte

Bit No.								Description
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
								OPTIONS Byte
p								**TEST Object**: p='0': Right withdrawn after failed test p='1': Right remains after failed test **SM Object**: p='0' Using **no** random number with SM p='1' Using random number with SM **Other Objects**: no meaning (shall be set to '0')
	n							**EnableBit n** for Generate-Bit (see FLAGS Byte, g Bit) n='1' GenerateBit enabled n='0' GenerateBit disabled
		q						**BS Objects** (depending on used algorithm): q='0' Last or single Object Component q='1' Object Component following
			0					Must be set to '0'
				r				r='0' Object will be passed on (i.e. object can be used in the DF where it was installed and in all child-DFs.) r='1' Object will not be passed on (i.e. object can only be used in the DF where it was installed.)
					0			Shall be set to '0'
						t	t	**BS Objects:** The tt settings depend on the BS object and the action to be performed with the BS object:

Commands	**Possible tt values**
EXTERNAL AUTHENTICATE [TEST-MAC_Key]	01, 11
EXTERNAL AUTHENTICATE [TEST-Public_Key]	01
INTERNAL AUTHENTICATE [AUTH-MAC_Key]	10, 11
INTERNAL AUTHENTICATE [AUTH-Private_Key]	10
PSO_CCC [PSO-MAC_Key]	10, 11
PSO_CDS [PSO-Private_Key]	10
PSO_DEC [PSO-DES_Key]	10, 11
PSO_DEC [PSO-Private_Key]	10
PSO_ENC [PSO-DES_Key]	01, 11
PSO_ENC [PSO-Public_Key]	01
PSO_H [PSO-Hash_Key]	don't care
PSO_VCC [PSO-MAC_Key]	01, 11
PSO_VDS [PSO-Public_Key]	01
VERIFY [PIN TEST]	10, 11
SM (Mode xCh only)	
Command-SM_ENC	10, 11
Command-SM_SIG	10, 11
Command-SM_ENC_SIG	10, 11
Response-SM_ENC	01, 11
Response-SM_SIG	01, 11
Response-SM_ENC_SIG	01, 11

OPTIONS
Bit p

This bit is relevant for TEST and SM objects only and shall be set to ‘0’ otherwise.

For TEST objects p=’0’ means, that an access right shall be withdrawn after a failed security test (should be the default setting). If p=’1’ an access right, which was granted before during a preceding security test, will not be withdrawn after a failed security test (This setting is needed for GSM-like behavior of the smart card).

For SM objects p=’0’ means, that SM_SIG, or SM_ENC_SIG shall not use a random number for MAC calculation (see sections 2.5.2.5, 2.5.2.6, 2.5.2.9 and 2.5.2.10).

OPTIONS
Bit n

The EnableBit n activates the GenerateBit g in the FLAGS byte.

- If n=1 and g=1, then the BS object data can be overwritten **once** using the GENERATE KEY PAIR command.
- If n=1 and g=0, then the BS object data **cannot** be overwritten using the GENERATE KEY PAIR command.
- If n=0 and g=1 or g=0, then the BS object data can **always** be overwritten using the GENERATE KEY PAIR command. In order to limit overwriting of the BS object data other standardized mechanisms of CardOS/M4 (e.g. AC GENKEY of the BS object) must be used.
- If g=1, only a generation of a key pair is possible. After the generation, the GenerateBit is reset implicitly by the command.

OPTIONS
Bit q

For all BS objects q=’0’ means, that the object is the only or last object component. For all symmetric key objects, PIN test objects and logical test objects q=’0’. Setting q=’1’ specifies, that an object component is following. A asymmetric key is only usable, if all key components are installed in the smart card. The end of a component chain is shown to CardOS/M4 by q=’0’ for the last component. During installation of the object components using PUT DATA_OCI the components need not to be installed in rising order.

OPTIONS
Bit r

For all objects r=’0’ means, that the object is passed on to all child-DFs. Then the backtracking mechanism can find an object not only in the DF, where the object is installed, but also in all child-DFs. If r=’1’ the object can be found only, if the backtracking mechanism starts in the DF, where the object is installed.

OPTIONS
Bits tt

For all BS objects representing keys or PINs the bits tt are relevant according to table 2.3.1-4. Unlike PINs and symmetric keys, where different tt-settings are possible, for each asymmetric key only one tt-setting is permitted.

FLAGS Byte Table 2.3.1-5 shows the FLAGS Byte

Table 2.3.1-5 FLAGS Byte

Bit No.								Description
7	6	5	4	3	2	1	0	
								FLAGS Byte
x								**DeathBit x:** x='0' CardOS/M4 does *not* reach life cycle phase DEATH if USECOUNTER reaches 00h. x='1' CardOS/M4 reaches life cycle phase DEATH if USECOUNTER reaches 00h and every DeathEnableBit in the complete DF/MF hierarchy is set.
	u							**TEST Objects**: u='0' Referenced right **not** granted during selection of the DF where the object is installed (shall be default). u='1' Referenced right granted during selection of the DF where the object is installed (e.g. for GSM). **Other Objects**: no meaning (shall be set to '0')
		g						**GenerateBit g:** The GenerateBit g is active only if the EnableBit n in the OPTIONS Byte is set to '1' g='1' The key can be generated (i.e. overwritten) using the GENERATE KEY PAIR command. After successful execution of GENERATE KEY PAIR GenerateBit g is set to '0'. g='0' The key cannot be overwritten using the GENERATE KEY PAIR command.
			v					Must be set to v='0'
				x	x	x	x	**BS Objects**: MaxErrorCounter: The CurrentErrorCounter (see ERRCNT byte) is set to the value of the MaxErrorCounter after a successful test. MaxErrorCounter='0000' deactivates the CurrentErrorCounter.

FLAGS DeathBit x If x=0 then CardOS/M4 does not reach life cycle phase *DEATH*, if USECOUNTER reaches 00h.

If x=1 then CardOS/M4 reaches life cycle phase *DEATH* (i.e. the smart card will be blocked irreversibly), if

- USECOUNTER reaches 00h **and**
- the DeathEnableBit in the complete DF hierarchy including MF is set**.**

This behavior applies to CardOS/M4.01 always and to CardOS/M4.00, if the so-called UseCounter_Package was loaded and activated before. Otherwise CardOS/M4.00 does not reach life cycle phase *DEATH*.

The motivation for this mechanism is, that after under certain conditions the card shall be made unusable.
The active part is taken by a BS object with the DeathBit set. As soon as the USECOUNT reaches zero, a try is made to switch the card to the DEATH phase. As this is a quite unusual requirement, no BS object should have the DeathBit set without definitive reason.
Now the passive part comes into focus. To prevent the card being made unusable without consent of application providers or card issuer each DF on the current selection path up to and including the MF must agree with the switch to DEATH by a set DeathEnableBit. If all DFs agree, the switch is made, otherwise nothing happens (beyond the BS Object being unusable due to the USECOUNT is zero).
So a typical configuration is to allow the switch on MF level (in the case, that a future application needs it) and prevent the switch on any completely specified application not needing it by letting the bit cleared.

FLAGS
Bit u

Only for TEST objects u has a meaning. For other BS objects the bit shall be set to ‘0’. Bit u=’0’ means, that the access right referenced by the TEST object ID is **not** granted automatically during selection of the DF, where the TEST object is installed. If bit u=’1’ the referenced access right is granted automatically (i.e. without performing a security test) during the DF selection.

FLAGS
GenerateBit g

Depending on the EnableBit n in the OPTIONS byte the following applies using the GenerateBit g:

- If n=1 and g=1, then the BS object data can be overwritten **once** using the GENERATE KEY PAIR command.
- If n=1 and g=0, then the BS object data **cannot** be overwritten using the GENERATE KEY PAIR command.
- If n=0 and g=1 or g=0, then the BS object data can **always** be overwritten using the GENERATE KEY PAIR command. In order to limit overwriting of the BS object data other standardized mechanisms of CardOS/M4 (e.g. AC GENKEY of the BS object) must be used.
- If g=1, only a generation of a key pair is possible. After the generation, the GenerateBit is reset implicitly by the command.

FLAGS
MaxErrorCounter

The MaxErrorCounter bits xxxx are relevant for all PIN TEST objects, for all TEST key objects (i.e. keys, which are used for a C/R test) and for all SM objects, which are used for MACing (i.e. MAC-verification).

The bits xxxx specify the maximum number of unsuccessful trails for a security test/MAC-verification until the object is blocked.

After a successful execution of the security test/MAC-verification the CurrentErrorCounter in the ERRCNT byte (see below) is set to MaxErrorCounter.

If xxxx = ‘0000’ then the CurrentErrorCounter of the ERRCNT byte is disabled. This means, that the number of unsuccessful trials for the security test/MAC-verification is not limited.

ALGO Byte Table 2.3.1-6 shows the ALGO Byte

Table 2.3.1-6 ALGO Byte

Bit No.								Description	
7	6	5	4	3	2	1	0		
								ALGO Byte	
y	y	y	y	y	y	y	y	**BS Objects**:	The byte specifies the following algorithms:
								ALGO-ID	*ALGO*
								00h	DES_ECB
								01h	DES_CBC
								03h	DES3_CBC
								08h	RSA
								0Ch	RSA_PURE
								7Fh	Logical
								80h	MAC
								81h	iMAC
								82h	MAC3
								83h	iMAC3
								87h	PIN (CHV)
								88h	RSA_SIG
								89h	DSA (*)
								8Ch	RSA_PURE_SIG
								C8h	RSA_SIG_SHA-1
								C9h	DSA_SHA-1 (*)
								CCh	RSA_PURE_SIG_SHA-1
								CFh	SHA-1

(*): DSA algorithms are available only, if DSA_Package was loaded and activated before.

ALGO Byte For all BS objects this byte specifies, which algorithm shall be used with the BS object.

ERRCNT Byte Table 2.3.1-7 shows the ERRCNT Byte

Table 2.3.1-7 ERRCNT Byte

Bit No.								Description
7	6	5	4	3	2	1	0	
								ERRCNT Byte
r								**PureResetRetry r:** **(applies to CardOS/M4.01 only)** r='0' RESET RETRY COUNTER is allowed to change the PIN r='1' RESET RETRY COUNTER is *not* allowed to change the PIN
	w							**BS Objects**: w='0': CurrentErrorCounter decreased **after** test execution (POST-Decrement) w='1': CurrentErrorCounter decreased **before** test execution (PRE-Decrement)
		z						**BS Objects**: z='0': CurrentErrorCounter set to value of MaxErrorCounter after successful test (default). z='1': CurrentErrorCounter **not** set to value of MaxErrorCounter after successful test.
			m					**TerminateBit m:** **(applies to CardOS/M4.01 only)** m='1' If CurrentErrorCounter reaches '0000' then the DF which contains the BS object is terminated, i.e. an "implicit" TERMINATE DF operation is performed. This applies to keys using a CurrentErrorCounter (e.g. SM keys, C/R test keys,...) m='0' If the CurrentErrorCounter reaches '0000' the DF which contains the BS object is **not** terminated.
				x	x	x	x	**BS Objects**: CurrentErrorCounter: The CurrentErrorCounter is set to the value of the MaxErrorCounter (see FLAGS byte) after a successful test and is decreased after a failed test. If CurrentErrorCounter='0000', then the BS object is blocked.

ERRCNT Byte For all BS objects used for a security test (except logical tests) or MAC-verification for SM (i.e. PIN TEST objects, C/R TEST objects and SM objects for MAC-verification) the ERRCNT byte is relevant.

ERRCNT PureResetRetry r For CardOS/M4.00 this bit has no meaning and is don't care. For compatibility reasons it is recommended that r='0' for CardOS/M4.00 applications.

In CardOS/M4.01 the bit r='0' specifies, that with the RESET RETRY COUNTER command the reference data of a PIN can be updated with the New_CHV_data.

If r='1', that with the RESET RETRY COUNTER command the reference data of a PIN cannot be updated with the New_CHV_data. The command is rejected.

ERRCNT
Bit w

The bit w='0' specifies, that the CurrentErrorCounter of the object is decreased **after** an unsuccessful execution of the security test/MAC-verification. With this setting EEPROM is only written, if the security test/MAC-verification fails. Note: This setting is faster during execution of a security test but less secure than setting w='1'!

If w='1', then the CurrentErrorCounter is decreased **before** execution of the security test/MAC-verification. This setting is more secure than the w='0' setting but **each time** the security test/MAC-verification starts EEPROM will be written. This may lead to worn out smart cards for security tests/MAC-verifications used extremely often (i.e. here the number of EEPROM write cycles of the chip must be taken into account).

ERRCNT
TerminateBit m

For CardOS/M4.00 this bit has no meaning and is don't care. For compatibility reasons it is recommended that s='0' for CardOS/M4.00 applications.

For CardOS/M4.01 the TerminateBit m='1' specifies, that the DF, which contains the BS object, is terminated when the CurrentErrorCounter reaches '0000'. If m='0', the DF is **not** terminated when the CurrentErrorCounter reaches '0000'.

This mechanism only applies to keys using the CurrentErrorCounter (e.g. SM keys, C/R test keys, ...). The termination is equivalent to an "implicit" TERMINATE DF operation according to ISO 7816-9.

ERRCNT
Bit z

The bit z='0' (i.e. default setting) specifies, that the CurrentErrorCounter is set to the value of the MaxErrorCounter of the FLAGS byte after successful execution of the security test/MAC-verification.

If z='1', then the CurrentErrorCounter is **not** set to the value of the MaxErrorCounter of the FLAGS byte after successful execution of the security test/MAC-verification. This means, that the CurrentErrorCounter cannot be increased.

ERRCNT
CurrentErrorCounter

The CurrentErrorCounter represents the maximum number of unsuccessful executions of a security test/MAC-verification until the object is blocked (i.e. CurrentErrorCounter = '0000').

After an **unsuccessful** execution of a security test/MAC-verification CurrentErrorCounter is **decreased**.

After a successful execution of a security test/MAC-verification and if z=’0’, the CurrentErrorCounter is set to the value of the MaxErrorCounter in the FLAGS byte.

USECOUNT Byte Table 2.3.1-8 shows the USECOUNT Byte

Table 2.3.1-8 USECOUNT Byte

Bit No.								**Description**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
								USECOUNT Byte
y	y	y	y	y	y	y	y	**BS Objects:** The byte represents the **UseCounter**, which is decreased each time the object is used. So the number of usages of an object can be limited absolutely. UseCounter=00h: Object blocked UseCounter=FFh Number of Object Usages not limited.

Using the UseCounter mechanism the absolute number of usages for an BS object (e.g. a key) can be limited. For all BS objects except logical TEST objects the UseCounter represents a counter, which is decreased each time the object is used successfully.

This means for TEST objects, that the UseCounter is **decreased** each time the test using this TEST object was **successful**. This behavior applies to CardOS/M4.01 always and to CardOS/M4.00, if the so-called UseCounter_Package was loaded and activated before. Otherwise CardOS/M4.00 decreases the UseCounter even if the test with the TEST object was unsuccessful.

The UseCounter is a predecrement counter.

Note that the smart card will be blocked irreversibly (i.e. reaches life cycle phase *DEATH*) if

- the FLAGS DeathBit x='1' and
- USECOUNTER reaches 00h and
- the DeathEnableBit in the complete DF hierarchy including MF is set**.**

This behavior applies to CardOS/M4.01 always and to CardOS/M4.00, if the so-called UseCounter_Package was loaded and activated before. Otherwise CardOS/M4.00 does not reach life cycle phase *DEATH*.

DEK Byte

Table 2.3.1-9 shows the DEK Byte

Table 2.3.1-9 DEK Byte

Bit No.								Description
7	6	5	4	3	2	1	0	
								DEK Byte
k	k	k	k	k	k	k	k	**must** be set to FFh

The DEK byte **must** be set to FFh.

ARA_COUNTER Byte

Table 2.3.1-10 shows the ARA_COUNTER Byte

Table 2.3.1-10 ARA_COUNTER Byte

Bit No.								Description
7	6	5	4	3	2	1	0	
								ARA_COUNTER Byte
v	v	v	v	v	v	v	v	**TEST Objects:** This byte limits the usage of a granted access right by a counter, which is decreased each time the access right is used (e.g. as AC for file access, USE of SM keys,...) ARA_COUNTER = 00h or = FFh: unlimited usage ARA_COUNTER > 00h and < FFh: limited usage **Other Objects:** no meaning (shall be set to 00h)

MINLEN Byte

Table 2.3.1-11 shows the MINLEN Byte

Table 2.3.1-11 MINLEN Byte

Bit No.								Description
7	6	5	4	3	2	1	0	
								MINLEN Byte
r	r	r	r	r	r	r	r	**TEST C/R or SM SIG_Objects**: MINLEN is the minimum length of the random number to be used. **PIN TEST** object: MINLEN is the minimum length of the PIN data. **Other Objects**: CardOS/M4.01: shall be set to 00h CardOS/M4.00: shall be set to 00h or eventually to the length of the object data (see CHANGE REFERENCE DATA)

MINLEN is interpreted as unsigned integer and specifies

- the **minimum length of the random number** to be used for the belonging **C/R test**,
- the **minimum length of the random number** to be used for the SIGning procedure of **SM** with the SIG_objects or
- the **minimum length of the PIN data** to be used for the belonging **PIN test**.

The MINLEN byte is relevant for **SM objects** used for SIGning. For the belonging SM SIG_objects the MINLEN must be a **multiple of 8**. Otherwise SM leads to undocumented results.
For **PIN TEST** objects the valid length of the PIN data must in between MINLEN and the length of the PIN object data, which are installed via the PUT DATA_OCI command.

For other objects MINLEN shall be set to **00h** or for CardOS/M4.00 eventually to the length of the object data installed during PUT DATA_OCI (for details see CHANGE REFERENCE DATA).

2.3.2 The Backtracking Mechanism

General

In order to search for any objects in the filesystem, CardOS/M4 uses the so-called **backtracking mechanism**. The backtracking mechanism searches for the objects using their Object Address (i.e. Object Class and optionally Object ID, if present).

When the backtracking mechanism finds an object, CardOS/M4 checks, whether the backtracking mechanism is permitted for the found object (see OPTIONS byte (bit r) in section 2.3.1). If backtracking is not permitted for the object, an error occurs. For the different CardOS/M4.0 versions the search is performed in the following order and is finished, when the object is found:

CardOS/M4.00:

1st CardOS/M4.00 searches for an object with the specified object address in the *current DF*.

2nd The OS now selects the parent-DF **of the *current DF*** and searches for the object there.

3rd The second step is repeated until the OS searched in the MF as well. If no object was found in the MF, then an error occurs.

CardOS/M4.01:

In order to improve security the behavior of CardOS/M4.01 was modified slightly.

1st If the object to be searched for is **not** referenced by another object (i.e. the object to be searched for is **the first** of a possible object chain), CardOS/M4.01 searches for an object with the specified object address in the ***current DF***.
If the object to be searched for **is referenced by another object** (i.e. the object to be searched for is **not the first** of a possible object chain), CardOS/M4.01 starts the search from **the DF containing the object referencing the object to be searched for**.

2nd The OS now selects the parent-DF and searches for the object there.

3rd The second step is repeated until the OS searched in the MF as well. If no object was found in the MF, then an error occurs.

Example general

For the paranoid the following example shows, how security of CardOS/M4.00 could be attacked, if the attacker is an official application owner inside the multiapplication smart card and has special possibilities to manipulate security settings, whereas this **attack is not possible with CardOS/M4.01** because of the modified backtracking behavior.

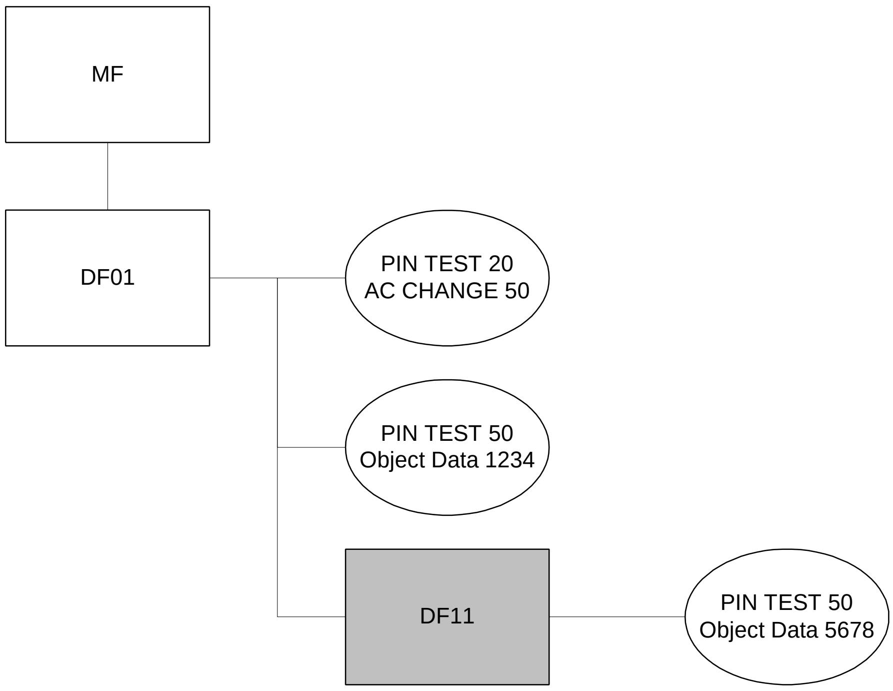
For **CardOS/M4.00** note that the **attack can be prevented by reducing** the security design possiblities of the official application owner inside the multiapplication smart card.

For **CardOS/M4.01** the **attack is not possible** because of the modified backtracking mechanism **without reducing** the security design possiblities of the official application owner inside the multiapplication smart card.

The following hierarchy and objects exists:

- DF01 under MF with a
    - PIN TEST with object ID “20” and AC CHANGE “50”
    - PIN TEST with object ID “50” and object data “1234”
    - DF11
- DF11 under DF01 with a
    - PIN TEST with object ID “50” and object data “5678”

The application owner of the multiapplication smart card has the possiblity to create child-DFs (i.e. DF11) and therefore child-objects (e.g. object “50” in DF11).



Example for CardOS/M4.00: DF11 is the ***current DF***.

A CHANGE REFERENCE DATA command is send to change the PIN TEST object with ID 20 in DF01. Old_Data is present in the CHANGE REFERENCE DATA command, so the AC CHANGE right (50) has to be granted with an implicit verification:

- After an unsuccessful search in DF11 for a PIN TEST with object ID 20 the parent DF01 will then be searched.
- The PIN TEST object 20 will be found in DF01.
- After that the referenced PIN TEST object 50 will be found in DF11 because search starts with the ***current DF***.
- If Old_Data is “5678” the command is executed correctly and the smart card will return SW1/SW2=9000h.
- If the value of Old_Data is “1234”, the authentication failes and the smart card returns SW1/SW2=6300h, because once the PIN TEST object with ID 20 has been found in DF01 the child DF11 will be searched again (as DF11 is the ***current DF***).

Example for CardOS/M4.01: DF11 is the ***current DF***.

A CHANGE REFERENCE DATA command is send to change the PIN TEST object with ID 20 in DF01. Old_Data is present in the CHANGE REFERENCE DATA command, so the AC CHANGE right (50) has to be granted with an implicit verification:

- After an unsuccessful search in DF11 for a PIN TEST with object ID 20 the parent DF01 will then be searched.
- The PIN TEST object 20 will be found in DF01. Therefore DF01 is the DF containing the object referencing object "50" (i.e the DF where the next object of the object chain shall be searched for the next step).
- After that the referenced PIN TEST object 50 will be found in DF01 because search starts with the **DF containing the object** (i.e. object 20 in DF01) **referencing the object to be searched for** (i.e. an object with ID=50)**.**
- If Old_Data is "5678", the authentication failes and the smart card returns SW1/SW2=6300h, because once the PIN TEST object with ID 20 has been found in DF01 the next object in the object chain will be searched starting from DF01 (as DF01 is the **DF containing the object** (i.e. object 20 in DF01) **referencing the object to be searched for** (i.e. an object with ID=50)).
- If the value of Old_Data is "1234", the command is executed correctly and the smart card will return SW1/SW2=9000h.

This example shows that with CardOS/M4.01 it is **not** possible to attack the security design of a hierarchically higher DF level by means of the backtracking mechanism even if it is allowed to create DFs below this DF level. This means that for CardOS/M4.01 it is **not** possible to undermine the security design of a hierarchically higher security setting (i.e. in a hierarchically higher DF level) by creating security objects in hierarchically lower DFs.

2.4 Cryptographic Algorithms

CardOS/M4 supports different algorithms for operation. All algorithms are separated depending on the cases whether a symmetric or asymmetric method is used and depending on their usage in the smart card. Table 2.4-1 shows the different algorithms, their methods, their usage and - if necessary - the belonging key type and the belonging padding method.

Table 2.4-1 Overview of Cryptographic Algorithms of CardOS/M4

Crypto-graphic Method	**Usage**	**Algorithm Type**	**Used Key**	**Possible Algorithms (Calculation Method)**	**ALGO Byte** (2)	**Padding Method(s) hashing / non-hashing** (3)
Symmetric	Encryption / Decryption	DES	DES_Key	DES ECB	00h	- / ISO-Padding
				DES CBC	01h	- / ISO-Padding
				DES3 CBC	03h	- / ISO-Padding
	MACing	MAC	MAC_Key	MAC	80h	ANSI-Padding / -
				iMAC	81h	ISO-Padding / -
				MAC3	82h	ANSI-Padding / -
				iMAC3	83h	ISO-Padding / -
Asymmetric	Encryption / Decryption	RSA for / En-/Decryption	Public_Key / Private_Key	RSA	08h	- / PKCS#1-BT2-Padding
				RSA_PURE	0Ch	- / Leading-Zeroes-Padding (4)
	Signature **without** hashing	RSA/DSA for Signature **without** Hashing	Public_Key / Private_Key	RSA_SIG	88h	- / PKCS#1-BT1-Padding
				DSA (1)	89h	- / DSA-Padding
				RSA_PURE_SIG	8Ch	- / Leading-Zeroes-Padding (4)
	Signature **with** hashing	RSA/DSA for Signature **with** Hashing	Public_Key / Private_Key	RSA_SIG_SHA-1	C8h	SHA-1-Padding / PKCS#1-BT1-Padding
				DSA_SHA-1 (1)	C9h	SHA-1-Padding / DSA-Padding
				RSA_PURE_SIG_SHA-1	CCh	SHA-1-Padding / Leading-Zeroes-Padding (4)
	Hashing	Hashing	no Key	SHA-1	CFh	SHA-1-Padding / -

(1): DSA algorithms are available only, if DSA_Package was loaded and activated before.
(2): ALGO Byte in the Object Parameters (see section Objects or section PUT DATA_OCI)
(3): Padding method for the hashing part / non-hashing part of the algorithm
(4): CardOS/M4.00 implements the padding method PKCS#1-BT0-Padding, i.e. PKCS#1 padding with block type 0. If the LeadingZero_Package is loaded, this method is replaced by Leading-Zeroes-Padding.
CardOS/M4.01 implements the padding method Leading-Zeroes-Padding.

All cryptographic algorithms consist of a **fixed calculation method** and a **fixed padding method**. This means that using an ALGO byte for an object fixes the calculation method and the padding to be used.

Except the hashing algorithm SHA-1 all algorithms need a key. Nevertheless for the PSO_H command the HASH component of the CSE must reference a so-called PSO-Hash_Key, which is set using the PUT DATA_OCI command but is no real key. The PSO object representing a PSO-Hash_Key has a dummy byte in the object data.

Depending on the operation of the command, which uses one or several keys, (i.e. depending on the algorithm usage) different keys, which may reside in different BS objects, must be used. Table 2.4-2 shows, which keys must be used for which command or operation and in which BS object the keys must reside.

DES algorithms and MAC algorithms are DES-based algorithms.

Table 2.4-2 Usage of Keys with different Commands and Operations

Command / Operation	Necessary BS Object	Possible Key(s)
EXTERNAL AUTHENTICATE	TEST Object	MAC_Key
		Public_Key
INTERNAL AUTHENTICATE	AUTH Object	MAC_Key
		Private_Key
PSO_CCC	PSO Object	MAC_Key
PSO_CDS	PSO Object	Private_Key
PSO_DEC	PSO Object	DES_Key
		Private_Key
PSO_ENC	PSO Object	DES_Key
		Public_Key
PSO_H	PSO Object	Hash_Key
PSO_VCC	PSO Object	MAC_Key
PSO_VDS	PSO Object	Public_Key
SM_ENC (SM encryption)	SM Object	DES_Key
SM_SIG (SM signature)	SM Object	MAC_Key

Note: A Hash_Key is a BS object with a hash algorithm and a dummy byte in the object data.

2.4.1 Symmetric Algorithms

Description This section describes the symmetric algorithms of CardOS/M4, which are all DES-based algorithms. Note that DES denotes the standard DES algorithm in encryption mode and DES-1 denotes the standard DES algorithm in decryption mode.

2.4.1.1 Symmetric Padding Methods

Description CardOS/M4 supports symmetric and asymmetric padding methods. This section describes the symmetric padding methods.

The input for the padding methods is denoted as **Input Data**.

The result of the padding method is denoted as **Encryption Block**.

Generally for symmetric padding methods the padding string is added at the end of the input string (see symmetric padding format in table 2.4.1.1-1).

Table 2.4.1.1-1 Symmetric Padding Format

Input Data	**Padding String**
u bytes	p bytes
k=(u+p) bytes	
Encryption Block	

u	Length of Input Data
k	Length of Encryption Block
p	Length of Padding String, which is governed by the following equations:
	k ≡ n*m (1), n must be a positive integer
	p = k-u (2)
Note	All lengths are denoted in bytes.

Table 2.4.1.1-2 shows the symmetric padding methods used in CardOS/M4.

Table 2.4.1.1-2 Symmetric Padding Methods

Padding Method	**Block Length m** (*)	**Padding String**	**Minimum ... Maximum Length of Padding String**
ISO-Padding	8 bytes	80h 00h ... 00h	1 ... 8
ANSI-Padding	8 bytes	00h 00h ... 00h	0 ... 7

m Block Length of DES-based algorithms

As a result of table 2.4.1.1-2 for ISO-Padding a padding string must always exist.

For ANSI-Padding no padding string exists, if the length of the input string using ANSI-Padding is a multiple of 8 bytes.

2.4.1.2 Symmetric Key Formats

Description

CardOS/M4 provides different key formats for the different symmetric algorithms.

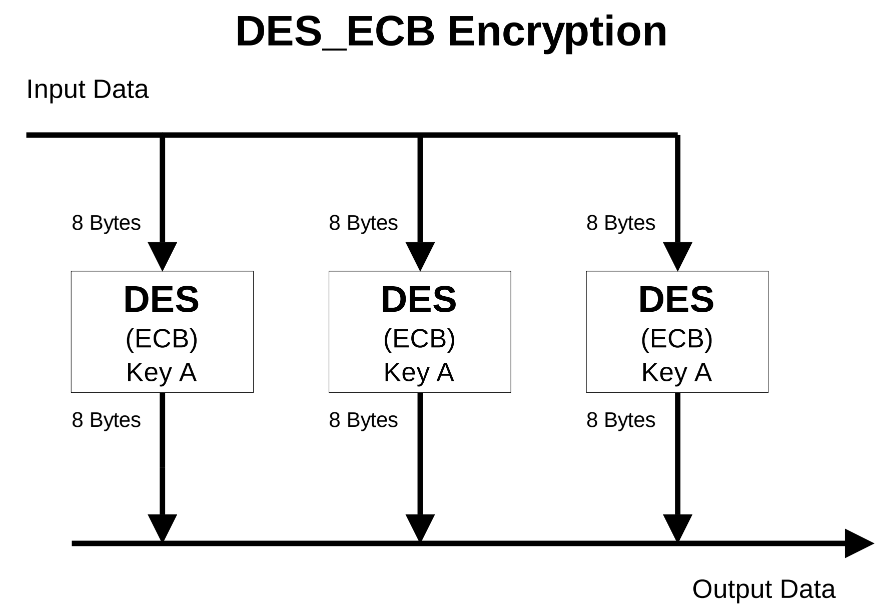
These formats are relevant for the Object Data (TAG=8Fh), which must be specified during installation or overwriting of a key using the PUT DATA_OCI command. The possible key formats for symmetric algorithms are summarized in table 2.4.1.2-1.

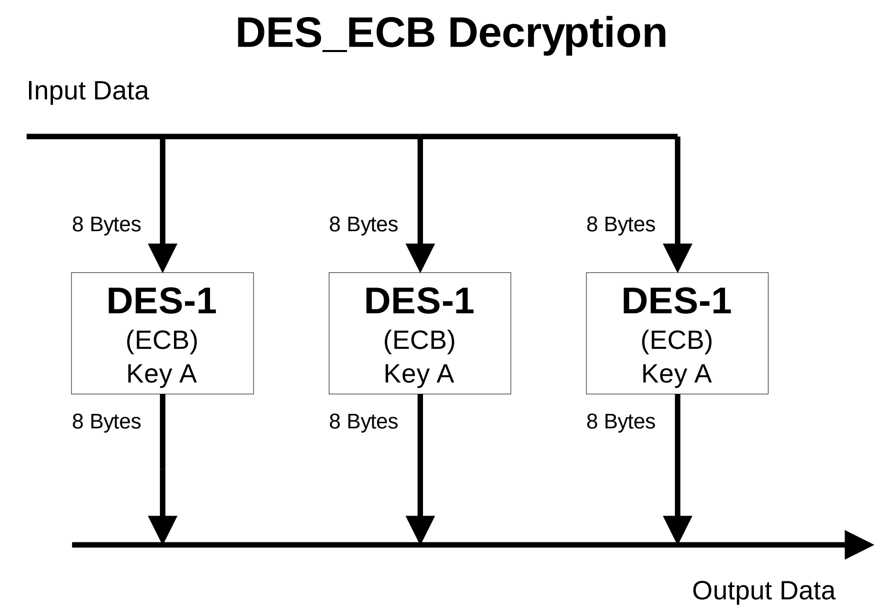
Table 2.4.1.2-1 Possible Key Formats for Symmetric Algorithms

Algorithm	**ALGO**	**Padding**	**Length of OCI (TAG=8Fh)**	**Value Field of OCI (TAG=8Fh)** Byte 1 ... Byte n (i.e. Byte 1 is leftmost)				**Comment**
				Meaning for Bytes 1 .. 8	**Meaning for Bytes 9 .. 16**	**Meaning for Bytes 17 .. 24**	**Meaning for Bytes 25 .. 32**	
DES_ECB	00h	ISO	08h	DES_Key A	-	-	-	-
DES_CBC	01h	ISO	08h	DES_Key A	-	-	-	Initial Vector = 0
			10h	DES_Key A	Initial Vector	-	-	-
DES3_CBC	03h	ISO	10h	DES_Key A	DES_Key B	-	-	Initial Vector = 0
			18h	DES_Key A	DES_Key B	DES_Key C	-	Initial Vector = 0
			20h	DES_Key A	DES_Key B	DES_Key C	Initial Vector	-
MAC	80h	ANSI	08h	MAC_Key A	-	-	-	Initial Vector = 0
			10h	MAC_Key A	Initial Vector	-	-	-
iMAC	81h	ISO	08h	MAC_Key A	-	-	-	Initial Vector = 0
			10h	MAC_Key A	Initial Vector	-	-	-
MAC3	82h	ANSI	10h	MAC_Key A	MAC_Key B	-	-	Initial Vector = 0
			18h	MAC_Key A	MAC_Key B	MAC_Key C	-	Initial Vector = 0
			20h	MAC_Key A	MAC_Key B	MAC_Key C	Initial Vector	-
iMAC3	83h	ISO	10h	MAC_Key A	MAC_Key B	-	-	Initial Vector = 0
			18h	MAC_Key A	MAC_Key B	MAC_Key C	-	Initial Vector = 0
			20h	MAC_Key A	MAC_Key B	MAC_Key C	Initial Vector	-

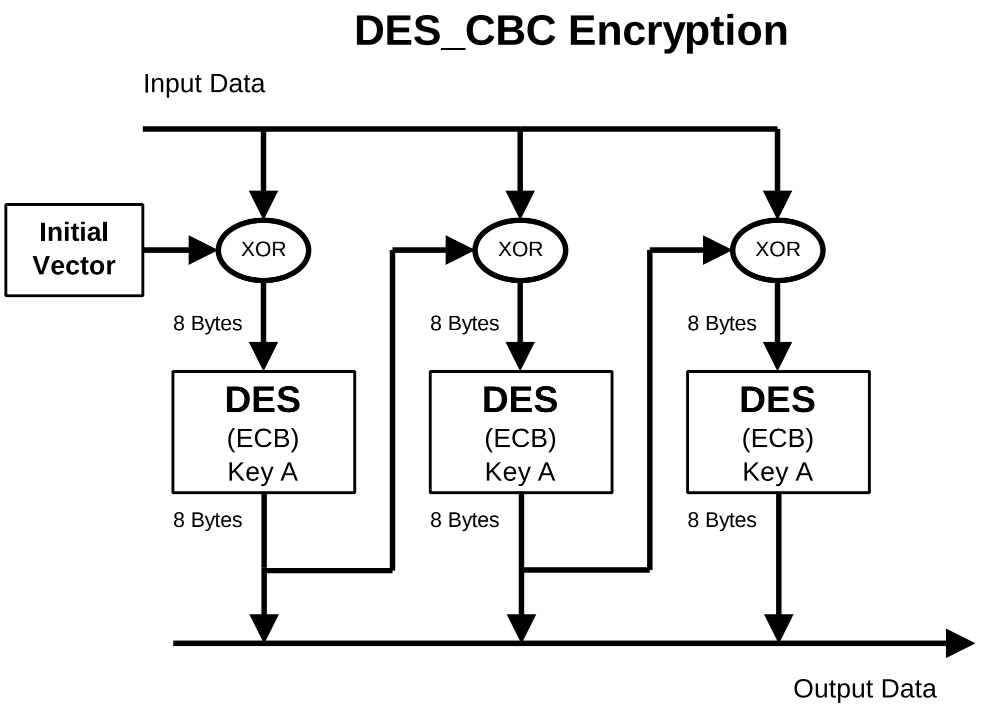
Initial Vector = 0 means, that all 8 bytes of Initial Vector are set to 00h.

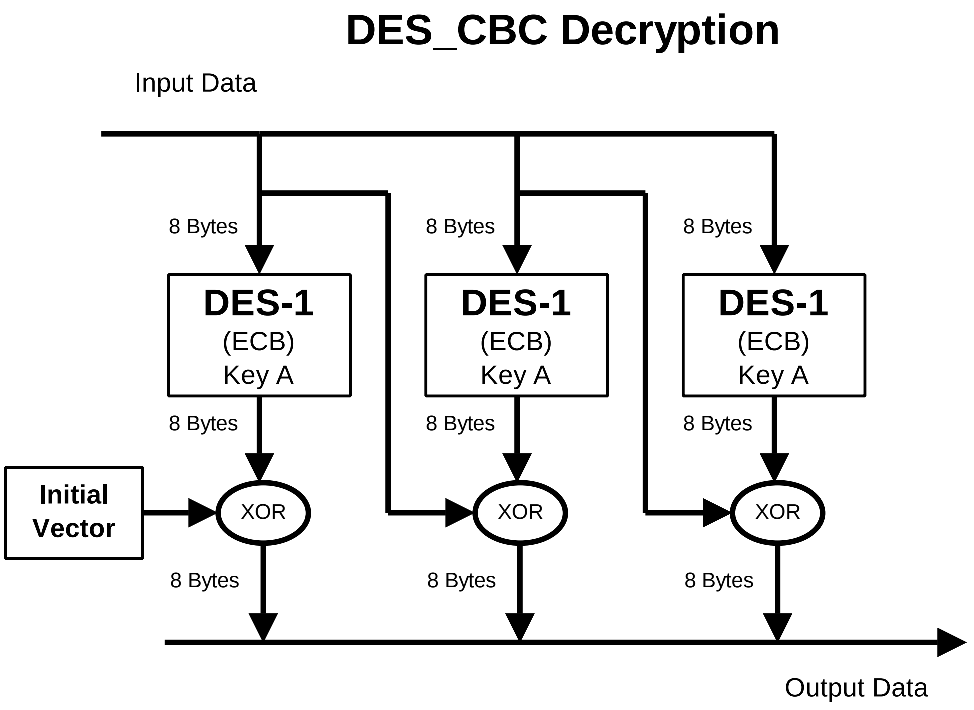
2.4.1.3 DES_ECB


DES_ECB Encryption
Input Data
8 Bytes
8 Bytes
8 Bytes
DES (ECB) Key A
DES (ECB) Key A
DES (ECB) Key A
8 Bytes
8 Bytes
8 Bytes
Output Data

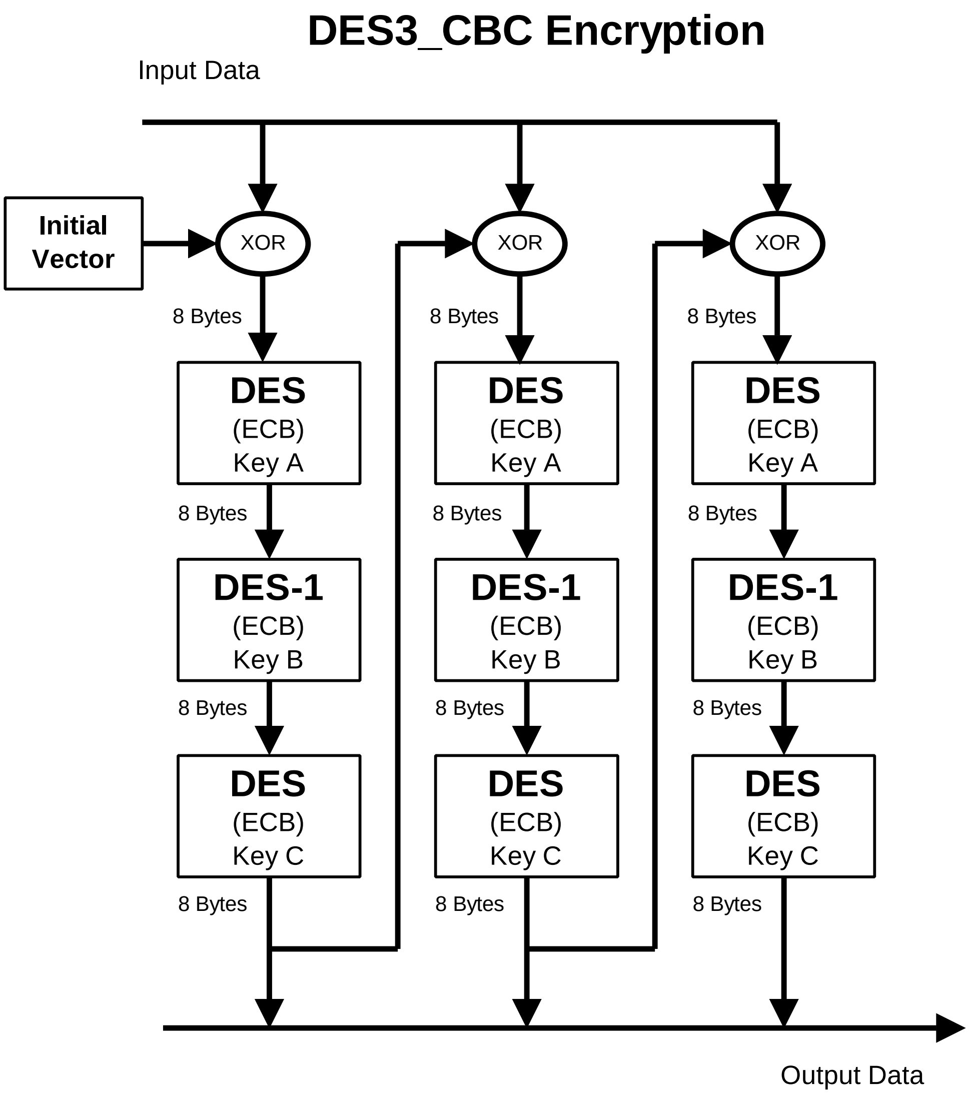

DES_ECB Decryption
Input Data
8 Bytes
8 Bytes
8 Bytes
DES-1 (ECB) Key A
DES-1 (ECB) Key A
DES-1 (ECB) Key A
8 Bytes
8 Bytes
8 Bytes
Output Data

2.4.1.4 DES_CBC

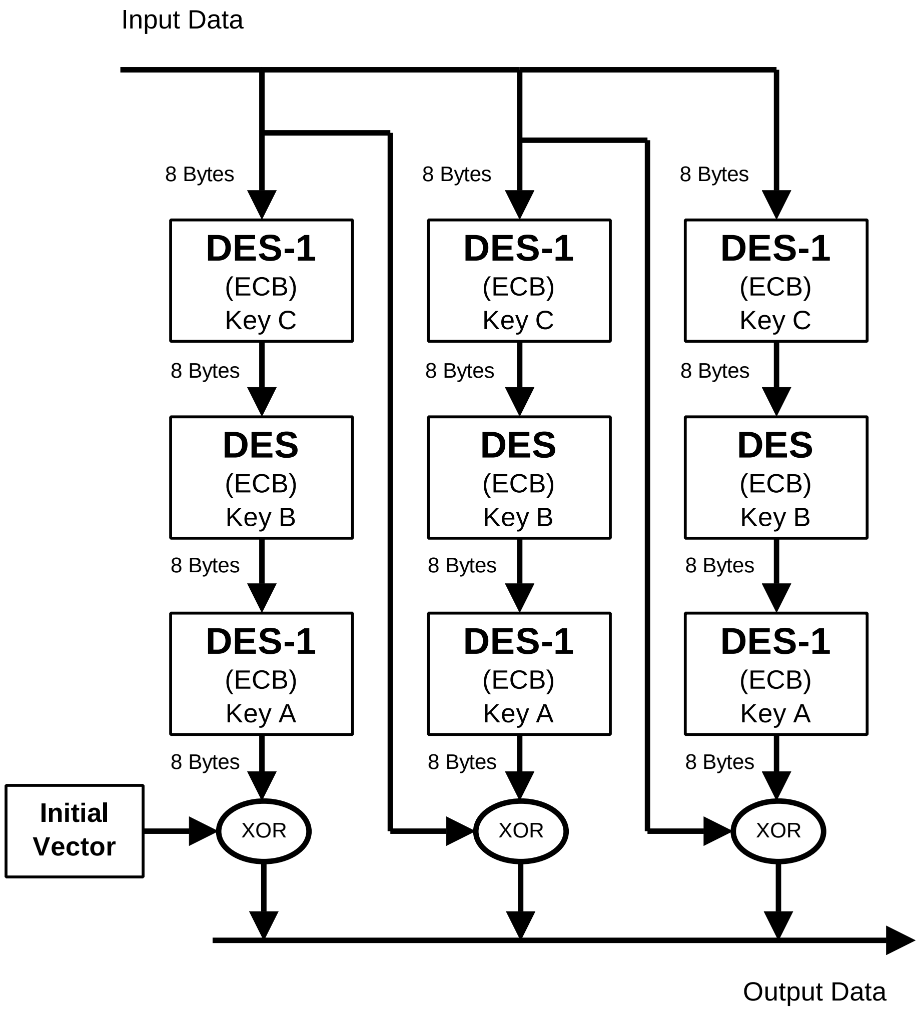

DES_CBC Encryption
Input Data
Initial Vector
XOR
XOR
XOR
8 Bytes
8 Bytes
8 Bytes
DES (ECB) Key A
DES (ECB) Key A
DES (ECB) Key A
8 Bytes
8 Bytes
8 Bytes
Output Data


DES_CBC Decryption
Input Data
8 Bytes
8 Bytes
8 Bytes
DES-1 (ECB) Key A
DES-1 (ECB) Key A
DES-1 (ECB) Key A
8 Bytes
8 Bytes
8 Bytes
Initial Vector
XOR
XOR
XOR
8 Bytes
8 Bytes
8 Bytes
Output Data

2.4.1.5 DES3_CBC

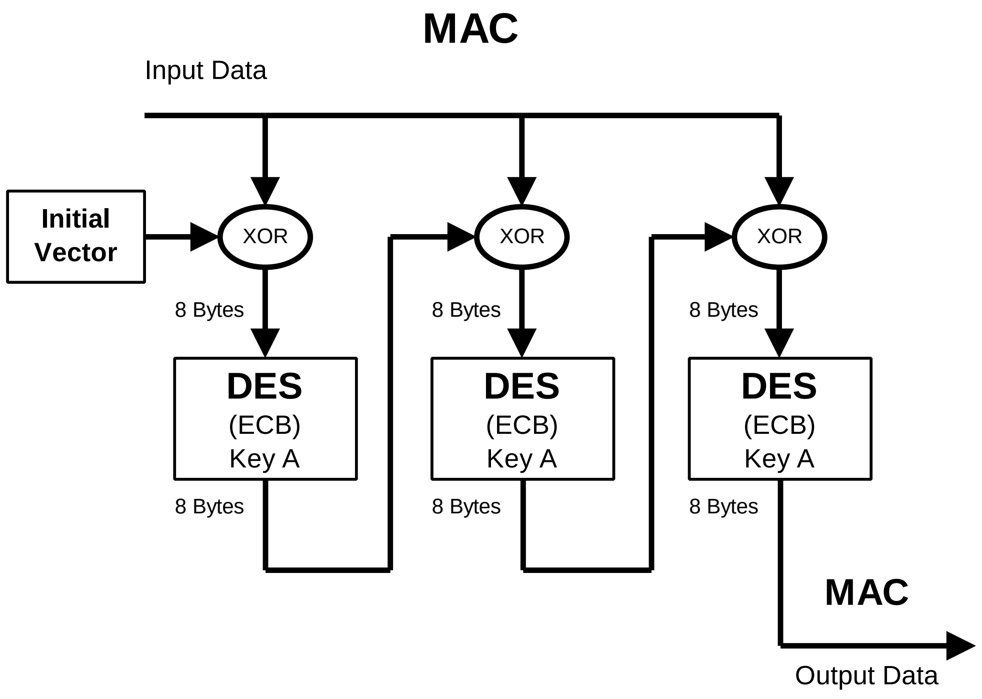


DES3_CBC Decryption



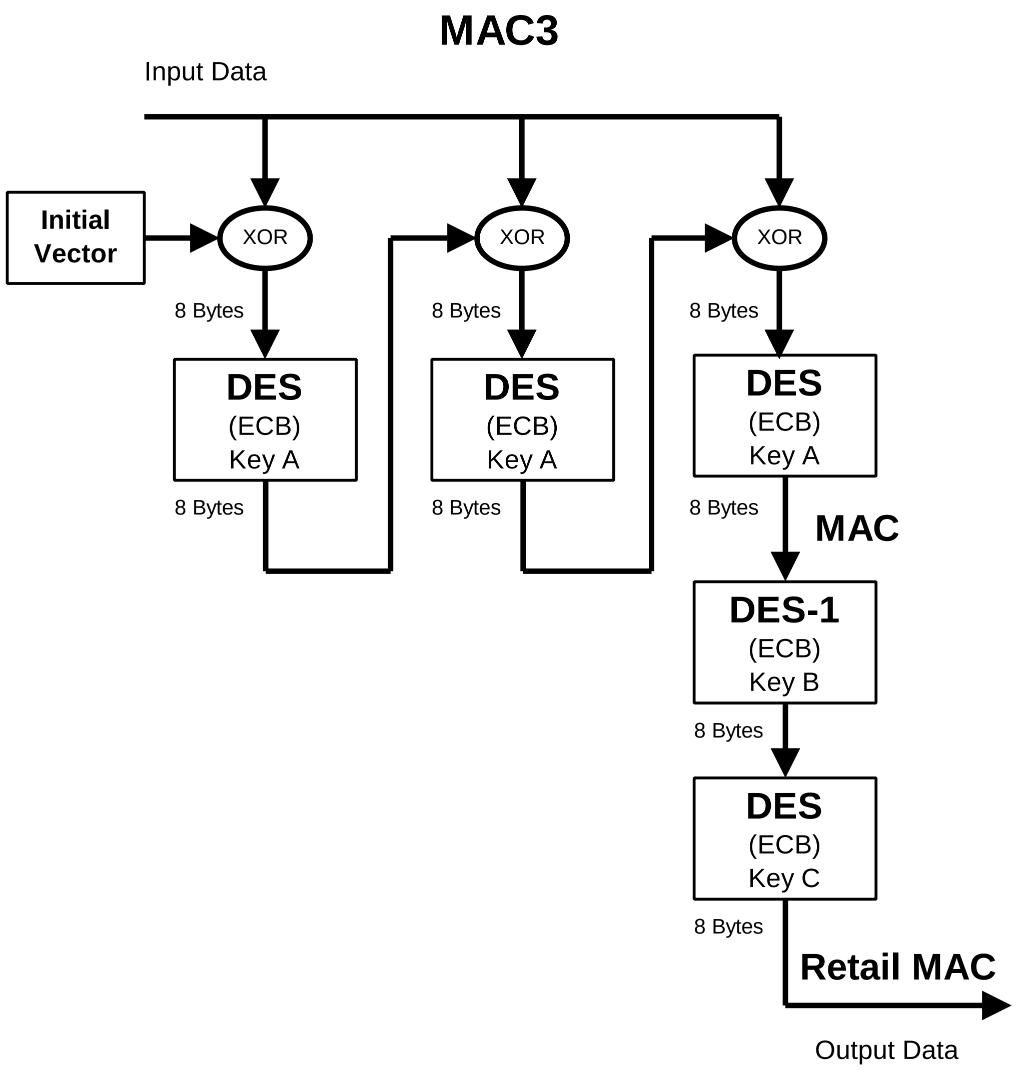
2.4.1.6 MAC and iMAC

Description The MAC and the iMAC algorithms are identical except the padding method. While the MAC algorithm supports ANSI-Padding, the iMAC algorithm supports ISO-Padding.



2.4.1.7 MAC3 and iMAC3 (Retail MAC)

Description The MAC3 and the iMAC3 algorithms are identical except the padding method. While the MAC3 algorithm supports ANSI-Padding, the iMAC3 algorithm supports ISO-Padding.



2.4.2 Asymmetric Algorithms

CardOS/M4 supports different asymmetric algorithms for operation. Table 2.4.2-1 shows the different algorithms, their methods, their usage and - if necessary - the belonging key type and the belonging padding method.

Table 2.4.2-1 Overview of Cryptographic Algorithms of CardOS/M4

Usage	**Algorithm Type**	**Used Key**	**Possible Algorithms (Calculation Method)**	**ALGO Byte** [2]	**Padding Method(s) hashing / non-hashing** [3]
Encryption / Decryption	RSA for / En-/Decryption	Public_Key / Private_Key	RSA	08h	- / PKCS#1-BT2-Padding
			RSA_PURE	0Ch	- / Leading-Zeroes-Padding [4]
Signature **without** hashing	RSA/DSA for Signature **without** Hashing	Public_Key / Private_Key	RSA_SIG	88h	- / PKCS#1-BT1-Padding
			DSA [1]	89h	- / DSA-Padding
			RSA_PURE_SIG	8Ch	- / Leading-Zeroes-Padding [4]
Signature **with** hashing	RSA/DSA for Signature **with** Hashing	Public_Key / Private_Key	RSA_SIG_SHA-1	C8h	SHA-1-Padding / PKCS#1-BT1-Padding
			DSA_SHA-1 [1]	C9h	SHA-1-Padding / DSA-Padding
			RSA_PURE_SIG_SHA-1	CCh	SHA-1-Padding / Leading-Zeroes-Padding [4]
Hashing	Hashing	no Key	SHA-1	CFh	SHA-1-Padding / -

(1): DSA algorithms are available only, if DSA_Package was loaded and activated before.
(2): ALGO Byte in the Object Parameters (see section Objects or section PUT DATA_OCI)
(3): Padding method for the hashing part / non-hashing part of the algorithm
(4): CardOS/M4.00 implements the padding method PKCS#1-BT0-Padding, i.e. PKCS#1 padding with block type 0. If the LeadingZero_Package is loaded, this method is replaced by Leading-Zeroes-Padding.
CardOS/M4.01 implements the padding method Leading-Zeroes-Padding.

All cryptographic algorithms consist of a **fixed calculation method** and a **fixed padding method**. This means that using an ALGO byte for an object fixes the calculation method and the padding to be used.

Except the hashing algorithm SHA-1 all algorithms need a key. Nevertheless for the PSO_H command the HASH component of the CSE must reference a so-called PSO-Hash_Key, which is set using the PUT DATA_OCI command but is no real key. The PSO object representing a PSO-Hash_Key has a dummy byte in the object data.

Depending on the command, which uses the key, (i.e. depending on the algorithm usage) different keys, which may reside in different BS objects, must be used.

2.4.2.1 Asymmetric Key Format (CCMS Format)

Description All keys used by asymmetric algorithms consist of key components, which must be specified in the so-called CCMS format. Key components are set with the PUT DATA_OCI command in the Object Data OCI (TAG=8Fh). The key components represent unsigned integer values.

CCMS Format The CCMS format is described in table 2.4.2.1-1

Table 2.4.2.1-1 CCMS Format for Key Components

Length	Trailing Zeroes	Value of Parameter		
		xxh	...	xxh
		MSB	...	LSB
k=v+1	t=**00h**	unsigned integer		
1 byte	1 byte	v bytes		
(v+2) bytes				

MSB	Most Significant Byte
LSB	Least Significant Byte
t	Must be set to 00h

Byte Format The 8 bits of a byte are interpreted as described in table 2.4.2.1-2.

Table 2.4.2.1-2 Interpretation of the 8 Bits inside a Byte.

Byte							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MSBit	...						LSBit

MSBit	Most Significant Bit
LSBit	Least Significant Bit

2.4.2.2 RSA Algorithms

Description

CardOS/M4 supports the RSA algorithm which is used for the RSA-based algorithms

- **RSA**
- **RSA_PURE**
- **RSA_SIG**
- **RSA_PURE_SIG**
- **RSA_SIG_SHA-1**
- **RSA_PURE_SIG_SHA-1**.

RSA Keys

RSA-based algorithms use RSA_Keys. RSA_Keys consist of RSA key components.

A **Public RSA Key** consists of the key components

- Modulus N and
- Exponent E.

A **Private RSA Key** consists of the key components

- Modulus N and
- Exponent D.

If both keys (public and private) must be stored on the smart card then two key objects must exist, i.e. even the modulus must be stored twice.

RSA Key Components

RSA key components represent unsigned integer values and must have the so-called CCMS format. The following RSA key components are provided (see PUT DATA_OCI command also).

Table 2.4.2.2-1 RSA Key Components

Key Component	Meaning / Description	Value / Limit / Condition	Component Number in Object
Modulus N	This is the product of two primes p and q, which must remain secret. **Maximum Length**	 **1024 bits** (128 bytes; 80h bytes)	000
Exponent E	This is the Public_Exponent.	E is relative prime to $(p\text{-}1) \times (q\text{-}1)$.	001
Exponent D	This is the Private_Exponent governed by the equation:	$D = E^{-1} \pmod{(p\text{-}1) \times (q\text{-}1)}$	001

The Encryption/Decryption operations are built according to the following equations:

- **Encrypting** $C = M^E \pmod N$
- **Decrypting** $M = C^D \pmod N$

C	Cipher Text
M	Plain Text
mod N	Modular operation using modulus N

All RSA-based algorithms of CardOS/M4 are based on the described Encrypting/Decrypting operations.

Padding

For RSA-based algorithms 3 padding methods are implemented in CardOS/M4

- **PKCS#1-BT1-Padding,**
- **PKCS#1-BT2-Padding and**
- **Leading-Zeroes-Padding**

Note that **PKCS#1-BT0-Padding** is only available in CardOS/M4.00 and if the LeadingZero_Package is not activated. Otherwise always **Leading-Zeroes-Padding** is implemented.

PKCS#1-BT1-Padding

This is the PKCS#1 padding with block type 1, which is used during the RSA signing procedure in the algorithms RSA_SIG (ALGO=88h) and RSA_SIG_SHA-1 (ALGO=C8h) (see format in table 2.4.2.2-2).

Table 2.4.2.2-2 Format for PKCS#1-BT1-Padding

Leading Zero Byte	**Block Type Byte**	**Padding String**	**Zero Byte**	**Input_Data**	
LZ_Byte	BT_Byte	Pad_String	Z_Byte	DSI_Object_Data	In_Data
00h	01h	FFh ... FFh	00h	s bytes / empty	d bytes
1 byte	1 byte	p bytes	1 byte	u bytes	
k bytes					
Encryption Block					

s	Length of DSI_Object_Data
d	Length of Data
u	Length of Input_Data
p	Length of Pad_String
k	Length of Encryption Block
m	Length of Modulus N for RSA algorithms
Note	All lengths are denoted in bytes.

If no DSI object is effective with the RSA keys, then s = 0. In_Data are the hash data for RSA_SIG and the data to be hashed for RSA_SIG_SHA-1.

CardOS/M4 checks the following:

- $k = m$ (1)
- $u \leq m - (s+3)$ (2)

According to the PKCS#1 standard the following is recommended additionally:

- $p \geq 8$ (3)

PKCS#1-BT2-Padding

This is the PKCS#1 padding with block type 2, which is used during the RSA encryption/decryption procedures with the algorithm RSA (ALGO=08h) (see format in table 2.4.2.2-3).

Table 2.4.2.2-3 Format for PKCS#1-BT2-Padding

Leading Zero Byte	**Block Type Byte**	**Padding String**	**Zero Byte**	**Input Data**
LZ_Byte	BT_Byte	Pad_String	Z_Byte	In_Data
00h	02h	Non-Zero_Random_ Number (*)	00h	Data
1 byte	1 byte	p bytes	1 byte	u bytes
k bytes				
Encryption Block				

(*)	The Non-Zero_Random_Number is a random number (i.e. random string) not containing any 00h byte.
u	Length of In_Data
p	Length of Pad_String
k	Length of Encryption Block
m	Length of Modulus N for RSA algorithms
Note	All lengths are denoted in bytes.

In_Data are the data to be encrypted/decrypted for RSA.

CardOS/M4 checks the following:

- $k = m$ (4)
- $u \leq m - 3$ (5)

According to the PKCS#1 standard the following is recommended additionally:

- $p \geq 8$ (6)

Leading-Zeroes-Padding

This is a proprietary padding method using the Input Data as unsigned integer with modulus length during the RSA-based algorithms RSA_PURE (ALGO=0Ch), RSA_PURE_SIG (ALGO=8Ch) and RSA_PURE_SIG_SHA-1 (ALGO=CCh) (see format in table 2.4.2.2-4).

Table 2.4.2.2-4 Format for Leading-Zeroes-Padding

Leading-Zeroes-Padding String	**Input_Data**	
Pad_String	DSI_Object_Data	In_Data
00h ... 00h	s bytes / empty	d bytes
p bytes	u bytes	
MSB	...	LSB
k bytes		
Encryption Block		

MSB	Most Significant Byte for unsigned integer
LSB	Least Significant Byte for unsigned integer
s	Length of DSI_Object_Data
d	Length of In_Data
u	Length of Input_Data
p	Length of Pad_String
k	Length of Encryption Block
m	Length of Modulus N for RSA algorithms
Note	All lengths are denoted in bytes.

If no DSI object is effective with the RSA keys, then s = 0. In_Data are the hash data for RSA_PURE_SIG, the data to be hashed for RSA_PURE_SIG_SHA-1 and the data to be encrypted/decrypted for RSA_PURE.

CardOS/M4 checks the following:

- $k = m$ (7)
- $u \leq m$ (8)

Condition (8) is checked by CardOS/M4.01 always. For CardOS/M4.00 condition (8) is checked only, if the so-called LeadingZeroes_Package is loaded and activated in the smart card. Otherwise condition (8a) is checked by CardOS/M4.00:

- $u \leq m - 3$ (8a)

As the Pad_String is not checked by CardOS/M4 the Pad_String may be empty (p=0) or the Pad_String may be used for other padding methods as well.

Note CardOS/M4.00 implements the padding method **PKCS#1-BT0-Padding**, i.e. PKCS#1 padding with block type 0. If the LeadingZeroes_Package is loaded, this method is replaced by **Leading-Zeroes-Padding**.
CardOS/M4.01 implements the padding method **Leading-Zeroes-Padding**.

2.4.2.3 DSA Algorithms

Description If the DSA package is loaded and activated, CardOS/M4 supports the DSA algorithm, which is used for the DSA-based algorithms

- **DSA**
- **DSA_SHA-1**

DSA Keys DSA-based algorithms use DSA_Keys. DSA_Keys consist of DSA key components.

A **Public DSA Key** consists of the key components

- Prime P
- Prime Q
- Base G
- Public Factor Y.

A **Private DSA Key** consists of the key components

- Prime P
- Prime Q
- Base G
- Private Factor X.

DSA Key Components DSA key components represent unsigned integer values and must be specified in the so-called CCMS format. The following DSA key components are provided (see PUT DATA_OCI command also).

Table 2.4.2.3-1 DSA Key Components

Key Component	Meaning / Description	Value / Limit / Condition	Component Number in Object
Prime P	Prime with specific length.		000
	Minimum Length	512 bits	
	Maximum Length	1024 bits (128 bytes; 80h bytes)	
	Length	Multiple of 64 bits	
Prime Q	Prime factor of (P-1) with specific length.		001
	Length	160 bits (20 bytes) exactly	
Base G	Maximum Length	Length of P	010
	Value	$G = h^{(P-1)/Q}$; $G \neq 0$	
		$h < (P-1)$	
		$h^{(P-1)/Q} \bmod Q > 1$	
Private Factor X	Maximum Length	Length of Q	011
	Value	$X < Q$; $X \neq 0$	
Public Factor Y	Maximum Length	Length of Q	011
	Value	$Y= G^X \pmod P$; $Y \neq 0$	

DSA-Padding

This is a proprietary padding method for the DSA algorithm (ALGO=89h; i.e. DSA without hashing). The Format is described in table 2.4.2.3-2.

Table 2.4.2.3-2 Format for DSA-Padding

Leading-Zeroes-Padding String	**Input Data**
00h ... 00h	Data
p bytes	u bytes
k=20 bytes	
Encryption Block	

u	Length of Input Data and	
	$u \leq 20$	(9)
k	Length of Encryption Block	
p	Length of Padding String, which is governed by the following conditions:	
	$k \equiv 20$	(10)
	$p = k-u$	(11)
Note	All lengths are denoted in bytes.	

2.4.2.4 SHA-1 Algorithm for Hashing

Description The SHA-1 algorithm is provided as standard hashing algorithm for CardOS/M4.

SHA-1-Padding This is the padding method for SHA-1 (see table 2.4.2.4-1).

Table 2.4.2.4-1 General Format for SHA-1-Padding

Input Data	**Padding**	
	Variable Padding String	**Input Data Length**
Data	80h 00h ... 00h	Length r
u bytes	v = 1 ... 56 bytes	8 bytes
	p = (v+8) bytes = 9 ... 64 bytes	
(u+p) bytes ≡ (n*64) bytes		

u	Length of Input Data in Bytes
r	Length of Input Data in Bits, i.e. r = (8*u)
n	Integer, n > 0
p	Length of Padding String, which is governed by the following conditions:

$v > 0$	(12)	$v \leq 64$	(13)
$p = v+8$	(14)	$(u+p) = (n*64)$	(15)

Note All lengths are denoted in bytes, if not denoted differently.

The SHA-1 is working on a Encryption Block length of exactly 64 (i.e. 40h) bytes. SHA-1 can be used in command chaining mode and therefore the input data for SHA-1 may exceed 64. As a result of this different cases for the **last two** Encryption Blocks must be distinguished (see table 2.4.2.4-2).

Table2.4.2.4-2 Different cases for the last two Encryption Blocks for SHA-1-Padding

Case	**Last but one Encryption Block (64 bytes)**			**Last Encryption Block (64 bytes)**			
1	**Last but one part of Input Data**			**Last part of Input Data**	**Padding**		
					80h	length r	
					1 byte	8 bytes	
	64 bytes			55 bytes	p = 9 bytes		
2	**Last but one part of Input Data**			**Last part of Input Data**	**Padding**		
					80h	00h...00h	length r
					1 byte	54 bytes	8 byte
	64 bytes			1 byte	63 bytes		
3	**Last part of Input Data**			**Padding**			
				80h	00h ... 00h		length r
				1 byte	55 bytes		8 bytes
	64 bytes			p = 64 bytes			
4	**Last part of Input Data**			**Padding**			
			80h	00h ... 00h			length r
			1 byte	56 bytes			8 bytes
	63 bytes			p = 65 bytes			
5	**Last part of Input Data**	**Padding**					
		80h	00h..00h	00h ... 00h			length r
		1 byte	7 bytes	56 bytes			8 bytes
	56 bytes	p = 72 bytes					

Hash_Key A Hash_Key shall be a BS object with a hash algorithm and any data in the object data.

2.5 SM (Secure Messaging)

Introduction

Secure messaging (SM) is used to protect communication data transmitted between smart card and terminal/host and vice versa. It is distinguished between

- **command-SM** (i.e. SM used for commands from the terminal/host to the smart card) and
- **response-SM** (i.e. SM used for responses from the smart card to the terminal/host).

For command-SM and response-SM

- **SM_ENC** (i.e. encryption of communication data),
- **SM_SIG** (i.e. signing of communication data; for CardOS/M4 this means only MACing)

or a combination of both can be provided.

Usage

For each command (i.e. APDU) the communication mode must be coded in the CLA byte. According to ISO 7816-4 bits 2^3 and 2^2 of the CLA byte determine the SM format. The bits 2^1 and 2^0 of the CLA byte define the logical channel. For easier notation these bits are not considered in specified SM modes (i.e. section 2.5) nor in command specifications (i.e. section 3). So x4h means ‘xxxx 01xx’ in binary notation for instance.

Communication Modes

For communication CardOS/M4 supports

- one regular (i.e. unprotected mode) using CLA=x0h and
- two SM modes (i.e. protected modes) using CLA=x4h or CLA=xCh and using **symmetric cryptography**.

Table 2.5-1 explains both SM modes.

Table 2.5-1 SM modes

SM Mode (using CLA byte)	**Comment**
x4h	This is a **proprietary** and unTAGged (i.e. does not use TAGs) SM mode for special purposes. **Usage:** This mode can only be used during smart card personalization (i.e. until the life cycle phase *ADMINISTRATION* is reached and during life cycle phase *ADMINISTRATION* for certain commands only) and cannot be used for other purposes. SM mode x4h is usable in life cycle phases, in which no CardOS/M4 filesystem and no objects exist. Consequently neither SM definitions nor SM objects can be used. Therefore SM mode x4h is only usable for **certain** commands using a **fixed** SM_SIG and a **fixed** SM_ENC method with **fixed** keys (see section 2.5.1).
xCh	The SM mode xCh is a TAGged (i.e. this mode uses TAGs) SM. **Usage:** This mode can only be used in the life cycle phases *ADMINISTRATION* and *OPERATIONAL*. SM mode xCh is provided for standard usage of SM. This means that SM mode xCh is fully scalable (see section 2.5.2).

2.5.1 SM Mode x4h

Description

SM mode x4h is used for initialization / personalization purposes. This means, that SM mode x4h can only be used for certain commands in the life cycle phases

- *MANUFACTURING*,
- *INITIALIZATION*,
- *PERSONALIZATION* and
- *ADMINISTRATION.*

As SM mode x4h is usable in life cycle phases, in which no filesystem and no objects exist, SM mode x4h uses

- a **fixed SM_SIG** using the algorithm MAC3 (i.e. ALGO=82h),
- a **fixed SM_ENC** using algorithm DES3_CBC (i.e. ALGO=03h) and
- **fixed system keys**.

System Keys

System keys are 16 bytes in length. According to the fixed algorithms mentioned above the DES3_CBC and the MAC3 algorithm are used using a A-B-A key structure.

System keys are stored in EEPROM locations outside the filesystem area of the smart card. For SM mode x4h CardOS/M4 uses the following system keys:

- **StartKey**
- **PackageLoadKey**
- **PersonalizationKey(s)**

Use of System Keys with SM mode x4h

For one command the same system key is used for SM_SIG and SM_ENC. The possible combinations for commands using SM mode x4h and the system keys is given in table 2.5.1-1.

Table 2.5.1-1 Combinations for commands using SM mode x4h and system keys

Command	Usable System Key(s)
CHANGE KEY	StartKey, PackageLoadKey, PersonalizationKey(s)
ERASE FILES	StartKey
FORMAT	StartKey
INITIALIZE EEPROM	StartKey
INITIALIZE END	StartKey
PERSONALIZE	PersonalizationKey(s)
LOAD EXECUTABLE	PackageLoadKey (for CardOS/M4.01 only)

Specials

All commands, which use SM mode x4h, send communication data from the host/terminal to the smart card only. Therefore **only command-SM is supported for SM mode x4h**.

SM Format

Generally for SM mode x4h CardOS/M4 **always** executes the following calculation steps in the given order:
1st Calculation of the SM_SIG using the MAC3 algorithm
2nd Calculation of the SM_ENC using the DES3_CBC algorithm.

For all commands mentioned above except the INITIALIZE EEPROM command **both calculation steps must be performed**. For the INITIALIZE EEPROM command SM mode x4h can be used

- calculating **SM_SIG only** or
- calculating **SM_ENC only** or
- performing **both calculation steps** as described above.

SM Protection Methods

As a result of this potentially the following **SM Protection Methods** exist for SM mode x4h:

- **Command-SM_SIG**,
- **Command-SM_ENC** and
- **Command-SM_SIG_ENC**
  (as a combination of command-SM_SIG and a following command-SM_ENC).

Taking this into account SM mode x4h builds the possible SM formats as described in the sections 2.5.1.1 to 2.5.1.3. For all SM format descriptions it is assumed, that the calculation is starting from a command without SM (e.g. the INITIALIZE EEPROM command without SM).

Scalability

Since the algorithms and the usable system keys are fixed for SM mode x4h, the only possibility to vary SM mode x4h is to change the used system keys using the CHANGE KEY command.

2.5.1.1 Command-SM_SIG (SM Mode x4h)

Description This section describes how to build command-SM only using SM_SIG for SM mode x4h.

Command to be used for SM_SIG (i.e command without SM)

CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
	Used for next action				Used for next action	

MAC calculation for SM_SIG

INS	P1	P2	LC	Command Data Field	ANSI-Padding
			LC=u+8	**Data**	00..00h
1 byte	1 byte	1 byte	1 byte	u bytes	0..7 bytes
Relevant for **MAC** calculation					

Including the MAC into the command to be sent

CLA	INS	P1	P2	LC	Command Data Field		LE
x4h				LC=w	Data	**MAC**	LE
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	8 bytes	1 byte
					w=(u+8) bytes		
Command-SM_SIG Format in SM Mode x4h							

2.5.1.2 Command-SM_ENC (SM Mode x4h)

Description This section describes how to build command-SM only using SM_ENC for SM mode x4h.

Command to be used for SM_ENC (i.e command without SM)

CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
					Used for next Step	

ENC calculation for SM_ENC

Command Data Field	ISO-Padding
Data	80..00h
u bytes	p = 1..8 bytes
(u+p) bytes	
Relevant for **ENC** calculation	

Including ENC into the command to be sent

CLA	INS	P1	P2	LC	Command Data Field	LE
x4h				LC=w	**ENC (encrypted Data and ISO-Padding)**	LE
1 byte	1 byte	1 byte	1 byte	1 byte	w=(u+p) bytes	1 byte
Command-SM_ENC Format in SM Mode x4h						

2.5.1.3 Command-SM_SIG_ENC (SM Mode x4h)

Description This section describes how to build command-SM using SM_SIG first and then SM_ENC for SM mode x4h.

Command to be used for SM_SIG (i.e command without SM)

CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
	Used for next action				Used for next action	

MAC calculation for SM_SIG

INS	P1	P2	LC	Command Data Field	ANSI-Padding
			LC=u+8	**Data**	00..00h
1 byte	1 byte	1 byte	1 byte	u bytes	0..7 bytes
Relevant for **MAC** calculation					

Including the MAC into the Command Data Field

Command Data Field	
Data	**MAC**
u bytes	8 bytes
(u+8) bytes	
Used for next action	

ENC calculation for Command Data Field and Padding

Command Data Field		**ISO-Padding**
Data	**MAC**	80..00h
u bytes	8 bytes	p = 1..8 bytes
(u+p+8) bytes		
Relevant for **ENC** calculation		

Including ENC into the command to be sent

CLA	INS	P1	P2	LC	Command Data Field	LE
x4h				LC=w	**ENC (encrypted Data, MAC and ISO-Padding)**	LE
1 byte	1 byte	1 byte	1 byte	1 byte	w=(u+p+8) bytes	1 byte
Command-SM_SIG_ENC Format in SM Mode x4h						

2.5.2 SM Mode xCh

Description

SM mode xCh is provided for standard usage of SM (i.e. for usage of the most commands in the life cycle phases *ADMINISTRATION* and *OPERATIONAL*).

Additionally to the CLA byte coding for SM (i.e. CLA=xCh) so-called **SM definitions** and **SM objects** are used in SM mode xCh.

SM Definitions

For all data transmission actions on files or objects (i.e. actions on files/objects, where data are received or transmitted by the smart card) SM definitions define, whether and how SM shall be used. This means that each SM definition is command- or response-specific (i.e. influences either command-SM or response-SM).

Each SM definition either references a SM object via the SM object ID (i.e. SM definition byte = **00h .. FEh**) or disables SM (i.e. SM definition byte = **FFh**).

SM definitions are **installed**

- during file creation
  (see FORMAT and CREATE FILE command) respectively
- during object creation
  (see PUT DATA_OCI command).

SM definitions can be **administrated**

- using the PUT DATA_FCI command for files.

SM definitions can be **overwritten** together with all other settings

- using the PUT DATA_OCI command for objects.

SM definitions for files and objects are described in the sections 2.5.2.1 and 2.5.2.2 or in the description of the commands FORMAT, CREATE FILE, PUT DATA FCI and PUT DATA_OCI.

For each action on files/objects

- one SM definition for **SM_ENC** and
- one SM definition for **SM_SIG**

are provided.

Additionally there may arise the situation, that SM is switched on for an action on a file/object (i.e. via CLA=xCh) and one communication direction (of the possible two directions) shall be **suppressed**. This means for instance, that only command-SM or only response-SM shall be used for a command needing both communication directions (e.g. the INCREASE command).
Such a situation is denoted as

- **Command-SM_suppressed** respectively
- **Response-SM_suppressed**.

A communication direction can be suppressed, if the corresponding SM definition disables SM (see above).

SM Protection Methods

As a result of the said potentially the following **SM Protection Methods** exist for SM mode xCh:

- **Command-SM_ENC**,
- **Command-SM_SIG**,
- **Command-SM_ENC_SIG**
  (as a combination of command-SM_ENC and a following command-SM_SIG),
- **Command-SM_suppressed,**
- **Response-SM_ENC,**
- **Response-SM_SIG** or
- **Response-SM_ENC_SIG**
  (as a combination of response-SM_ENC and a following response-SM_SIG),
- **Response-SM_suppressed.**

Taking this into account SM mode xCh builds the possible SM formats as described in the following sections. For all SM format descriptions it is assumed, that the calculation is starting from a command without SM and using command-SM as well as response-SM (e.g. the INCREASE command without SM)

SM Objects

A SM object contains all information necessary for using SM. SM objects, **do not** affect the *security status* of any DF. As a SM objects is a special kind of BS object, SM objects are described in section 2.3.

SM_Data_Objects

SM mode xCh uses **SM_Data_Objects** (see section 2.5.2.3) in the command data field or in the response data field.

2.5.2.1 SM Definitions for Files

Description The following tables 2.5.2.1-1 to 2.5.2.1-3 show for a MF, a DF and an EF, which command(s) (i.e. access action(s)) on files use which SM definition(s).

Note In CardOS/M4.01 SM definitions for MF and DFs have been changed, because it is not possible to distinguish wether UPDATE or APPEND occurs before decryption of the determining file-id. So the definitions ENC UPDATE and ENC APPEND have been combined to ENC UPDATE/APPEND and further the definitions SIG UPDATE and SIG APPEND to SIG UPDATE/APPEND.

Table 2.5.2.1-1 SM Definitions for the MF

Byte No.	SM definitions (possible values)	Meaning for MF	Commands using the SM definition
1	00h ... FFh	rfu	-
2	00h ... FFh	rfu	-
3	00h ... FFh	ENC UPDATE (Objects) (0) / ENC UPDATE/APPEND (Objects) (1)	PUT DATA_ATR, PUT DATA_DSI, PUT DATA_OCI, PUT DATA_SECI
4	00h ... FFh	SIG UPDATE (Objects) (0) / SIG UPDATE/APPEND (Objects) (1)	PUT DATA_ATR, PUT DATA_DSI, PUT DATA_OCI, PUT DATA_SECI
5	00h ... FFh	ENC APPEND (Objects) (0) / rfu (1)	PUT DATA_ATR, PUT DATA_DSI, PUT DATA_OCI, PUT DATA_SECI
6	00h ... FFh	SIG APPEND (Objects) (0) / rfu (1)	PUT DATA_ATR, PUT DATA_DSI, PUT DATA_OCI, PUT DATA_SECI
7	00h ... FFh	rfu	-
8	00h ... FFh	rfu	-
9	00h ... FFh	rfu	-
10	00h ... FFh	rfu	-
11	00h ... FFh	rfu	-
12	00h ... FFh	rfu	-
13	00h ... FFh	ENC ADMIN (MF)	PUT DATA_FCI
14	00h ... FFh	SIG ADMIN (MF)	PUT DATA_FCI
15	00h ... FFh	ENC CREATE (Files)	CREATE FILE
16	00h ... FFh	SIG CREATE (Files)	CREATE FILE
17	00h ... FFh	rfu	-
18	00h ... FFh	rfu	-
19	00h ... FFh	ENC GENKEY	GENERATE KEY PAIR
20	00h ... FFh	SIG GENKEY	GENERATE KEY PAIR
21	00h ... FFh	rfu	-
22	00h ... FFh	rfu	-
23	00h ... FFh	rfu	-
24	00h ... FFh	rfu	-

Meaning of possible values:

00h ... FEh	ID of SM-key to be used for referenced access action, ENC/SIG and communication direction.
FFh:	**No** SM.
(0):	CardOS/M4.00
(1):	CardOS/M4.01

Table2.5.2.1-2 SM Definitions for a DF

Byte No.	SM definitions (possible values)	Meaning for DF	Commands using the SM definition
1	00h ... FFh	rfu	-
2	00h ... FFh	rfu	-
3	00h ... FFh	ENC UPDATE (Objects) (0) / ENC UPDATE/APPEND (Objects) (1)	PUT DATA_ATR, PUT DATA_DSI, PUT DATA_OCI, PUT DATA_SECI
4	00h ... FFh	SIG UPDATE (Objects) (0) / SIG UPDATE/APPEND (Objects) (1)	PUT DATA_ATR, PUT DATA_DSI, PUT DATA_OCI, PUT DATA_SECI
5	00h ... FFh	ENC APPEND (Objects) (0) / rfu (1)	PUT DATA_ATR, PUT DATA_DSI, PUT DATA_OCI, PUT DATA_SECI
6	00h ... FFh	SIG APPEND (Objects) (0) / rfu (1)	PUT DATA_ATR, PUT DATA_DSI, PUT DATA_OCI, PUT DATA_SECI
3	00h ... FFh	ENC UPDATE (Objects)	PUT DATA_ATR, PUT DATA_DSI, PUT DATA_OCI, PUT DATA_SECI
4	00h ... FFh	SIG UPDATE (Objects)	PUT DATA_ATR, PUT DATA_DSI, PUT DATA_OCI, PUT DATA_SECI
5	00h ... FFh	ENC APPEND (Objects)	PUT DATA_ATR, PUT DATA_DSI, PUT DATA_OCI, PUT DATA_SECI
6	00h ... FFh	SIG APPEND (Objects)	PUT DATA_ATR, PUT DATA_DSI, PUT DATA_OCI, PUT DATA_SECI
7	00h ... FFh	rfu	-
8	00h ... FFh	rfu	-
9	00h ... FFh	rfu	-
10	00h ... FFh	rfu	-
11	00h ... FFh	ENC DELETE (DF)	DELETE FILE
12	00h ... FFh	SIG DELETE (DF)	DELETE FILE
13	00h ... FFh	ENC ADMIN (DF)	PUT DATA_FCI
14	00h ... FFh	SIG ADMIN (DF)	PUT DATA_FCI
15	00h ... FFh	ENC CREATE (files)	CREATE FILE
16	00h ... FFh	SIG CREATE (files)	CREATE FILE
17	00h ... FFh	rfu	-
18	00h ... FFh	rfu	-
19	00h ... FFh	ENC GENKEY	GENERATE KEY PAIR
20	00h ... FFh	SIG GENKEY	GENERATE KEY PAIR
21	00h ... FFh	rfu	-
22	00h ... FFh	rfu	-
23	00h ... FFh	rfu	-
24	00h ... FFh	rfu	-

Meaning of possible values:

00h ... FEh	ID of SM-key to be used for referenced access action, ENC/SIG and communication direction.
FFh:	**No** SM.
(0):	CardOS/M4.00
(1):	CardOS/M4.01

Table2.5.2.1-3 SM Definitions for a EF

Byte No.	SM definitions (possible values)	Meaning for EF	Commands using the SM definition
1	00h ... FFh	ENC READ-OUT (Data)	READ BINARY, READ RECORD
2	00h ... FFh	SIG READ-OUT (Data)	READ BINARY, READ RECORD
3	00h ... FFh	ENC UPDATE (Data)	UPDATE BINARY, UPDATE RECORD
4	00h ... FFh	SIG UPDATE (Data)	UPDATE BINARY, UPDATE RECORD
5	00h ... FFh	ENC APPEND (Record) / rfu	APPEND RECORD
6	00h ... FFh	SIG APPEND (Record) / rfu	APPEND RECORD
7	00h ... FFh	rfu	-
8	00h ... FFh	rfu	-
9	00h ... FFh	rfu	-
10	00h ... FFh	rfu	-
11	00h ... FFh	rfu	-
12	00h ... FFh	rfu	-
13	00h ... FFh	ENC ADMIN (EF)	PUT DATA_FCI
14	00h ... FFh	SIG ADMIN (EF)	PUT DATA_FCI
15	00h ... FFh	ENC INCREASE-IN (Data) / rfu	INCREASE
16	00h ... FFh	SIG INCREASE-IN (Data) / rfu	INCREASE
17	00h ... FFh	ENC DECREASE-IN (Data) / rfu	DECREASE
18	00h ... FFh	SIG DECREASE-IN (Data) / rfu	DECREASE
19	00h ... FFh	ENC INCREASE-OUT (Data) / rfu	INCREASE
20	00h ... FFh	SIG INCREASE-OUT (Data) / rfu	INCREASE
21	00h ... FFh	ENC DECREASE-OUT (Data) / rfu	DECREASE
22	00h ... FFh	SIG DECREASE-OUT (Data) / rfu	DECREASE
23	00h ... FFh	ENC READ-IN (DATA) / rfu	READ BINARY, READ RECORD
24	00h ... FFh	SIG READ-IN (Data) / rfu	READ BINARY, READ RECORD

Meaning of possible values:

00h ... FEh	ID of SM-key to be used for referenced access action, ENC/SIG and communication direction.
FFh:	**No** SM.

2.5.2.2 SM Definitions for BS Objects

Description Table 2.5.2.2-1 shows for BS objects, which command(s) (i.e. access action(s)) use which SM definition(s).

Table 2.5.2.2-1 SM Definitions for BS Objects

Byte No.	SM definitions (possible values)	Meaning	Commands using the SM definition
1	00h ... FFh	ENC USE-IN	EXTERNAL AUTHENTICATE, INTERNAL AUTHENTICATE, PSO_CCC, PSO_CDS, PSO_DEC, PSO_ENC, PSO_H, PSO_VCC, PSO_VDS, VERIFY
2	00h ... FFh	SIG USE-IN	EXTERNAL AUTHENTICATE, INTERNAL AUTHENTICATE, PSO_CCC, PSO_CDS, PSO_DEC, PSO_ENC, PSO_H, PSO_VCC, PSO_VDS, VERIFY
3	00h ... FFh	ENC CHANGE (Object Data) / rfu (SE Objects)	CHANGE REFERENCE DATA
4	00h ... FFh	SIG CHANGE (Object Data) / rfu (SE Objects)	CHANGE REFERENCE DATA
5	00h ... FFh	ENC UNBLOCK (BS Objects) / rfu (SE Objects)	RESET RETRY COUNTER
6	00h ... FFh	SIG UNBLOCK (BS Objects) / rfu (SE Objects)	RESET RETRY COUNTER
7	00h ... FFh	ENC rfu	-
8	00h ... FFh	SIG rfu	-
9	00h ... FFh	ENC rfu	-
10	00h ... FFh	SIG rfu	-
11	00h ... FFh	ENC rfu	-
12	00h ... FFh	SIG rfu	-
13	00h ... FFh	ENC rfu	-
14	00h ... FFh	SIG rfu	-
15	00h ... FFh	ENC USE-OUT	INTERNAL AUTHENTICATE, PSO_CCC, PSO_CDS, PSO_DEC, PSO_ENC, PSO_H
16	00h ... FFh	SIG USE-OUT	INTERNAL AUTHENTICATE, PSO_CCC, PSO_CDS, PSO_DEC, PSO_ENC, PSO_H

Meaning of possible values:

00h ... FEh	ID of SM-key to be used for referenced access action, ENC/SIG and communication direction.
FFh:	**No** SM.

2.5.2.3 SM_Data_Objects using SM TAGs

Description

This section describes the SM_Data_Objects used by SM mode xCh. (Note that SM mode x4h does not use SM_Data_Objects).

Each SM_Data_Object is **non-BER-TLV coded** and has a context-specific TAG (i.e. SM TAG) specified by ISO 7816-4. Table 2.5.2.3-1 shows all SM_Data_Objects supported by CardOS/M4.

Table 2.5.2.3-1 SM_Data_Objects with TAGs used by SM mode xCh

SM_Data_Object	Context-specific TAG	Value / Description
Plain_Text_Object	80h	Plain text data: Plain_Text_Object **does not become part of a MAC calculation**. Any other TAGged data for a command or response are **inside** the Plain_Text_Object.
Plain_Text_MAC_Object	81h	Plain text data: Plain_Text_Object becomes part of a MAC calculation. Any other TAGged data for a command or response are **inside** the Plain_Text_Object.
Cipher_Text_Object	86h	Cipher text data: Chipher_Text_Object **does not become part of a MAC calculation**. Any other TAGged data for a command or response are **inside** the Cipher_Text_Object.
Cipher_Text_MAC_Object	87h	Cipher text data: Chipher_Text_Object becomes part of a MAC calculation. Any other TAGged data for a command or response are **inside** the Cipher_Text_Object.
MAC_Object	8Eh	The MAC itself
Net_LE_Object	96h	Net_LE Byte: Net_LE_Object **does not become part of a MAC calculation.**
Net_LE_MAC_Object	97h	Net_LE Byte: Net_LE_MAC_Object becomes part of a MAC calculation.

According to ISO 7816-4 SM_Data_Objects using **odd** TAGs are **included** in the MAC calculation and those using **even** TAGs are **not included**.

SM mode xCh accepts only the context-specific TAGs given in table 2.5.2.3-1. If any other context-specific TAGs are inside a command, then either CardOS/M4 throws them away (except when they occur inside the context-specific TAGs of table 2.5.2.3-1) or an error occurs.

SM mode xCh expects only **one** of the TAGs 81h or 87h. The command data or the response data must therefore be **encapsulated** in a 81h or 87h TAG to let SM pass them through.

The net LE information can only occur **once**, either with TAG 96h (i.e. as Net_LE_Object) or with TAG 97h (i.e. as Net_LE_MAC_Object). If both SM_Data_Objects are included in a command data field, then this leads to unpredictable results, i.e. it is not predictable, if the MAC verification succeeds or not.

Net LE

The net LE is necessary for commands using response-SM. The reason is, that the net data **length** of the **response data** requested by a command (i.e. the **net LE**) may differ from the **length** of the **response data field** (i.e. **LE**) for example because of padding bytes. For such cases the net LE **and** the LE must be specified by the command. Additionally it must be taken into account, whether the Net_LE_Object or Net_LE_MAC_Object is present completely or the value (i.e. the net LE) is missing.

Table 2.5.2.3-2 gives an overview of the possible values for LE, Net_LE_Object/Net_LE_MAC_Object and net LE and their meaning.

Table 2.5.2.3-2 Meaning for different values of LE, Net_LE_Object/Net_LE_MAC_Object and net LE

LE	**Net_LE_Object /. Net_LE_MAC_Object**			**Meaning / Result**
	T	**L**	**V = Net LE**	
empty	empty			no data returned
	96h / 97h	00h	empty	Error
	96h / 97h	01h	00	Error
	96h / 97h	01h	xxh	Error
00h	empty			maximum number of bytes returned
	96h / 97h	00h	empty	maximum number of bytes returned
	96h / 97h	01h	00	Error
	96h / 97h	01h	xxh	Error
xxh	empty			xxh bytes returned
	96h / 97h	00h	empty	Error
	96h / 97h	01h	00h	Error
	96h / 97h	01h	xxh	xxh bytes returned
	96h / 97h	01h	yyh	xxh < yyh: Error
	96h / 97h	01h	zzh	xxh > zzh: zzh bytes returned

2.5.2.4 Command-SM_ENC (SM Mode xCh)

Description This section describes how to build command-SM only using SM_ENC for SM mode xCh.

1st Step **Building the Cipher_Text_Object:**

Command to be used for SM_ENC (i.e command without SM)

CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
					Used for next action	

ENC calculation for SM_ENC

Command Data Field	ISO-Padding (for all ENC algoritms)
Data	80..00h
u bytes	p = 1..8 bytes
(u+p) bytes	
Relevant for **ENC** calculation	

Including ENC into the Cipher_Text_Object

T	L	V	
		Padding Indicator	Enciphered Data
86h / 87h	u+p+1	01h	**ENC**
1 byte	1 byte	1 byte	(u + p) bytes
(u + p + 3) bytes			
Cipher_Text_Object to be used for one of the following steps			

Cipher_Text_Object	The enciphered command data ENC must be sent inside a Cipher_Text_Object (i.e. TAG=86h or TAG=87h). Inside the value field of the Cipher_Text_Object the fixed padding indicator 01h must be added.
r	r specifies the Net_LE

2nd Step Building the Net_LE_Object (if necessary):

Command to be used for SM_ENC (i.e command without SM)

CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
						Used for next action

Building the Net_LE_Object

T	L	V
96h	01h	**r = Net_LE**
1 byte	1 byte	1 byte
3 bytes		
Net_LE_Object to be used for one of the following steps		

r	r specifies the Net_LE

3rd Step Building the Command:

1st Step **2nd Step**

Including Cipher_Text_Object and Net_LE_Object into command

CLA	INS	P1	P2	LC	Command Data Field		LE
xCh				LC=w	**Cipher_Text_Object**	**Net_LE_Object** (*) / **empty** (**)	LE=t
1 byte	1 byte	1 byte	1 byte	1 byte	(u+p+3) bytes	m = 3 bytes (*) / m = 0 bytes (**)	1 byte
					w = (u+p+6) bytes (*) / w = (u+p+3) bytes (**)		
Command_SM_ENC Format in SM Mode xCh							

CLA:	The CLA byte must be set to xCh
(*), (**)	In the command data field the Net_LE_Object (TAG=96h) can (i.e. (*)) or cannot (i.e. (**)) be specified. Therefore two different possible values w exist for the length of the command data field.
t	This is the length LE of the requested response data field taking into account all auxiliary bytes necessary for response-SM

2.5.2.5 Command-SM_SIG (SM Mode xCh)

Description This section describes how to build command-SM only using SM_SIG for SM mode xCh.

1st Step **Building the Header_Block:**

Command to be used for SM_SIG (i.e command without SM)

CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
	Used for next action					

Internal Random Number	CLA	INS	P1	P2	Padding (according to SIG algorithm)
I_RAND	xCh				Padding Bytes
n * 8 bytes / empty	1 byte	1 byte	1 byte	1 byte	4 bytes
(n+1) * 8 bytes					
Header_Block used for one of the next steps					

CLA:	The CLA byte must be set to xCh
I_RAND:	The internal random number I_RAND may be used with SM (defined in the belonging SM object). If I_RAND is not used, then the I_RAND field is empty (n=0). If I_RAND is used, then its length must be a multiple of 8 bytes and I_RAND must have been requested by the host/terminal using GET CHALLENGE before.
Padding Bytes	As SM uses only DES-based algorithms, there are 4 padding bytes necessary always.
r	r specifies the Net_LE

2nd Step Building the Plain_Text_Object and the Plain_Text_Block:

Command to be used for SM_SIG (i.e command without SM)

CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
					Used for next action	

Building the Plain_Text_Object and the Plain_Text_Block

Plain_Text_Object			Padding
T	**L**	**V**	**(according to SIG algorithm)**
80h / 81h	u	**Data**	Padding Bytes
1 byte	1 byte	u bytes	
(u+2) bytes			s bytes
(u+2+s) bytes			
Plain_Text_Block used for one of the following steps **(built for TAG=81h only!)**			

Plain_Text_Block	If **TAG=81h** in the Plain_Text_Object, then the Plain_Text_Block is **built** as shown above. If **TAG=80h** in the Plain_Text_Object, then the Plain_Text_Block is **empty**, i.e. its length is 0 bytes.

3rd Step Building the Net_LE_MAC_Object and the Net_LE_MAC_Block (if necessary):

Command to be used for SM_SIG (i.e command without SM)

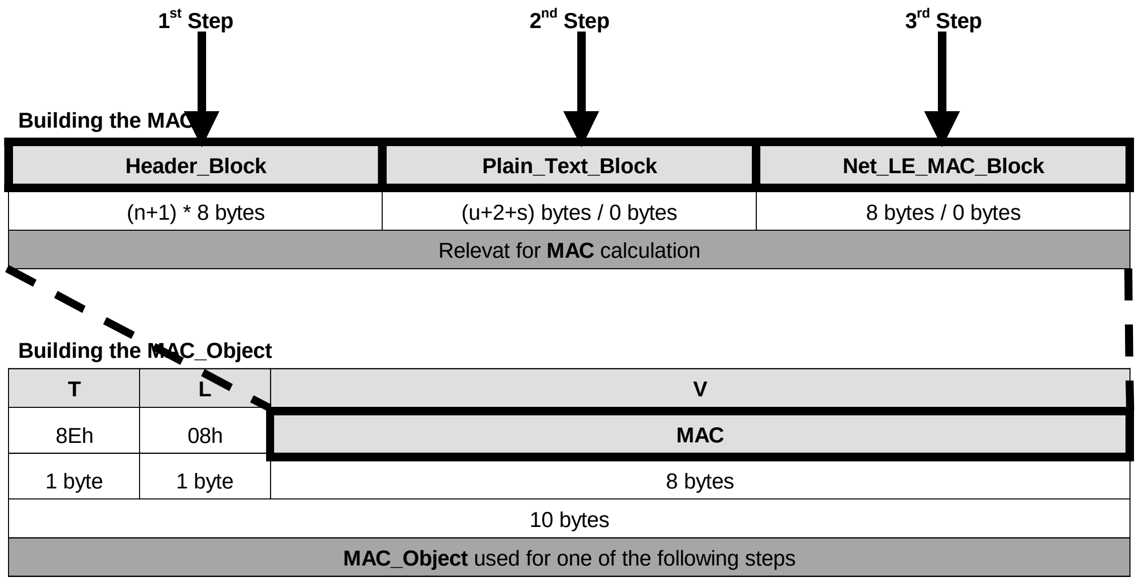
CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
						Used for next action

Building the Net_LE_MAC_Object and the Net_LE_MAC_Block

Net_LE_MAC_Object			Padding
T	**L**	**V**	**(according to SIG algorithm)**
97h	01h	**r = Net_LE**	Padding Bytes
1 byte	1 byte	1 byte	
3 bytes			5 bytes
8 bytes			
Net_LE_MAC_Block used for one of the following steps			

r	r specifies the Net_LE

4th Step Building the MAC and the MAC_Object:



Building the MAC

Header_Block	Plain_Text_Block	Net_LE_MAC_Block
(n+1) * 8 bytes	(u+2+s) bytes / 0 bytes	8 bytes / 0 bytes
Relevat for **MAC** calculation		

Building the MAC_Object

T	L	V
8Eh	08h	**MAC**
1 byte	1 byte	8 bytes
10 bytes		
MAC_Object used for one of the following steps		

Plain_Text_Block	If **TAG=81h** in the Plain_Text_Object, then the Plain_Text_Block **exists** as shown above. If **TAG=80h** in the Plain_Text_Object, then the Plain_Text_Block is **empty**, i.e. its length is 0 bytes.
Net_LE_MAC Block	Note that the Net_LE_MAC_Block is optional and may be empty.
MAC calculation	The order of occurrence for the Plain_Text_Object and the Net_LE_MAC_Object in the resulting command data field (see last step of this section) is free. In order to calculate a correct MAC, the order of occurrence for the Plain_Text_Object and the Net_LE_MAC_Object inside the command data field must be the same as the order of occurrence for the Plain_Text_Block and the Net_LE_MAC_Block during MAC calculation.

5th Step Building the Net_LE_Object (if necessary)

Command to be used for SM_SIG (i.e command without SM)

CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
						Used for next action

Building the Net_LE_Object

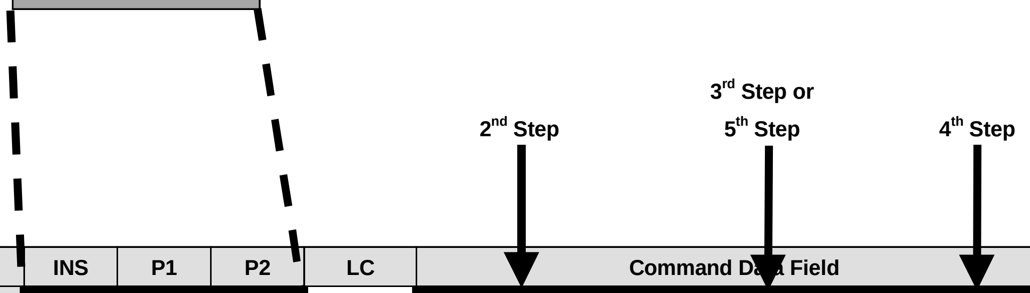
T	L	V
96h	01h	**r = Net_LE**
1 byte	1 byte	1 byte
3 bytes		
Net_LE_Object used for one of the following steps		

r	r specifies the Net_LE

6th Step Building the Command

Command to be used for SM_SIG (i.e command without SM)

CLA	**INS**	**P1**	**P2**	**LC**	**Command Data Field**	**LE**
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
	Used for next action					



CLA	**INS**	**P1**	**P2**	**LC**	**Command Data Field**			**LE**
xCh				LC=w	**Plain_Text_Object**	**Net_LE_MAC_Object** (*) **/ Net_LE_Object** (**) **/ empty** (***)	**MAC_Object**	LE=t
1 byte	1 byte	1 byte	1 byte	1 byte	(u+2) bytes	3 bytes (*) / 3 bytes (**) / 0 bytes (***)	10 bytes	1 byte
					w=(u+15) bytes (*) / w=(u+15) bytes (**)/ w=(u+12) bytes (***) /			
Command_SM_SIG Format in SM Mode xCh								

CLA:	The CLA byte must be set to xCh
(*), (**), (***)	In the command data field either the Net_LE_MAC_Object (TAG=97h; (*)) or the Net_LE_Object (TAG=96h; (**)) or none (i.e. (***)) can be specified. If both objects are sent with the command data field this leads to unpredictable results.
MAC calculation	The order of occurrence for the Plain_Text_Object and the Net_LE_MAC_Object in the resulting command data field is free. In order to calculate a correct MAC, the order of occurrence for the Plain_Text_Object and the Net_LE_MAC_Object inside the command data field must be the same as the order of occurrence for the Plain_Text_Block and the Net_LE_MAC_Block during MAC calculation (see 4th step).
r	r specifies the Net_LE
t	This is the length LE of the requested response data field taking into account all auxiliary bytes necessary for response-SM.

2.5.2.6 Command-SM_ENC_SIG (SM Mode xCh)

Description This section describes how to build command-SM using SM_ENC_SIG for SM mode xCh (i.e. first SM_ENC and then a following SM_SIG).

1st Step **Building the Header_Block:**

Command to be used for SM_ENC_SIG (i.e command without SM)

CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
	Used for next action					

Internal Random Number	CLA	INS	P1	P2	Padding (according to SIG algorithm)
I_RAND	**xCh**				Padding Bytes
n * 8 bytes / empty	1 byte	1 byte	1 byte	1 byte	4 bytes
(n+1) * 8 bytes					
Header_Block used for one of the next steps					

CLA: The CLA byte must be set to xCh

I_RAND: The internal random number I_RAND may be used with SM (defined in the belonging SM object). If I_RAND is not used, then the I_RAND field is empty (n=0). If I_RAND is used, then its length must be a multiple of 8 bytes and I_RAND must have been requested by the host/terminal using GET CHALLENGE before.

Padding Bytes As SM uses only DES-based algorithms, there are 4 padding bytes necessary always.

r r specifies the Net_LE

2nd Step Building the Cipher_Text_Object and the Cipher_Text_Block:

Command to be used for SM_ENC_SIG (i.e command without SM)

CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
					Used for next action	

ENC calculation for SM_ENC

Command Data Field	ISO-Padding (for all ENC algorithms)
Data	80..00h
u bytes	p = 1..8 bytes
(u+p) bytes	
Relevant for **ENC** calculation	

Including ENC into the Cipher_Text_Object

T	L	V	
		Padding Indicator	Enciphered Data
86h / 87h	u+p+1	01h	**ENC**
1 byte	1 byte	1 byte	(u + p) bytes
(u + p + 3) bytes			
Cipher_Text_Object to be used for the next action			

Building the Cipher_Text_Block

Cipher_Text_Object	Padding (according to SIG algorithm)
Cipher_Text_Object	Padding Bytes
(u+p+3) bytes	s bytes
(u+p+3+s) bytes	
Cipher_Text_Block used for one of the following steps **(built for TAG=87h only!)**	

Cipher_Text_Object	The enciphered command data ENC must be sent inside a Cipher_Text_Object (i.e. TAG=86h or TAG=87h). Inside the value field of the Cipher_Text_Object the fixed padding indicator 01h must be added.
Cipher_Text_Block	If **TAG=87h** in the Cipher_Text_Object, then the Cipher_Text_Block is **built** as shown above. If **TAG=86h** in the Cipher_Text_Object, then the Cipher_Text_Block is **empty**, i.e. its length is 0 bytes.
r	r specifies the Net_LE

3rd Step Building the Net_LE_MAC_Object and the Net_LE_MAC_Block (if necessary):

Command to be used for SM_ENC_SIG (i.e command without SM)

CLA	**INS**	**P1**	**P2**	**LC**	**Command Data Field**	**LE**
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
						Used for next action

Building the Net_LE_MAC_Object and the Net_LE_MAC_Block

Net_LE_MAC_Object			**Padding (according to SIG algorithm)**
T	**L**	**V**	
97h	01h	**r = Net_LE**	Padding Bytes
1 byte	1 byte	1 byte	
3 bytes			5 bytes
8 bytes			
Net_LE_MAC_Block used for one of the following steps			

r	r specifies the Net_LE

4th Step Building the MAC and the MAC_Object:

1st Step **2nd Step** **3rd Step**

Building the MAC

Header_Block	**Cipher_Text_Block**	**Net_LE_MAC_Block**
(n+1) * 8 bytes	(u+p+3+s) bytes / 0 bytes	8 bytes / 0 bytes
Relevat for **MAC** calculation		

Building the MAC_Object

T	**L**	**V**
8Eh	08h	**MAC**
1 byte	1 byte	8 bytes
10 bytes		
MAC_Object used for one of the following steps		

Cipher_Text_Block	If **TAG=87h** in the Cipher_Text_Object, then the Cipher_Text_Block **exists** as shown above. If **TAG=86h** in the Cipher_Text_Object, then the Cipher_Text_Block is **empty**, i.e. its length is 0 bytes.
Net_LE_MAC Block	Note that the Net_LE_MAC_Block is optional and may be empty.
MAC calculation	The order of occurrence for the Cipher_Text_Object and the Net_LE_MAC_Object in the resulting command data field (see last step of this section) is free.

In order to calculate a correct MAC, the order of occurrence for the Cipher_Text_Object and the Net_LE_MAC_Object inside the command data field must be the same as the order of occurrence for the Cipher_Text_Block and the Net_LE_MAC_Block during MAC calculation.

5th Step Building the Net_LE_Object (if necessary):

Command to be used for SM_ENC (i.e command without SM)

CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
						Used for next action

Building the Net_LE_Object

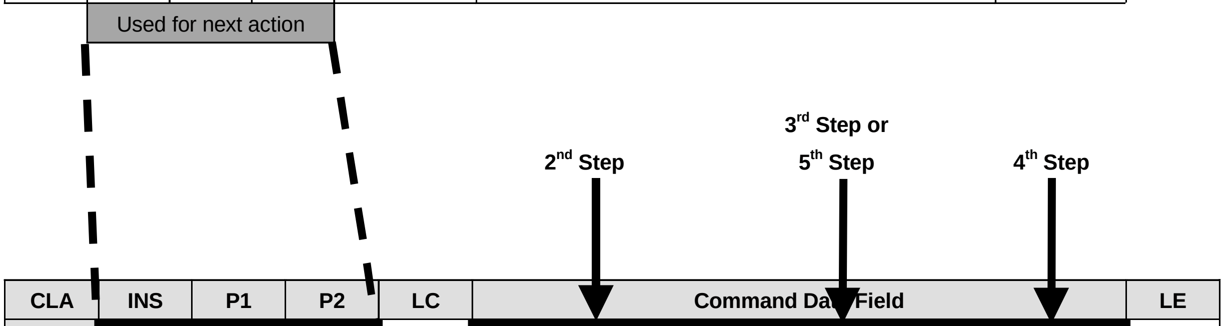
Net_LE_Object		
T	**L**	**V**
96h	01h	**r = Net_LE**
1 byte	1 byte	1 byte
3 bytes		
Net_LE_Object to be used for one of the following steps		

r r specifies the Net_LE

6th Step Building the Command

Command to be used for SM_SIG (i.e command without SM)

CLA	**INS**	**P1**	**P2**	**LC**	**Command Data Field**	**LE**
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
	Used for next action					



CLA	**INS**	**P1**	**P2**	**LC**	**Command Data Field**			**LE**
xCh				LC=w	**Cipher_Text_Object**	**Net_LE_MAC_Object (*) / Net_LE_Object (**) / empty (***)**	**MAC_Object**	LE=t
1 byte	1 byte	1 byte	1 byte	1 byte	(u+p+3) bytes	3 bytes (*) / 3 bytes (**) / 0 bytes (***)	10 bytes	1 byte
					w=(u+p+16) bytes (*) / w=(u+p+16) bytes (**)/ w=(u+p+13) bytes (***) /			
Command_SM_SIG Format in SM Mode xCh								

CLA:	The CLA byte must be set to xCh
(*), (**), (***)	In the command data field either the Net_LE_MAC_Object (TAG=97h; (*)) or the Net_LE_Object (TAG=96h; (**)) or none (i.e. (***)) can be specified. If both objects are sent with the command data field this leads to unpredictable results.
MAC calculation	The order of occurrence for the Cipher_Text_Object and the Net_LE_MAC_Object in the resulting command data field is free. In order to calculate a correct MAC, the order of occurrence for the Cipher_Text_Object and the Net_LE_MAC_Object inside the command data field must be the same as the order of occurrence for the Cipher_Text_Block and the Net_LE_MAC_Block during MAC calculation (see 4th step).
r	r specifies the Net_LE
t	This is the length LE of the requested response data field taking into account all auxiliary bytes necessary for response-SM.

2.5.2.7 Command-SM_suppressed

Description This section describes for SM mode xCh how to build command-SM, which is suppressed by the corresponding SM definitions.

1st Case If the command data field (of the command without SM) is empty, for Command-SM_suppressed the command data field is empty as well.

2nd Case If the command data field (of the command without SM) is **not** empty, for Command-SM_suppressed the command data field is built as follows:

Command to be used for SM_SIG (i.e command without SM)

CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
					Used for next action	

Building the Plain_Text_Object and the Plain_Text_Block

T	L	V
80h / 81h	u	**Data**
1 byte	1 byte	u bytes
(u+2) bytes		
Plain_Text_Object used for the next action		

Command to be used for SM_SIG (i.e command without SM)

CLA	INS	P1	P2	LC	Command Data Field	LE
xCh				LC=w	**Plain_Text_Object**	LE=t
1 byte	1 byte	1 byte	1 byte	1 byte	w=(u+2) bytes	1 byte
Command_SM_suppressed Format in SM Mode xCh for 2nd Case						

CLA:	The CLA byte must be set to xCh
r	r specifies the Net_LE
t	This is the length LE of the requested response data field taking into account all auxiliary bytes necessary for response-SM.

2.5.2.8 Response-SM_ENC (SM Mode xCh)

Description This section describes how to build response-SM only using SM_ENC for SM mode xCh.

Single Step **Building the Cipher_Text_Object and the response:**

Response to be used for SM_ENC (i.e response without SM)

Response Data Field
Response
r bytes
Used for next action

ENC calculation for SM_ENC

Response Data Field	**ISO-Padding (for all ENC algorithms)**
Response	80..00h
r bytes	p = 1..8 bytes
(r+p) bytes	
Relevant for **ENC** calculation	

Including ENC into the Cipher_Text_Object

T	**L**	**V**	
		Padding Indicator	**Enciphered Response**
86h / 87h	u+p+1	01h	**ENC**
1 byte	1 byte	1 byte	(r + p) bytes
(r+p+3) bytes			
Cipher_Text_Object to be used for the next action			

Including Cipher_Text_Object into response

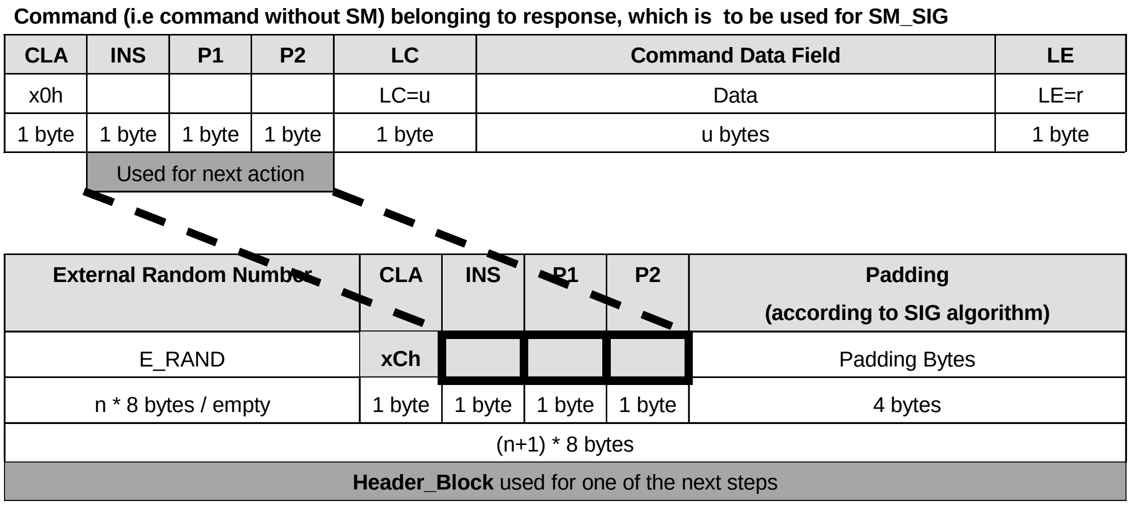
Response Data Field
Cipher_Text_Object
t = (r+p+3) bytes
Response-SM_ENC Format in SM Mode xCh

Cipher_Text_Object	The enciphered response ENC must be sent inside a Cipher_Text_Object (i.e. TAG=86h or TAG=87h). Inside the value field of the Cipher_Text_Object the fixed padding indicator 01h must be added.
r	r specifies the Net_LE
t	This is the length LE of the requested response data field taking into account all auxiliary bytes necessary for response-SM. LE is specified in the belonging command.

2.5.2.9 Response-SM_SIG (SM Mode xCh)

Description This section describes how to build response-SM only using SM_SIG for SM mode xCh.

1st Step **Building the Header_Block:**



Command (i.e command without SM) belonging to response, which is to be used for SM_SIG

CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
	Used for next action					

External Random Number	CLA	INS	P1	P2	Padding (according to SIG algorithm)
E_RAND	xCh				Padding Bytes
n * 8 bytes / empty	1 byte	1 byte	1 byte	1 byte	4 bytes
(n+1) * 8 bytes					
Header_Block used for one of the next steps					

CLA:	The CLA byte must be set to xCh
E_RAND:	The external random number E_RAND may be used with SM (defined in the belonging SM object). If E_RAND is not used, then the E_RAND field is empty (n=0). If E_RAND is used, then its length must be a multiple of 8 bytes and E_RAND must have been sent by the host/terminal using GIVE RANDOM before.
Padding Bytes	As SM uses only DES-based algorithms, there are 4 padding bytes necessary always.
r	r specifies the Net_LE

2nd Step Building the Plain_Text_Object and the Plain_Text_Block:

Response to be used for SM_ENC (i.e response without SM)

Response Data Field
Response
r bytes
Used for next action

Building the Plain_Text_Object and the Plain_Text_Block

Plain_Text_Object			**Padding (according to SIG algorithm)**
T	**L**	**V**	
80h / 81h	u	**Response**	Padding Bytes
1 byte	1 byte	r bytes	
(r+2) bytes			s bytes
(r+2+s) bytes			
Plain_Text_Block used for one of the following steps **(built for TAG=81h only!)**			

Plain_Text_Block	If **TAG=81h** in the Plain_Text_Object, then the Plain_Text_Block is **built** as shown above. If **TAG=80h** in the Plain_Text_Object, then the Plain_Text_Block is **empty**, i.e. its length is 0 bytes.
r	r specifies the Net_LE

3rd Step Building the MAC and the MAC_Object:

1st Step **2nd Step**

Building the MAC

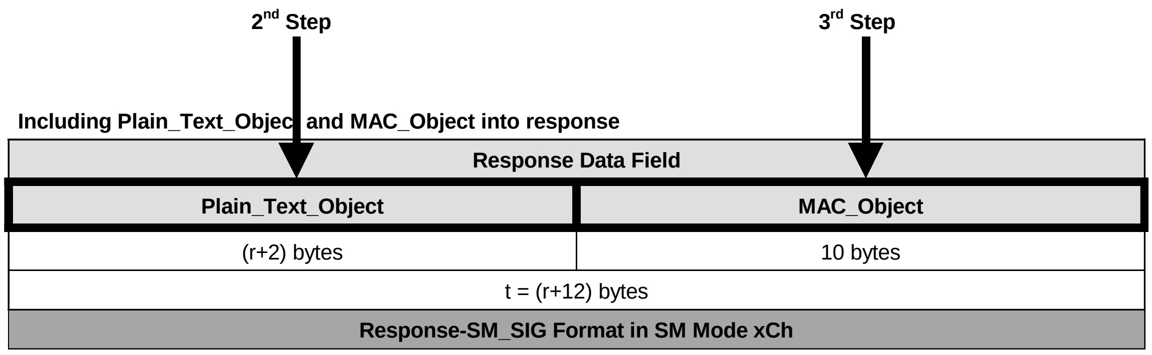
Header_Block	**Plain_Text_Block**
(n+1) * 8 bytes	(r+2+s) bytes / 0 bytes
Relevat for **MAC** calculation	

Building the MAC_Object

T	**L**	**V**
8Eh	08h	**MAC**
1 byte	1 byte	8 bytes
10 bytes		
MAC_Object used for one of the following steps		

Plain_Text_Block	If **TAG=81h** in the Plain_Text_Object, then the Plain_Text_Block **exists** as shown above. If **TAG=80h** in the Plain_Text_Object, then the Plain_Text_Block is **empty**, i.e. its length is 0 bytes.

4th Step Building the Response:



Response Data Field	
Plain_Text_Object	**MAC_Object**
(r+2) bytes	10 bytes
t = (r+12) bytes	
Response-SM_SIG Format in SM Mode xCh	

r	r specifies the Net_LE
t	This is the length LE of the requested response data field taking into account all auxiliary bytes necessary for response-SM. LE is specified in the belonging command.

2.5.2.10 Response-SM_ENC_SIG (SM Mode xCh)

Description This section describes how to build response-SM only using SM_ENC_SIG for SM mode xCh (i.e. first SM_ENC and then a following SM_SIG).

1st Step **Building the Header_Block:**

Command (i.e command without SM) belonging to response, which is to be used for SM_ENC_SIG

CLA	INS	P1	P2	LC	Command Data Field	LE
x0h				LC=u	Data	LE=r
1 byte	1 byte	1 byte	1 byte	1 byte	u bytes	1 byte
	Used for next action					

External Random Number	CLA	INS	P1	P2	Padding (according to SIG algorithm)
E_RAND	**xCh**				Padding Bytes
n * 8 bytes / empty	1 byte	1 byte	1 byte	1 byte	4 bytes
(n+1) * 8 bytes					
Header_Block used for one of the next steps					

CLA:	The CLA byte must be set to xCh
E_RAND:	The external random number E_RAND may be used with SM (defined in the belonging SM object). If E_RAND is not used, then the E_RAND field is empty (n=0). If E_RAND is used, then its length must be a multiple of 8 bytes and E_RAND must have been sent by the host/terminal using GIVE RANDOM before.
Padding Bytes	As SM uses only DES-based algorithms, there are 4 padding bytes necessary always.
r	r specifies the Net_LE

2nd Step Building the Cipher_Text_Object and the Cipher_Text_Block:

Response to be used for SM_ENC_SIG (i.e response without SM)

Response Data Field
Response
r bytes
Used for next action

ENC calculation for SM_ENC_SIG

Response Data Field	**ISO-Padding (for all ENC algorithms)**
Response	80..00h
r bytes	p = 1..8 bytes
(r+p) bytes	
Relevant for **ENC** calculation	

Including ENC into the Cipher_Text_Object

T	**L**	**V**	
		Padding Indicator	**Enciphered Response**
86h / 87h	u+p+1	01h	**ENC**
1 byte	1 byte	1 byte	(r+p) bytes
(r+p+3) bytes			
Cipher_Text_Object to be used for the next action			

Building the Cipher_Text_Block

Cipher_Text_Object	**Padding (according to SIG algorithm)**
Cipher_Text_Object	Padding Bytes
(r+p+3) bytes	s bytes
(r+p+3+s) bytes	
Cipher_Text_Block used for one of the following steps **(built for TAG=87h only!)**	

Cipher_Text_Object	The enciphered response ENC must be sent inside a Cipher_Text_Object (i.e. TAG=87h). Inside the value field of the Cipher_Text_Object the fixed padding indicator 01h must be added.
Cipher_Text_Block	If **TAG=87h** in the Cipher_Text_Object, then the Cipher_Text_Block is **built** as shown above. If **TAG=86h** in the Cipher_Text_Object, then the Cipher_Text_Block is **empty**, i.e. its length is 0 bytes.
r	r specifies the Net_LE

3rd Step Building the MAC and the MAC_Object:

1st Step 2nd Step

Building the MAC

Header_Block	**Cipher_Text_Block**
(n+1) * 8 bytes	(r+p+3+s) bytes / 0 bytes
Relevat for **MAC** calculation	

Building the MAC_Object

T	**L**	**V**
8Eh	08h	**MAC**
1 byte	1 byte	8 bytes
10 bytes		
MAC_Object used for one of the following steps		

Cipher_Text_Block	If **TAG=87h** in the Cipher_Text_Object, then the Cipher_Text_Block **exists** as shown above. If **TAG=86h** in the Cipher_Text_Object, then the Cipher_Text_Block is **empty**, i.e. its length is 0 bytes.

4th Step Building the Response:

2nd Step 3rd Step

Including Cipher_Text_Object and MAC_Object into response

Response Data Field	
Cipher_Text_Object	**MAC_Object**
(r+p+3) bytes	10 bytes
t = (r+p+13) bytes	
Response-SM_ENC_SIG Format in SM Mode xCh	

r	r specifies the Net_LE
t	This is the length LE of the requested response data field taking into account all auxiliary bytes necessary for response-SM. LE is specified in the belonging command.

2.5.2.11 Response-SM_suppressed

Description This section describes for SM mode xCh how to build response-SM, which is suppressed by the corresponding SM definitions.

General The response data field is **not changed** for Response-SM_suppressed (i.e. in comparison with the response data field of the response without SM).

2.6 Packages (CardOS Extensions)

Description

Extensions to CardOS/M4 are called packages. Packages represent functions or commands which are added to the standard ROM masked CardOS/M4. CardOS/M4 supports two kinds of packages:

- **System Packages** and
- **Application Packages**.

Definitions

System packages are stored in images inside the filesystem of CardOS/M4.

Application packages are stored in so-called *CODE* files, which can be created and updated exactly like *BINARY* files.

General Rules

Packages must be loaded and activated in **one and the same** life cycle phase. This means:

- **System packages** can only be installed and activated in life cycle phase ***INITIALIZATION***.
- **Application packages** can only be installed and activated in life cycle phase ***ADMINISTRATION***.

2.6.1 System Packages

System Packages

System packages are stored in images inside the filesystem of CardOS/M4. They are installed and activated in the life cycle phase *INITIALIZATION* using the INITIALIZE EEPROM command respectively LOAD EXECUTABLE command.

Installation of a System Package

For installation of a system package the following steps must be performed:

1st Changing to life cycle phase *INITIALIZATION*.

2nd Installing the image of the system package using the INITIALIZE EEPROM command.

Note

Generally during installation **package data must not be changed** anyhow as package activation via LOAD EXECUTABLE will fail, if package data are changed.

After the installation steps the package is **installed** but **not activated**. This means that the functions/commands, which are programmed in the package, are not available at this moment.

Activate a System Package

In order to activate the system package (i.e. link the system package to the CardOS/M4 operating system) the following steps must be performed:

1st Performing the LOAD EXECUTABLE command using the Package_Checksum (given by the manufacturer of the system package) necessary for system package activation.
Note that
- for **CardOS/M4.00** the LOAD EXECUTABLE command must be sent **without SM** whereas
- for **CardOS/M4.01** the LOAD EXECUTABLE command must be sent **using SM** mode x4h with the *PackageLoadKey*.

The command checks the Package_Checksum using the data of the system package and the so-called *PackageLoadKey*
- If the Package_Checksum verification fails the error counter of the *PackageLoadKey* will be decreased and an error code will be issued.
- If the Package_Checksum verification is successful the system package will be activated for CardOS/M4.

After package activation the existence of the system package can be checked using the GET DATA command (in MODE P2=88h).

Uninstall a System Package

In order to uninstall a system package (i.e. remove the system package functions/commands from the CardOS/M4 operating system) the UNINSTALL PACKAGE command must be used.

2.6.2 Application Packages

Application Packages

Application packages are stored in so-called *CODE* files, which can be created and updated exactly like *BINARY* files.

Installation of an Application Package

In order to install an application package the following steps must be performed:

1st Changing to life cycle phase *ADMINISTRATION*.

2nd Creation of a *CODE* file using the CREATE FILE command. If the application package shall be removed from the smart card later, the AC *DELETE* of the *CODE* file must be set accordingly during *CODE* file creation.

3rd Filling the *CODE* file with the application package data using the UPDATE BINARY command.

4th In order to activate an application package with a succeeding LOAD EXECUTABLE command the belonging *CODE* file must be selected.

Note

Generally during installation **package data must not be changed** anyhow as package activation via LOAD EXECUTABLE will fail, if package data are changed.

After the installation steps the package is **installed** but **not activated**. This means that the functions/commands, which are programmed in the package, are not available at this moment.

Activate an Application Package

In order to activate the application package (i.e. link the application package to the CardOS/M4 operating system) the following steps must be performed:

1st The access right referenced by the AC *EXECUTE* of the MF must be acquired performing the belonging security test. This security test is necessary, because any application package can be activated via the LOAD EXECUTABLE command. LOAD EXECUTABLE can only be executed, if the access right referenced by the MF’s AC *EXECUTE* is granted in the MF.

2nd The CODE file must be selected.

3rd Performing the LOAD EXECUTABLE command using the Package_Checksum (given by the manufacturer of the application package) necessary for application package activation.
Note that
- for **CardOS/M4.00** the LOAD EXECUTABLE command must be sent **without SM** whereas
- for **CardOS/M4.01** the LOAD EXECUTABLE command must be sent **using SM** mode x4h with the *PackageLoadKey*.

The command checks the Package_Checksum using the *CODE* file data and the so-called *PackageLoadKey*
- If the Package_Checksum verification fails the error counter of the *PackageLoadKey* will be decreased and an error code will be issued.
- If the Package_Checksum verification is successful the application package will be activated for CardOS/M4 and the belonging *CODE* file will be deactivated in order to prevent any change of the *CODE* file data. After deactivation of the *CODE* file

a reactivation of the CODE file using the ACTIVATE FILE command is not possible.

After package activation the existence of the application package can be checked using the GET DATA command (in MODE P2=88h).

Uninstall an Application Package

In order to uninstall an application package (i.e. remove the application package functions/commands from the CardOS/M4 operating system following steps must be performed:

1st The access right referenced by the AC *DELETE* of the *CODE* file must be acquired performing the belonging security test. This security test is necessary, because any application package is uninstalled via the UNINSTALL PACKAGE command. UNINSTALL PACKAGE can only be executed, if the access right referenced by the AC *DELETE* of the *COD*E file is granted in the *current DF*.

2nd The UNINSTALL PACKAGE command must be performed using the PID (i.e. Package ID).

2.6.3 Available Packages

Description

All available Packages are application packages, which for CardOS/M4.00 are delivered with the CardOS/M4.00 Packages & Release Notes.
For CardOS/M4.01 the CardOS/M4.01 Packages & Release Notes are relevant.
Make sure that the **newest** releases are used always. In the **belonging** manuals the newest information about the packages can be found.

Available and Planned Packages

Table 2.6.3-1 lists the packages for CardOS/M4.00 and CardOS/M4.01.

Table 2.6.3-1 Available Packages

CardOS Version	Package	Comment
CardOS/M4.00	Delete_File_Package	Adds the command: DELETE FILE
	Uninstall_Package	Adds the command: UNINSTALL PACKAGE
	PCSC_Package	Adds the commands: DIRECTORY RESET SECURITY STATUS
	DSA_Package (1)	Adds the algorithms: DSA and DSA_SHA-1
	Generate_Key_Pair_Package	Adds the command: GENERATE KEY PAIR
	Manage_Channel_Package	Adds the command: MANAGE CHANNEL
	LeadingZeroes_Package	Adds the mechanism: LeadingZeroes for compatibility with CardOS/M3.0 (see 2.4.2.2)
	UseCounter_Package	Adds the mechanisms using: DeathBit x (see 2.3.1 FLAGS byte) and UseCounter mechanism after a successful test only (see 2.3.1 USECOUNT byte).
	DSI_Extension_Package	Adds the DSI mechanism for the algorithms: RSA_SIG, DSA and RSA_PURE_SIG (see PSO_VDS and PSO_CDS commands)
	Sign_By_Decryption_Key_Package	Adds the command: SIGN BY DECRYPTION KEY
	RSA_Extention_Package	Adds the additional strengthen functionality for RSA algorithms
	Service_package_01	Improves the smart card security
CardOS/M4.01	Sign_By_Decryption_Key_Package(1)	Adds the command: SIGN BY DECRYPTION KEY
	DSA_Package (1)	Adds the algorithms: DSA and DSA_SHA-1
	Service_package_02	Improves the smart card security

(1): on request

This is a empty page!

3 Commands

3.1 Introduction and Overview

General

This chapter describes the commands of CardOS/M4.

Many of the commands described can be used with and without SM (secure messaging). In order to become familiar with the commands faster the command data field and the response data field are described for each command **without using SM** if this is possible for the respective command. If a command can only be used with SM then the description of the command data field and the response data field takes this into account.

For the options description the following is assumed:

- P3 specifies, whether the P3 byte of the T=0 transmission protocol will be interpreted as LC or as LE (**Note**: On request for CardOS/M4 the T=0 transmission protocol can be made available.).
- Tr specifies, whether the command can (Tr=ON) or cannot (Tr=OFF) be performed during a transaction. Transactions will be supported with an additional package to be delivered in the future.
- AC and SM define the AC or the SM definitions used.

Data Structures

File Organization:

The categories DF and EF are implemented in CardOS/M4. EFs may be working EFs of the following types:

- LINEAR FIXED
- CYCLIC FIXED
- LINEARE TLV
- BINARY and
- CODE

File Referencing Methods:

File IDs (FIDs) are 16 bits long. The FID=3F00h is reserved for the MF. Inside a DF all FIDs must be unique. Using selection by path the FID of the DF (see SELECT FILE, Mode 09h) respectively of the MF (see SELECT FILE, Mode 08h) is not included in the path specification.

Using a SFI (Short File Identifier) the FID to be used is built adding the constant value **FE00h** to the SFI specification of the belonging command. DF Names (i.e. AIDs) must be unique inside the whole smart card. CardOS/M4 checks the uniqueness of AIDs.

Data Referencing Methods:

For the access to record oriented files two record pointers exist:

- The logically first record and
- the current record.

Using CYCLIC FIXED files the logically first record is the record, which was written in APPEND mode last. For all other record oriented files the logically first record is the record, which was physically written first.

For LINEAR TLV files the record identifier is the TAG of the TLV record. For all other record oriented files the record identifier is the logical number of the record.

The table 3.1-1 gives an overview of the CardOS/M4 commands.

Table 3.1-1 Overview of the CardOS/M4 commands.

Command			Command available in **Life Cycle Phase**						Command Execution depending on **AC**	Command using **SM** according to **SM Definition**
	CLA [hex]	**INS [hex]**	MANUFACTURING	INITIALIZATION	PERSONALIZATION	ADMINISTRATION	OPERATIONAL	DEATH		
ACTIVATE FILE	00h	44h				X	X		ACTIVATE (file)	none
APPEND RECORD	0xh	E2h				X	X		APPEND (file)	APPEND (file)
CHANGE KEY	84h	24h	X	X	X				none	none (Mode x4h)
CHANGE REFERENCE DATA	0xh	24h				X	X		CHANGE (BS Object)	CHANGE (BS Object)
CREATE FILE	0xh	E0h				X			CREATE (current DF)	CREATE (current DF)
DEACTIVATE FILE	00h	04h				X	X		DEACTIVATE (file)	none
DECREASE	8xh	30h				X	X		DECREASE (file)	DECREASE-IN/-OUT (file)
DELETE FILE (01)	0xh	E4h				X			DELETE (file)	DELETE (file)
DIRECTORY (01)	80h	16h				X	X		none	none
ERASE FILES	84h	06h	X	X	X	X			none	none (Mode x4h)
EXTERNAL AUTHENTICATE	0xh	82h				X	X		USE (BS Object)	USE-IN (BS Object)
FORMAT	84h	40h	X						none	none (Mode x4h)
GENERATE KEY PAIR (01)	0xh	46h				X			GENKEY (BS Object) AND APPEND (file for public key storage)	GENKEY (current DF)
GET CHALLENGE	00h	84h				X	X		none	none
GET DATA	00h	CAh	X	X	X	X	X	X	none	none
GIVE RANDOM	80h	86h				X	X		none	none
INCREASE	8xh	32h				X	X		INCREASE (file)	INCREASE-IN/-OUT (file)
INITIALIZE EEPROM	80h / 84h	02h		X					none	none, (Mode x4h)
INITIALIZE END	84h	00h		X					none	none (Mode x4h)
INTERNAL AUTHENTICATE	0xh	88h				X	X		USE (BS Object)	USE-IN/-OUT (BS Object)
LOAD EXECUTABLE	80h	20h		X		X			EXECUTE (MF) / none(*)	none
MANAGE CHANNEL (01)	00h	70h				X	X		none	none
MANAGE SECURITY ENVIRONMENT (MSE)	00h	22h				X	X		CHANGE (CSE Object)	none
PERFORM SECURITY OPERATION (PSO)	0xh / 1xh	2Ah				X	X		none, USE (BS Object)	none, USE-IN (BS Object)
PERSONALIZE	80h / 84h	08h			X				none	none (Mode x4h)
PHASE CONTROL	80h	10h				X	X		none, LCYCLE (current DF)	none
PUT DATA	0xh	DAh				X			APPEND/UPDATE (current DF), ADMIN (file)	APPEND/UPDATE (current DF), ADMIN (file)
READ BINARY	0xh	B0h				X	X		READ (file)	READ-OUT/-IN (file)
READ RECORD	0xh	B2h				X	X		READ (file)	READ-OUT/-IN (file)
RESET RETRY COUNTER	0xh	2Ch				X	X		UNBLOCK (BS Object)	UNBLOCK (BS Object)
RESET SECURITY STATUS (01)	80h	EAh				X	X		none	none
SELECT FILE	00h	A4h				X	X		none	none
UNINSTALL PACKAGE (01)	80h / 84h(*)	E4h		X		X			DELETE (CODE file) / none (*)	none / none (Mode x4h) (*)
UPDATE BINARY	0xh	D6h				X	X		UPDATE (file)	UPDATE (file)
UPDATE RECORD	0xh	DCh				X	X		UPDATE (file)	UPDATE (file)
VERIFY	0xh	20h				X	X		USE (BS Object)	USE-IN/-OUT (BS Object)

(*) For system packages only

(01) CardOS/M4.00: Available, if belonging package was loaded and activated before.
CardOS/M4.01: Available always

3.2 ACTIVATE FILE

Command ACTIVATE FILE

Function Activate a deactivated file or file tree.

Standard / Use ISO 7816-9 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
00h	44h	00h	00h

Data

LC	Command Data Field	LE
*	empty	./.

* Length of subsequent command data field

Response **./.**

Description This command reactivates the current file in the file system. The command can be performed even, if the file or the DF is already active.

For deactivation of a file or file tree the DEACTIVATE FILE command can be used.

Prerequisites The command can only be executed, if the right referenced by the file's *AC ACTIVATE* is granted in the *security status* of the *current DF*.
The command cannot be used on *CODE* files.

Options P3=LC, Tr=ON, AC=ACTIVATE (file), SM=none

3.3 APPEND RECORD

Command APPEND RECORD

Function Create new records

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh	E2h	00h	EF-ID

Data

LC	Command Data Field	LE
	Record_Data	
*	(var. length)	./.

* Length of subsequent command data field

Response ./.

Description This command creates a new record in the file, which is currently selected, or, if EF-ID was specified, in the file, which is referenced by the parameter.

A file ID can be transferred in the parameter EF-ID. See table 3.2-1 for format. The referenced file is selected then.

Table 3.2-1 EF-ID Byte P2

7	6	5	4	3	2	1	0	¬ Bit Number / Meaning ¯
0	*0*	*0*	*0*	*0*	0	0	0	Use current EF
p	*p*	*p*	*p*	*p*	0	0	0	Use SFI *'p pppp' ('p pppp' = '1 1111' not permitted)*
Any other value								rfu

Using a SFI the FID to be used is built adding the constant value **FE00h** to the specified SFI.

The contents of the new record are transferred into the Record_Data. When a *LINEAR FIXED* or a *CYCLIC FIXED* file is involved, its length (i.e. length of Record_Data) must match the length specified by the belonging CREATE FILE command. When *LINEAR TLV* files are involved, the data field must have a valid TLV format.

If there is not sufficient memory space to store the new record, the command - when a *CYCLIC FIXED* file is involved – is handled the same as the UPDATE RECORD command in PREV mode. An error message is generated for all other types of files.

For all types of files the newly written record becomes the *current record*. For *CYCLIC FIXED* files the newly written record becomes the *logical first record*.

For *LINEAR TLV* files it is assumed that the length field of the record is one byte in length. The TAG byte is not interpreted by APPEND RECORD.

Prerequisites The command can only be executed, if the right referenced by the file’s *AC APPEND* is granted in the *security status* of the *current DF*.
The command can only be used on *LINEAR FIXED*, *CYCLIC FIXED* and *LINEAR TLV* files.

Options P3=LC, Tr=ON, AC=APPEND (file), SM=APPEND (file)

3.4 CHANGE KEY

Command CHANGE KEY

Function Changes system key data to the new key data provided with the command.

Standard / Use Proprietary / ROM-masked; MANUFACTURING, INITIALIZATION, *PERSONALIZATION*

Parameters

CLA	INS	P1	P2
84h	24h	00h / xxh	Key_ID

Data

LC	Command Data Field			LE
	Enciphered(New_Key_Data,	MAC,	Padding$_{ENC}$)	
20h	(16 bytes) (0) / (16/17 bytes) (1)	(8 bytes)	(8 bytes)	./.

(0): CardOS/M4.00
(1): CardOS/M4.01

Response ./.

Description The command can only be executed using SM mode x4h. The New_Key_Data with a length of 16 bytes in CardOS/M4.00 or 16/17 bytes in CardOS/M4.01 is protected with a MAC and enciphered. SM mode x4h uses the old system key in the EEPROM (i.e. the system key to be changed) for MAC verification. After successful MAC verification the old system key data will be replaced by the New_Key_Data.

As system keys are 16 bytes in length, they use an A-B-A key structure. Each system key has an error counter with a maximum of **10**.

- Each time a MAC-verification with a system key fails, the corresponding error counter is decreased. This means that after 10 unsuccessful usages any system key is blocked irreversibly (i.e. error counter = 00h) and all commands using that system key cannot be performed any more. Removal of the card or a card-reset during command execution may result in less available attempts to verify the MAC and may even block the card.
- Each time the MAC-verification of the system keys *StartKey* or *PersonalizationKeys* is successful the error counter of the system key is set to those maximum (i.e. 10). In CardOS/M4.01 the error counter of the *PackageLoadKey* however will never be increased.

Parameter P2 specifies the Key_ID. If P2 specifies the *PersonalizationKey* then parameter P1 specifies the Personalization Number (see table 3.4-1). For the *StartKey* and the *PackageLoadKey* P1 shall be set to 00h.

Table 3.4-1 Key_ID Byte P2 and Parameter P1

P1	P2 = Key_ID	Meaning
00h	00h	StartKey
00h	01h	PackageLoadKey
xxh	02h	PersonalizationKey

In order to prevent misuse of the CardOS/M4 smart cards and for security reasons, it is strongly recommended to change the *StartKey*, which was set by the chip manufacturer.

Version byte If in CardOS/M4.01 the sent New_Key_Data field after secure messaging is performed is 17 bytes long, the last byte will be interpreted as the version byte of the key. A version byte is only stored for the *StartKey* and the *PackageLoadKey*. For *PersonalizationKeys* the version byte will be ignored. This system information can be read with the GET DATA command Mode 96h.

Prerequisites The command is only allowed in life cycle phases *MANUFACTURING, INITIALIZATION* and *PERSONALIZATION*.
The command can only be executed using SM method x4h with the key to be changed.

Options P3=LC, Tr=OFF, AC=none, SM = SM mode x4h with key specified by Key_ID

3.5 CHANGE REFERENCE DATA

Command CHANGE REFERENCE DATA

Function Change data of a BS object

Standard / Use ISO 7816-8 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh	24h	Object-Parms	ID

Data

LC	Command Data Field		LE
	Old_Data	New_Data	
*	m bytes / empty	n bytes	**./.**

* Length of subsequent command data field

Response **./.**

Description This command replaces the data of the BS object referenced by P1/P2 with the New_Data and sets the CurrentErrorCounter to ist maximum.

For **CardOS/M4.00** the following applies:
Except PIN TEST objects CHANGE REFERENCE DATA functions correctly only, if the MINLEN Byte of the BS object is identical with the length of the BS object data, i.e. length specified during PUT DATA_OCI in the BS object data TAG=8Fh.

- As MINLEN is used in order to specify the minimum length of a random number used either with a C/R test object or with a SIGning key for SM, only C/R test objects or SIGning keys for SM can be changed, whose minimum length for the random number matches the length of the key data.
    - If this match is not possible because of requirements from the application, MINLEN specifies the minimum random number length to be used and the key **must not be changed using CHANGE REFERENCE DATA**. In this case the key can only be changed using the PUT DATA_OCI command.

For **CardOS/M4.01** the following applies:
The described CardOS/M4.00-restriction for C/R test objects and SIGning keys for SM does not exist for CardOS/M4.01. This means that any BS object data can be changed using CHANGE REFERENCE DATA. Except for PIN TEST objects the length of the BS object data must match the length of New_Data (n). MINLEN has the following meanings:

- For PIN TEST objects MINLEN specifies the minimum length of the PIN data.
- For C/R test objects and SIGning keys for SM MINLEN specifies the minimum length of the random number to be used with these keys.
- For other BS objects MINLEN has no meaning and shall be set to 00h.

Note: For CardOS/M4 only the data length of a PIN TEST object can be changed using the CHANGE REFERENCE DATA command.

Table 3.5-1 explains the meaning of the Object-Parms Parameter P1.

Table 3.5-1 Object-Parms Byte P1

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	¬ **Bit Number / Meaning** ¯
0								Bit 2^7 must be set to '0'
	u	u	u					Object Usage Bits *'uuu'*
				k	k	k		Object Component Bits *'kkk'*
							y	y=0: Right granted by **implicit** test
								y=1: Right granted by **explicit** test

In P1 the object usage is given in bits *uuu* and the object component is given in bits *kkk* (see PUT DATA_OCI command also). Bit 2^7 must be set to '0'. Bit 2^0 (i.e. y) specifies, whether Old_Data is empty (y=1), or, whether Old_Data is present (y=0).

If y=1 then the right in the AC attribute *CHANGE* of the referenced BS object must be granted in order to perform CHANGE REFERENCE DATA (i.e. the right must have been granted by an explicit test). This y=1 mode can be used for any kind of BS object.

The y=0 mode can only be used for PIN TESTs (i.e. TEST objects with ALGO=PIN (87h)). If y=0 then the right in the AC attribute *CHANGE* of the referenced TEST object will be granted by an implicit test during the CHANGE REFERENCE DATA command.

If y=0 mode is used the AC attribute *CHANGE* of the PIN TEST must reference (via a right) the PIN TEST itself or another PIN TEST 2 (e.g. Super-PIN). For command execution of CHANGE REFERENCE DATA the right in the **AC *CHANGE*** of the referenced PIN TEST is **not relevant in y=0 mode**.

- If the PIN TEST itself is referenced, then Old_Data is verified with the data of the PIN TEST implicitly and - if successful – the right is granted. If the right was granted, CHANGE REFERENCE DATA changes the data of the PIN TEST with New_Data. The CurrentErrorCounter of **PIN TEST** will be used. The UseCounter of the **PIN TEST** will **not** be used. For this case the Old_Data length *m* **need not** to be identical with the New_Data length *n*.
- If the PIN TEST 2 is referenced, then Old_Data is verified with the data of the PIN TEST 2 implicitly and - if successful – the right is granted. If the right was granted, CHANGE REFERENCE DATA changes the data of the PIN TEST with New_Data. The CurrentErrorCounter and the UseCounter of the **PIN TEST 2** will be used. For this case the Old_Data length *m* **need not** to be identical with the New_Data length *n*.
- If the AC *CHANGE* of the PIN TEST is set to always CardOS/M4 searches for the PIN TEST or the PIN TEST 2 via the CSE.

In **CardOS/M4.01** with the CHANGE REFERENCE DATA command a PIN can be changed if old verification data is present or the right **AC CHANGE** referring a LOGICAL TEST (i.e. TEST objects with ALGO=Logical (7Fh)) is granted. CardOS/M4.01 iterates (from first to last byte) through the logical expression data of the LOGICAL TEST to find a valid PIN TEST as follows:

- Operator bytes (0x00 and 0xFF) are skipped.
- The AC number must correspond to a PIN TEST
- The test must be usable (GenerateBit clear, MaxErrorCounter=0 or CurrentErrorCounter>0, USECOUNT>0).

The first matching PIN TEST is taken as reference for the old verification data. The search for this PIN TEST starts in the DF of LOGICAL TEST.

Table 3.5-2 explains the meaning of the ID Byte P2.

Table 3.5-2 ID Byte P2

7	6	5	4	3	2	1	0	¬ **Bit Number / Meaning** ¯
z								z=0: Object search in MF
								z=1: Object search starting in current DF (backtracking)
	v	v	v	v	v	v	v	Object ID ‘*vvv vvvv*’:
								permitted values: ’000 0001’... ’111 1111’

The referenced BS object is searched for starting with the *current DF* and according to the backtracking mechanism of CardOS/M4.

BS objects may consist of object components (see PUT DATA_OCI). If an BS object consists of more than one object component, then for each BS object component a CHANGE REFERENCE DATA command must be used in order to change the object data. In this case all object components must be changed in rising order starting with object component ‘000’ (see table 3.5-1).

If the BS object, whose data are changed using CHANGE REFERENCE DATA, is a TEST object then the following applies:

- The BS object references an access right in the security status.
- The referenced access right in the security status will **not** be withdrawn after completion of the CHANGE REFERENCE DATA command.

Prerequisites

The command can only be executed, if the right referenced by the BS object’s *AC CHANGE* is granted in the *security status* of the *current DF* or in the case of sent Old_Data, if the Old_Data are identical with the TEST object data (see description above).

Unlike the PUT DATA command which can only be performed, if the right referenced by the **current DF’s** *AC UPDATE* is granted in the *security status* of the *current DF*, the CHANGE REFERENCE DATA command can only be performed, if the right referenced by the **BS object’s** *AC CHANGE* is granted in the *security status* of the *current DF*. Because of security reasons it is recommended not to reference the same right for the PUT DATA and for the CHANGE REFERENCE DATA command.

Options

P3=LC, Tr=ON, AC=CHANGE (BS object); SM CHANGE (BS object)

3.6 CREATE FILE

Command CREATE FILE

Function Creates a file (only EF or DF)

Standard / Use ISO 7816-9 / ROM-masked; *ADMINISTRATION*

Parameters

CLA	INS	P1	P2
0xh	E0h	00h	00h

Data

LC	DATA			LE
*	FCI template			./.
	TAG (T)	Length (L)	Value (V)	
	6Fh	*n*	*FCI data*	

FCI data																	
FCI 1			FCI 2			FCI 3			FCI 4			FCI 5			FCI 6 / empty		
T	L	V	T	L	V	T	L	V	T	L	V	T	L	V	T	L	V
For TLV-data of FCIs see table below																	

* Length of subsequent command data field
n Length of subsequent FCI data field

Response ./.

Description The command creates a file below the *current DF*. The FCIs (*File Control Information* according to ISO 7816-4) describe all options of the file to be created. The format of the FCIs is the same as with the SELECT FILE command.

The physical memory required for a file exceeds the net size (EF) / application limit (DF) by 10h bytes. These 16 bytes are needed for the file header. *CYCLIC FIXED* files need one byte extra per record inside the file.

The newly created file will be initialized with 00h bytes. Records will not be created using CREATE FILE.

The FCIs are described in table 3.6-1

Table 3.6-1 Description of FCIs for the CREATE FILE command

T	L	V (Value)	m /o	Description
81h	02h	Net size in bytes (EF) Application Limit (DF, rfu)	m	File Size
82h	03h	**For: BINARY, LINEAR TLV, CODE files and DFs** Byte 1: File Type: 01h: BINARY 05h: LINEAR TLV 21h: CODE 38h: DF Byte 2: Data coding byte 21h, don’t care Byte 3: rfu **For: LINEAR FIXED and CYCLIC FIXED files** Byte 1: File Type: 02h: LINEAR FIXED 06h: CYCLIC FIXED Byte 2: Data coding byte 21h, don't care Byte 3: Record length	m	File Type
83h	02h	File ID (16 bits)	m	File ID
85h	03h	**File status for DF:** Byte 1: 2^7 Deactivation Flag: 1/0: deactivated/activated 2^5 Termination Flag: must be set to ‘0’ 1/0: terminated/usable 2^4 DeathEnableBit: 1/0 permits the switch to life cycle phase DEATH. (*) 2^0 Checksum: don’t care bit Checksum is built over the header always Byte 2: HI byte of DF body size Byte 3: LO byte of DF body size **File status for EF:** Byte 1: 2^7 Deactivation Flag: 1/0: deactivated/activated 2^6 Deactivation Mode: 1: READ can be used for deactivated file 0: READ cannot be used for deactivated file (default) 2^5 Termination Flag: must be set to ‘0’ 1/0: terminated/usable 2^0 Checksum: 1: using header only 0: using header + body (default) Byte 2: rfu Byte 3: rfu	m	File Status
86h	q	Definitions for Access Conditions	m	AC Definitions (see table 3.6-2)
8B	r	Definitions for Secure Messaging If this FCI is not present the **default SM definitions** are used (i.e. SM_{xxx} = FFh, **no SM** for any action)	o	SM Definitions (see table 3.6-3)

m/o:	mandatory/optional
q:	Length of AC definitions in bytes
r:	Length of SM definitions in bytes
(*):	This behavior applies to CardOS/M4.01 always and to CardOS/M4.00, if the so-called UseCounter_Package was loaded and activated before.

The **DeathEnableBit** permits the switch to lifecycle phase DEATH if an object located at this or deeper level tries to perform it due the condition

- USECOUNT reaches zero
- the DeathBit of the object is set

Since this affects the whole card, every DF in the hierarchy and the MF can veto this option by letting this bit cleared. This behavior applies to CardOS/M4.01 always and to CardOS/M4.00, if the so-called UseCounter_Package was loaded and activated before.

The FCIs may be specified in any order inside the FCI data field.

Creating record oriented files the following system limitation must be taken into account:
Without using SM in a record oriented file the *maximum record length* is limited to 255 bytes. Using SM in a record oriented file may reduce the *maximum record length* (i.e. length of the records to be appended, updated or read) according to the algorithms used in the SM definitions. Therefore it makes no sense to create a record oriented file with record lengths exceeding this *maximum record length* as such records cannot be used correctly.

Tables 3.6-2 and 3.6-3 show the possible AC and SM definitions.

Table 3.6-2 AC Definitions in FCI 86h

Byte No.	AC definitions (possible values)	Meaning for DF	Meaning for EF
1	00h ... 7Fh, FFh	AC LCYCLE	AC READ (Data)
2	00h ... 7Fh, FFh	AC UPDATE (Objects)	AC UPDATE (Data)
3	00h ... 7Fh, FFh	AC APPEND (Objects)	AC APPEND (Record) / rfu
4	00h ... 7Fh, FFh	AC DEACTIVATE (DF)	AC DEACTIVATE (EF)
5	00h ... 7Fh, FFh	AC ACTIVATE (DF)	AC ACTIVATE (EF)
6	00h ... 7Fh, FFh	AC DELETE (DF)	AC DELETE (EF)
7	00h ... 7Fh, FFh	AC ADMIN (DF)	AC ADMIN (EF)
8	00h ... 7Fh, FFh	AC CREATE (Files)	AC INCREASE / rfu
9	00h ... 7Fh, FFh	rfu	AC DECREASE / rfu

Meaning of possible values:

00h:	Access action referenced by AC can **always** be performed (i.e. right is granted).
01h ... 7Fh	Access action referenced by AC can be performed, if right 01h ... 7Fh is granted.
FFh:	Access action referenced by AC can **never** be performed.

Table 3.6-3 SM Definitions in FCI 8Bh

Byte No.	SM definitions (possible values)	Meaning for DF	Meaning for EF
1	00h ... FFh	rfu	ENC READ-OUT (Data)
2	00h ... FFh	rfu	SIG READ-OUT (Data)
3	00h ... FFh	ENC UPDATE (0) / ENC UPDATE/APPEND (1)	ENC UPDATE (Data)
4	00h ... FFh	SIG UPDATE (0) / SIG UPDATE/APPEND (1)	SIG UPDATE (Data)
5	00h ... FFh	ENC APPEND (0) / rfu (1)	ENC APPEND (Record) / rfu
6	00h ... FFh	SIG APPEND (0) / rfu (1)	SIG APPEND (Record) / rfu
7	00h ... FFh	rfu	rfu
8	00h ... FFh	rfu	rfu
9	00h ... FFh	rfu	rfu
10	00h ... FFh	rfu	rfu
11	00h ... FFh	ENC DELETE (files)	rfu
12	00h ... FFh	SIG DELETE (files)	rfu
13	00h ... FFh	ENC ADMIN (DF)	ENC ADMIN (EF)
14	00h ... FFh	SIG ADMIN (DF)	SIG ADMIN (EF)
15	00h ... FFh	ENC CREATE (files)	ENC INCREASE-IN (Data) / rfu
16	00h ... FFh	SIG CREATE (files)	SIG INCREASE-IN (Data) / rfu
17	00h ... FFh	rfu	ENC DECREASE-IN (Data) / rfu
18	00h ... FFh	rfu	SIG DECREASE-IN (Data) / rfu
19	00h ... FFh	ENC GENKEY	ENC INCREASE-OUT (Data) / rfu
20	00h ... FFh	SIG GENKEY	SIG INCREASE-OUT (Data) / rfu
21	00h ... FFh	rfu	ENC DECREASE-OUT (Data) / rfu
22	00h ... FFh	rfu	SIG DECREASE-OUT (Data) / rfu
23	00h ... FFh	rfu	ENC READ-IN (DATA) / rfu
24	00h ... FFh	rfu	SIG READ-IN (Data) / rfu

Meaning of possible values:

00h ... FEh	ID of SM-key to be used for referenced access action, ENC/SIG and communication direction.
FFh:	**No** SM.
(0):	CardOS/M4.00 (Objects)
(1):	CardOS/M4.01 (Objects)

Unlike the FCI 86h, which is mandatory, neither the FCI 8Bh nor any of the AC/SM definitions must be specified explicitly.

- If none or not all of the described AC definitions are specified with FCI 86h, then the missing AC definitions are supposed to be specified with **FFh** values **implicitly** (i.e. the access action referenced by the AC can **never** be performed).
- If either FCI 8Bh is missing or none/not all of the described SM definitions are specified with FCI 8Bh, then the missing SM definitions are supposed to be specified with **FFh** values **implicitly** (i.e. **no SM** is specified).

After execution of CREATE FILE the newly created file is selected as *current EF* or *current DF*.

Prerequisites The command can only be executed, if the right referenced by the current DF's *AC CREATE* is granted in the *security status* of the *current DF*.

Options P3=LC, Tr=OFF, AC=CREATE (current DF), SM=CREATE (current DF)

3.7 DEACTIVATE FILE

Command DEACTIVATE FILE

Function Deactivate a file or a file tree

Standard / Use ISO 7816-9 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
00h	04h	00h	00h

Data

LC	Command Data Field	LE
*	empty	./.

* Length of subsequent command data field

Response **./.**

Description This command deactivates the current file inside the filesystem of CardOS/M4. The deactivated file can only be

- selected,
- reactivated or
- deleted. (If the current DF is a deactivated DF neither a child-EF nor a child-DF can be deleted.)

Other actions on the deactivated file cannot be performed. Depending on the bit *Deactivation Mode* in the file status a *READ* action can be performed (see CREATE FILE). The command can be executed even, if the file or DF is already deactivated.

If a DF is deactivated, all child EFs and DFs will not be able to be selected (i.e. deactivation of a complete file path). Table 3.7-1 gives an overview of the available commands in deactivated DFs. In this table commands using structures of the deactivated DF are listed only.

Table 3.7-1 Available CardOS/M4.00 and CardOS/M4.01 commands for deactivated DFs

CardOS/M4.00	CardOS/M4.01
ACTIVATE FILE	ACTIVATE FILE
EXTERNAL AUTHENTICATE	
INTERNAL AUTHENTICATE	
RESET RETRY COUNTER	
SELECT FILE (not for child EFs/DFs)	SELECT FILE (not for child EFs/DFs)
VERIFY	

For security reasons the command set for deactivated DFs was reduced in CardOS/M4.01.

A deactivated file can be reactivated using the ACTIVATE FILE command.

Prerequisites	The command can only be executed, if the right referenced by the file's *AC DEACTIVATE* is granted in the *security status* of the *current DF*. The command cannot be used on *CODE* files.
Options	P3=LC, Tr=ON, AC=DEACTIVATE (file), SM=none

3.8 DECREASE

Command DECREASE

Function Decrease record value

Standard / Use EN 726 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
8xh	30h	00h	EF_ID

Data

LC	Command Data Field	LE
	Sub_Value	
*	HI-LO (m bytes)	**

* Length of subsequent command data field
** Length of requested response data field

Response

Response Data Field	
New_Record_Value	Sub_Value
HI-LO (n bytes)	HI-LO (m bytes)

Description The command decreases the value of the first record of a *CYCLIC FIXED* file by the Sub_Value.

Bits 2^3 to 2^7 of parameter P2 specify the file to be used for the access (i.e. *EF-ID*). Bits 2^0 to 2^2 of parameter P2 are not used (see table 3.8-1)

Table 3.8-1 EF_ID Byte P2

7	6	5	4	3	2	1	0	¬ Bit Number / Meaning ¯
0	*0*	*0*	*0*	*0*				Use current EF
p	*p*	*p*	*p*	*p*				Use SFI *'p pppp' ('p pppp' = '1 1111' not permitted)*
					x	x	x	Bits *xxx* not used (shall be set to 000)

Using a SFI the FID to be used is built adding the constant value **FE00h** to the specified SFI.

The record value is interpreted as unsigned integer in HI-LO format. If an underflow occurs during command execution an error code will be issued. The length *m* of the Sub_Value must be smaller than or equal to the New_Record_Value length *n*. If the command is executed using SM then *n+m* must not exceed certain limits determined by the SM mode and SM algorithms used.

After successful command execution the result of the subtraction will be stored in the oldest record of the file or in an additional record which will be created automatically (depending on free memory/record(s) in the file). This record will become the *current record* and the *logical first record*.

Prerequisites The command can only be executed, if the right referenced by the file's *AC DECREASE* is granted in the *security status* of the *current DF*.

Options P3=LC, Tr=ON, AC=DECREASE (file), SM=DECREASE-IN (file) and/or SM=DECREASE-OUT (file)

3.9 DELETE FILE

Command DELETE FILE

Function Deletes a file

Standard / Use ISO 7816-9 / CardOS/M4.00: Delete_File_Package, CardOS/M4.01: ROM-masked; *ADMINISTRATION*

Parameters

CLA	INS	P1	P2
0xh	E4h	00h	00h

Data

LC	Command Data Field	LE
	File_ID	
*	2 bytes	./.

* Length of subsequent command data field

Response ./.

Description DELETE FILE deletes the file referenced by the File_ID. The file to be deleted must be a child-EF or child-DF of the *current DF*.

A DF can only be deleted, if it does not contain other files (i.e. file trees cannot be deleted using a single DELETE FILE command but as many DELETE FILE commands as the number of involved files).

Deactivated files can be deleted, if none of the parent DFs starting from the file to be deleted up to the MF is **not deactivated nor** the **checksum** of any parent DF **is corrupted**.

The MF can never be deleted using DELETE FILE (for deletion of the MF and the complete filesystem the command ERASE FILES must be used).

The free memory of the deleted file will **be initialized** with FFh and the freed memory will **not be defragmented**.

If the checksum of the file to be deleted is wrong (and therefore the header of the file may be corrupt) the file can be deleted using DELETE FILE even if the necessary right referenced by the *AC DELETE* is **not** granted. Note in this context that a DF to be deleted must not contain other files.

Packages cannot be deleted using DELETE FILE. Packages can be removed using the UNINSTALL PACKAGE command.

Note

In combination with the MANAGE CHANNEL command and during execution the DELETE FILE command

- **must be performed in logical channel 00h** and
- **other logical channels must not be opened**.

This usage of the DELETE FILE command in combination with the MANAGE CHANNEL command

- is mandatory for CardOS/M4.01 and
- may otherwise lead to inconsistent data in a CardOS/M4.00 smart card. This problem for CardOS/M4.00 can be solved, if other security mechanisms of CardOS/M4 are used to prohibit other usages of DELETE FILE in combination with MANAGE CHANNEL as described above.

Prerequisites

The command is available for CardOS/M4.01 always and for CardOS/M4.00 only, if the Delete_File_Package was loaded and activated before. Except the file to be deleted has a wrong checksum (see description above) the command can only be executed, if the right referenced by the file's *AC DELETE* is granted in the *security status* of the *current DF*.

Options

P3=LC, Tr=OFF, AC=DELETE (file), SM=DELETE (current DF)

3.10 DIRECTORY

Command	DIRECTORY
Function	Read file information of EFs and/or DFs in the current DF
Standard / Use	PCSC ICC Service Provider Part 6 / CardOS/M4.00: PCSC_Package, CardOS/M4.01: ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
80h	16h	SEL	OFFSET

Data

LC	Command Data Field	LE
*	empty	**

* Length of subsequent command data field
** Length of requested response data field

Response

Response Data Field									
Information_String									
BER-TLV1			BER-TLV2			...	BER-TLVn		
T	L	V	T	L	V	...	T	L	V
6Fh	xxh		6Fh	xxh		...	6Fh	xxh	

V Each value field of a BER-TLV construct can contain between 1 and 4 TLV constructs, which may be concatenated in any order.
L Length of belonging value field V

Description

DIRECTORY returns information about files (except CODE files) inside the current DF. The SEL byte P1 specifies, whether information about EFs and/or DFs shall be returned (see table 3.10-1)

Table 3.10-1 Usage of SEL byte P1

P1	Meaning
00h	Return information about **DFs** located inside the current DF.
01h	Return information about **EFs** located inside the current DF.
02h	Return information about **DFs** and **EFs** located inside the current DF.

For each file found in accordance to P1 one BER-TLV construct is returned. Each BER-TLV construct contains in its value field TLV constructs describing the belonging file. These TLV constructs (see table 3.10-2) may be concatenated in any order.

Table 3.10-2 Possible TLV constructs for DFs and/or EFs

T (TAG)	L (Length)	V (Value)
82h	01h	File Type
86h	02h	File Identifier (FID)
8Ah	01h	Next_Offset for next call of DIRECTORY command

The OFFSET byte P2 (P2 = 00h ... m) specifies, how many (m) BER-TLV constructs (of all potential ones), which are found according to P1, shall be skipped before CardOS/M4 shall start with the Information_String. Normally the first time DIRECTORY is executed P2=00h.

If the length of the Information_String exceeds 255 bytes (i.e. the I/O buffer size), then the Information_String will be cut. Then the cut Information_String contains complete BER-TLV constructs. The last BER_TLV construct of the cut Information_String has TAG = 8Ah and contains the value Next_Offset (see table 3.10-2).

In order to get the next part of the remaining information for the following DIRECTORY command OFFSET = Next_Offset of the previous DIRECTORY command shall be specified. The P1 byte must not be changed for this following DIRECTORY command.

If no information according to P1 and P2 is available, then no response data are returned.

Information about CODE files cannot be read using DIRECTORY.

Prerequisites The command is available for CardOS/M4.01 always and for CardOS/M4.00 only, if the PCSC_Package was loaded and activated before.

Options P3=LC, Tr=OFF, AC=none, SM=none

3.11 ERASE FILES

Command ERASE FILES

Function Erase file system in EEPROM.

Standard / Use Proprietary / ROM-masked; *ADMINISTRATION, INITIALIZATION, PERSONALIZATION*

Parameters

CLA	INS	P1	P2
84h	06h	00h	00h

Data

LC	Command Data Field		LE
	Enciphered(MAC,	Padding$_{ENC}$)	
10h	8 bytes	8 bytes	./.

Response ./.

Description The command erases the following areas:

- the filesystem
- the system or application package(s) if they exist,
- the application descriptors if they exist.

The filesystem on the chip needs to be initialized again to make it usable. When the command finishes normally, the card changes to the life cycle phase *MANUFACTURING*.

Note: Granted access rights will **not** be withdrawn during or after command execution of ERASE FILES. Therefore and for security reasons it is strongly recommended to reset the CardOS/M4.0 smart card after an ERASE FILES command.

Note In combination with the MANAGE CHANNEL command and during execution the ERASE FILES command

- **must be performed in logical channel 00h** and
- **other logical channels must not be opened**.

This usage of the ERASE FILES command in combination with the MANAGE CHANNEL command

- is mandatory for CardOS/M4.01 and
- may otherwise lead to inconsistent data in a CardOS/M4.00 smart card. This problem for CardOS/M4.00 can be solved, if other security mechanisms of CardOS/M4 are used to prohibit other usages of ERASE FILES in combination with MANAGE CHANNEL as described above.

Erase in Progress CardOS/M3 or higher provides the so-called life cycle phase “ERASE IN PROGRESS (29h)”. This phase is passed usually during the run of the ERASE FILES command. If the GET DATA command returns this value it is therefore recommended to run the ERASE FILES command once again to complete the process.

Prerequisites The command can only be executed in the life cycle phases *INITIALIZATION*, *PERSONALIZATION* and *ADMINISTRATION*.
The command can only be executed using SM mode x4h with the *StartKey*.

Options For CardOS/M4.00:
P3=LC, Tr=OFF, AC=none, SM=SM mode x4h with *StartKey*

For CardOS/M4.01 in *INITIALIZATION* and *PERSONALIZATION* life cycle phases:
P3=LC, Tr=OFF, AC=none, SM=SM mode x4h with *StartKey*

For CardOS/M4.01 in *ADMINISTRATION* life cycle phase:
P3=LC, Tr=OFF, AC= ERASE(MF), SM=SM mode x4h with *StartKey*

3.12 EXTERNAL AUTHENTICATE

Command EXTERNAL AUTHENTICATE

Function Perform a challenge/response test

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh	82h	00h	ID

Data

LC	Command Data Field	LE
	C/R_Test_Response	
*	(var. length)	./.

* Length of subsequent command data field

Response **./.**

Description The command performs a challenge / response test with a symmetric or an asymmetric key.

EXTERNAL AUTHENTICATE can only be used either with a

- **TEST-MAC_Key**
  (i.e. TEST object using tt='01' or tt='11' in the OPTIONS byte **and** using ALGO:
  - MAC (ALGO=80h) or
  - iMAC (ALGO=81h) or
  - MAC3 (ALGO=82h) or
  - iMAC3 (ALGO=83h))

or a

- **TEST-Public_Key**
  (i.e. TEST object using tt='01' in the OPTIONS byte and using ALGO:
  - RSA_SIG (ALGO=88h) or
  - DSA (ALGO=89h) (*)
  - RSA_PURE_SIG (ALGO=8Ch) or
  - RSA_SIG_SHA-1 (ALGO=C8h) or
  - DSA_SHA-1 (ALGO=C9h) (*) or
  - RSA_PURE_SIG_SHA-1 (ALGO=CCh)).

(*) DSA algorithms are available only, if DSA_Package was loaded and activated before

Table 3.12-1 shows the ID Byte P2

Table 3.12-1 ID Byte P2

7	6	5	4	3	2	1	0	¬ **Bit Number / Meaning** ¯
z								z=0: Object search in MF
								z=1: Object search starting in current DF (backtracking)
	v	v	v	v	v	v	v	Object ID ‘*vvv vvvv*’:
								permitted values: ’000 0001’... ’111 1111’
0	0	0	0	0	0	0	0	Object referenced in CSE

The range of the object IDs is limited because the belonging TEST object is always referencing exactly one access right in the *security status* of the belonging DF (see PUT DATA_OCI command also). Each DF supports the access rights 01h ... 7Fh (i.e. ‘0000 0001’ ... ‘0111 1111’). Other values for ‘vvv vvvv’ as specified above lead to errors.

If P2=00h, then CardOS/M4 “knows” which object shall be used. For this the TEST object ID referenced in the CSE is used (TEST component is byte 5 of the CSE). The backtracking mechanism for searching the TEST object ID starts from the *current DF*.

Before EXTERNAL AUTHENTICATE can be used a random number (i.e. the challenge) must be requested from the smart card using the GET CHALLENGE command. A requested challenge can be used once only and need not to be requested immediately before the EXTERNAL AUTHENTICATE command but during the same smart card session.

- The minimum length of the challenge to be requested from the smart card is specified in the *MINLEN* byte of the TEST object (see PUT DATA_OCI).
- The C/R_Test_Response must be built using the complete challenge (i.e. using all bytes of the challenge incl. padding bytes). If the length of the challenge requested is less than the length specified in *MINLEN* then an error code is issued during command execution of EXTERNAL AUTHENTICATE.

The C/R_Test_Response is verified differently for TEST-MAC_Keys (i.e. symmetric keys) and for TEST-Public_Keys (i.e. asymmetric keys).

- For symmetric keys the C/R_Test_Response is verified with a response internally calculated for the challenge sent with the GET CHALLENGE command before.
- For asymmetric keys first the challenge is internally calculated using the C/R_Test_Response and the Public_Key. Then the internally calculated challenge and the challenge sent with the GET CHALLENGE command are verified.

If the C/R_Test_Response is checked successfully by EXTERNAL AUTHENTICATE the referenced access right is granted, the CurrentErrorCounter in the *ERRCNT* byte of the TEST object is set to its maximum (i.e. MaxErrorCounter) (if defined so in the TEST object; see PUT DATA_OCI).

If the C/R_Test_Response is **not** checked successfully the referenced access right will be withdrawn (depending on the *FAULT* bit 2^7 in the *OPTIONS* byte of the TEST object; see PUT DATA_OCI); in addition the CurrentErrorCounter of the TEST object is decreased

Depending on the definition in the TEST object the UseCounter in the TEST object is decreased (see PUT DATA_OCI).

The access right will be granted or withdrawn in the DF, where the TEST object is stored, and in all child-DFs.

Prerequisites The command can only be executed, if a TEST object (i.e. the key usage bits *'uuu'* of the object are *'000'*) with the object ID specified in P2 exists (according to the backtracking mechanism of CardOS/M4) **and** the TEST object is **not blocked** (see UseCounter and CurrentErrorCounter of objects) **and** the right referenced by the TEST object's *AC USE* is granted in the *security status* of the *current DF*.

Specials According to ISO 7816-4 the bits 2^5 ... 2^6 of byte P2 (ID) shall be set to '00'. For future compatibility it is recommended to take this restriction into account when defining object IDs.

CardOS/M4 supports **only one** storage for an internal random number. Since EXTERNAL AUTHENTICATE needs an internal random number for execution, an eventual SM for EXTERNAL AUTHENTICATE **cannot use command-SM with** a – in this case – second **internal random number**.

Options P3=LC, Tr=ON, AC=USE (TEST object referenced by object ID),
SM=USE-IN (TEST object referenced by object ID)

3.13 FORMAT

Command FORMAT

Function Change life cycle phase from *MANUFACTURING* to *INITIALIZATION* or to *ADMINISTRATION*.

Standard / Use Proprietary / ROM-masked; *MANUFACTURING*

Parameters

CLA	INS	P1	P2
84h	40h	00h	PHASE

Data

LC	DATA			LE
	Enciphered (MAC, Padding$_{ENC}$) / Enciphered (FCI template, MAC, Padding$_{ENC}$)			
*	(8 + *p*) bytes / (*n* + 10 + *p*) bytes			./.
	Empty / FCI Template	**MAC**	**Padding$_{ENC}$**	
	0 bytes / *n*+2 bytes	8 bytes	*p* bytes	

FCI Template		
T (TAG)	**L (Length)**	**V (Value)**
6Fh	*n*	*FCI data*

FCI data								
FCI 1			**FCI 2**			**FCI 3 / empty**		
T	L	V	T	L	V	T	L	V

*	Length of subsequent command data field
n	Length of subsequent FCI data field

Response ./.

Description Depending on byte P2 the command changes the life cycle phase from *MANUFACTURING* to *ADMINISTRATION* (i.e. logical personalization) or to *INITIALIZATION* (i.e. physical personalization) (see table 3.13-1).

Table 3.13-1 Usage of PHASE byte P2

P2 (PHASE)	Meaning
00h	Change from *MANUFACTURING* to *INITIALIZATION*
01h	Change from *MANUFACTURING* to *ADMINISTRATION* and creation of the MF

If P2 = 00h, then the FCI template is empty.

If P2 = 01h, then the FCI template must be specified and the FCIs (*File Control Information* according to ISO 7816-4) describe all options of the MF to be created. **After** execution of FORMAT the MF is selected as *current DF*.

The FCIs are described in table 3.13-2. The FCIs may be specified in any order inside the FCI data field.

Table 3.13-2 Description of FCIs for the FORMAT command

T	L	V (Value)	m /o	Description
85h	03h	Byte 1: 2^7 Invalidation flag: 1/0: invalid/valid 2^4 DeathEnableBit: 1/0 permits the switch to life cycle phase DEATH. (*) 2^0 Checksum: don't care bit Checksum is built over the header always Byte 2: HI byte of MF body size Byte 3: LO byte of MF body size	m	File Status
86h	q	Definitions for Access Conditions	m	AC Definitions (see table 3.13-3)
8Bh	r	Definitions for Secure Messaging If this FCI is not present the **default SM definitions** are used (i.e. SM_{xxx} = FFh, **no SM** for any action)	o	SM Definitions (see table 3.13-4)

m/o:	mandatory / optional
q:	Length of AC definitions in bytes
r:	Length of SM definitions in bytes
(*):	This behavior applies to CardOS/M4.01 always and to CardOS/M4.00, if the so-called UseCounter_Package was loaded and activated before.

The **DeathEnableBit** permits the switch to lifecycle phase DEATH if an object located at this or deeper level tries to perform it due the condition

- USECOUNT reaches zero
- the DeathBit of the object is set

Since this affects the whole card, every DF in the hierarchy and the MF can veto this option by letting this bit cleared. This behavior applies to CardOS/M4.01 always and to CardOS/M4.00, if the so-called UseCounter_Package was loaded and activated before.

Tables 3.13-3 and 3.13-4 show the possible AC and SM definitions.

Table 3.13-3 AC definitions in FCI 86h

Byte No.	AC definitions (possible values)	Meaning for MF
1	00h ... 7Fh, FFh	AC LCYCLE
2	00h ... 7Fh, FFh	AC UPDATE (Objects)
3	00h ... 7Fh, FFh	AC APPEND (Objects)
4	00h ... 7Fh, FFh	AC DEACTIVATE (MF)
5	00h ... 7Fh, FFh	AC ACTIVATE (MF)
6	00h ... 7Fh, FFh	AC ERASE (MF) (1); shall be set to 00h (0)
7	00h ... 7Fh, FFh	AC ADMIN (MF)
8	00h ... 7Fh, FFh	AC CREATE (Files)
9	00h ... 7Fh, FFh	AC EXECUTE

(0) Applies to CardOS/M4.00
(1) Applies to CardOS/M4.01

Meaning of possible values:

00h:	Access action referenced by AC can **always** be performed (i.e. right is granted).
01h ... 7Fh	Access action referenced by AC can be performed, if right 01h ... 7Fh is granted.
FFh:	Access action referenced by AC can **never** be performed.

Unlike the FCI 86h which is mandatory in the FCI template none of the AC definitions must be specified explicitly.

- If none/not all of the described AC definitions are specified with FCI 86h, then the missing AC definitions are supposed to be specified with **FFh** values **implicitly** (i.e. the access action referenced by the AC can **never** be performed). If all access actions on the MF are blocked, then this leads to an **unusable** smart card **irreversibly**. Therefore, using the FORMAT command it is strongly recommended to **specify all AC definitions for** the **MF.**

Table 3.13-4 SM definitions in FCI 8Bh

Byte No.	SM definitions (possible values)	Meaning for MF
1	00h ... FFh	rfu
2	00h ... FFh	rfu
3	00h ... FFh	ENC UPDATE (Objects) (0) / ENC UPDATE/APPEND (Objects) (1)
4	00h ... FFh	SIG UPDATE (Objects) (0) / SIG UPDATE/APPEND (Objects) (1)
5	00h ... FFh	ENC APPEND (Objects) (0) / rfu (1)
6	00h ... FFh	SIG APPEND (Objects) (0) / rfu (1)
7	00h ... FFh	rfu
8	00h ... FFh	rfu
9	00h ... FFh	rfu
10	00h ... FFh	rfu
11	00h ... FFh	rfu
12	00h ... FFh	rfu
13	00h ... FFh	ENC ADMIN (MF)
14	00h ... FFh	SIG ADMIN (MF)
15	00h ... FFh	ENC CREATE (Files)
16	00h ... FFh	SIG CREATE (Files)
17	00h ... FFh	rfu
18	00h ... FFh	rfu
19	00h ... FFh	ENC GENKEY
20	00h ... FFh	SIG GENKEY
21	00h ... FFh	rfu
22	00h ... FFh	rfu
23	00h ... FFh	rfu
24	00h ... FFh	rfu

Meaning of possible values:

00h ... FEh	ID of SM-key to be used for referenced access action, ENC/SIG and communication direction.
FFh:	**No** SM.
(0):	CardOS/M4.00
(1):	CardOS/M4.01

Neither the FCI 8Bh nor any of the SM definitions must be specified explicitly.

- If either FCI 8Bh is missing or none/not all of the described SM definitions are specified with FCI 8Bh, then the missing SM definitions are supposed to be specified with **FFh** values **implicitly** (i.e. **no SM** is specified).

Prerequisites This command can only be executed using SM mode x4h with the *StartKey*.

Options P3=LC, Tr=OFF, AC=none, SM=SM mode x4h with the *StartKey*

3.14 GENERATE KEY PAIR

Command GENERATE KEY PAIR

Function Generate a key pair for the RSA algorithm within the card

Standard / Use ISO FDIS7816-8 / CardOS/M4.00: Generate_Key_Pair_Package, CardOS/M4.01: ROM-masked; *ADMINISTRATION*

Parameters

CLA	INS	P1	P2
0xh	46h	00h (ignored)	00h (ignored)

Data

LC	Command Data Field					LE
	Obj_Add	File_ID	Add_RMT	Diff_pq	L_PubEx	
*	2 bytes	2 bytes	1 byte	1 byte	2 bytes	./.

* Length of subsequent command data field

Response **./.**

Description The command creates a public key pair for RSA algorithms on the CardOS/M4 smart card.

For this the private key components are stored **unreadable** in an existing key object of the current DF. The public key components are appended to a **readable** and existing LINEAR TLV EF of the current DF.

Table 3.14-1 Meaning of the different parts of the command data field

Part	mandatory/ optional (m/o)	Meaning
Obj_Add	m	The private RSA key components to be generated are referenced by Obj_Add.
File_ID	m	The public RSA key components to be generated will be stored in the LINEAR TLV EF referenced by File_ID.
Add_RMT	o	Specifies the number of **additional** Rabin-Miller tests to be performed.
Diff_pq	o	Specifies the difference in length of p and q in **bits**.
L_PubEx	o	Specifies the length of the public exponent in **bits**.

Only the first two items of the command data field are mandatory, all following items are optional. In BNF the structure of data field is

<Obj_Add> <File_ID> [<Add_RMT>[<Diff_PQ>[<L_PubEx>]]].

The private RSA key objects referenced by Obj_Add have to be created before via the PUT DATA. There must exist two components of sufficient size.

- It is advised to create both components of the private key with the same size, since a short private exponent may compromise security. Especially if the size of the public exponent is specified, the full modulus size may be required.

- All header information of the objects in the tags 83h, 85h, 86h and possibly 8Bh has to be specified as desired for the private RSA key to be generated. They are NOT modified by the command GENERATE KEY PAIR. The data part, i.e. the private key data contained in the tag 8Fh should be filled with the desired number of **FFh** bytes. These data are overwritten by the generated private key data. (One consequence is, that the size of the modulus N can only be specified in multiples of 8 bits)

- The right referenced by each private key object's AC GENKEY, which is specified in TAG 86h of each key object must be granted and the ALGO-ID must be one of the RSA family.

Obj_Add (mandatory)

The private key object is referenced using the two bytes Obj_Add in the data field. The first byte contains the object class, where the bits $2^0..2^2$ *must be zero*, the second byte contains the object ID. The private key objects have to reside in the current DF.

File_ID (mandatory)

The specified EF (of LINEAR TLV type) has to reside in the current DF but may be empty. Inside this EF the public key components (i.e. modulus N and exponent E) are stored by the command GENERATE KEY PAIR. In order store the key components each time GENERATE KEY PAIR is performed successfully two TLV structures are **appended** to the EF.

- one record with TAG **10h** for modulus N and
- one record with TAG **11h** for Exponent E.

Modulus N and the exponent E are stored in CCMS format. As a consequence of CCMS the data part consists of an additional length byte, a zero byte for the number of trailing zeroes and the key data.

As two TLV structures are appended to the EF the right referenced by the EF's AC APPEND has to be granted for command execution.

Add_RMT (optional)

For computation of each prime number p and q first a random number is generated using the hardware generator. The random number is checked for small prime factors first and subsequently a number of iterations of the probabilistic Rabin-Miller-Test are performed. The default is five successful iterations, but this can be increased by specifying a positive byte number increment using Add_RMT (allowed 0 – 13 (0x0D)).

Diff_pq (optional)

The basis for the length of the prime factors is the half size of the modulus N. Since the same size of factors is a potential security problem, the length p is enlarged thereby shortening q. The default is 16 bits difference but can be overridden specifying Diff_pq (allowed 0 – ¼ of length of modulus(bits)). (The three least significant bits are ignored, since only byte granularity is available.)

L_PubEx (optional)
The default for the public exponent length is 160 bits, which speeds up encryption. It may be overridden specifying L_PubEx (allowed values *16 – 1024 bits*; these values are **rounded up** to a multiple of 8 by CardOS/M4.).

Note
The GENERATE KEY PAIR command can only be executed, if the EnableBit and the GenerateBit of the private RSA key are set correctly. For these bits the following applies:

- If the EnableBit and GenerateBit are set, then the BS object data can be overwritten **once** using the GENERATE KEY PAIR command.
- If the EnableBit is set and GenerateBit is zero, then the BS object data cannot be overwritten using the GENERATE KEY PAIR command.
- If the EnableBit is zero, then the BS object data can always be overwritten using the GENERATE KEY PAIR command.
- If the GenerateBit is set, only a generation of a key pair is possible. After the generation, the GenerateBit is reset implicitly by the command.

EnableBit and GenerateBit are located in TAG 85h (object parameters) of the PUT DATA_OCI command.

Prerequisites
The command can only be executed, if the right referenced by the private RSA key object's *AC GENKEY* and the right referenced by the EF's *AC APPEND* is granted in the *security status* of the *current DF*.

Additionally
- the EnableBit n in the OPTIONS byte and
- the GenerateBit g in the FLAGS byte

of the private RSA key must be set correctly. For these bits the following applies:

- If n=1 and g=1, then the BS object data can be overwritten **once** using the GENERATE KEY PAIR command.
- If n=1 and g=0, then the BS object data **cannot** be overwritten using the GENERATE KEY PAIR command.
- If n=0 and g=1 or g=0, then the BS object data can **always** be overwritten using the GENERATE KEY PAIR command. In order to limit overwriting of the BS object data other standardized mechanisms of CardOS/M4 (e.g. AC GENKEY of the BS object) must be used.

The command requires XRAM temporarily and is only available in the *ADMINISTRATION* life cycle phase.

For CardOS/M4.01 the GENERATE KEY PAIR command is available always whereas for CardOS/M4.00 the Generate_Key_Pair_Package must have been loaded and activated before execution.

Options
P3=LC, Tr=ON, AC=GENKEY (private RSA key objects) **AND** AC=APPEND (EF for storage of the public RSA key components) **AND** settings of EnableBit n (of the public RSA key components) **AND** GenerateBit g (of the public RSA key components) accordingly, SM=GENKEY (current DF)

3.15 GET CHALLENGE

Command	GET CHALLENGE
Function	Generate a random number
Standard / Use	ISO 7816-4 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
00h	84h	00h	00h

Data

LC	Command Data Field	LE
*	empty	**

* Length of subsequent command data field
** Length of requested response data field (i.e. length of random number)

Response

Response Data Field
RAND
n bytes

NOTE: The **maximum length** of the random number is **248 bytes** (**F8h bytes**).

Description

This command generates the internal random number RAND, which can be used by the EXTERNAL AUTHENTICATE command or by **command-SM**. The length *n* of the random number to be returned is specified by the LE parameter of the command. Valid values for *n* are in the range **01h ... F8h**.

If LE=00h or LE≥F9h then *n*=F8h bytes will be returned. For all other LE values *n*=LE bytes will be returned.

The internal random number RAND can be used only once. For RAND one temporarily storage is reserved. This means that the internal random number is valid during the smart card session until it is used by EXTERNAL AUTHENTICATE, command-SM or until it is overwritten by a new GET CHALLENGE command.

The internal random number RAND is derived from a real hardware random number generator. The internal random number RAND uses the XRAM area of the smart card.

Prerequisites ./.

Options P3=LE, Tr=ON, AC=none, SM=none

3.16 GET DATA

Command GET DATA

Function Read system information

Standard / Use ISO 7816-4 / ROM-masked; *MANUFACTURING, INITIALIZATION, PERSONALIZATION, ADMINISTRATION, OPERATIONAL, DEATH*

Parameters

CLA	INS	P1	P2
00h	CAh	01h	MODE

Data

LC	Command Data Field	LE
*	empty	**

* Length of subsequent command data field
** Length of requested response data field

Response

Response Data Field
System_Data
(format of the object data see table 3.16-1)

Description The GET DATA command supplies information on the current status of the system. MODE specifies the information to be returned (see table 3.16-1).

Table 3.16-1 MODE Byte P2

MODE	Results	Format of System_Data
80h	Product name, version, release date, copyright string	ASCII string (incl. terminating 00h)
81h	Chip production data as supplied by Infineon, PROM area	32 bytes (Byte 1 .. 32) Byte 9: Chip Type Byte 11 .. 16 Unique Chip Identification Number
82h	OS Version in ATR coding	Byte 1: OS Class (1st historical byte in ROM-ATR) Byte 2: OS Version (2nd historical byte in ROM-ATR) For CardOS/M4.00: C8h 02h For CardOS/M4.01: C8h 03h
83h	Current life cycle phase of *current DF*	1 byte
84h	current security status of *current DF*	For each granted access right the number of the access right is returned (e.g. 03h 01h means that access rights 3 and 1 in the current security status are granted.)
85h	free memory within body of *current DF*	16 bit Integer (Hi-Lo)
86h	ATR status (ROM-/EEPROM-ATR)	1 byte (00h: ROM-ATR, 80h: EEPROM-ATR)
87h	Absolute path to the current DF (used for PC/SC GetCurrentDir)	Concatenation of m FIDs starting from the MF (FID of MF not included!) For CardOS/M4.00: Available only if PCSC_Package is loaded and activated.
88h	Packages	ID of loaded packages (every TLV record denoting one package using TAG as manufacturer ID) TAGs: 01h..3Fh used for Siemens packages (see UNINSTALL PACKAGE command)
89h	Hardware and memory characteristics	Byte 1 .. 2: XRAM size Byte 3 .. 4: EEPROM size Byte 5: CPU type (44 or 66) Byte 6: Chip configuration (corresponds to contents of CONFIGP register)
8Ah	EEPROM memory (size of largest free EEPROM block)	16 bit Integer (Hi-Lo)
96h	Retry Counter of common system keys	Byte 1: Factory Version of PackageLoadKey Byte 2: Retry Counter of PackageLoadKey Byte 3: Factory Version of StartKey Byte 4: Retry Counter of StartKey
98h	Application Descriptor(s)	n * 4 bytes for n Application Descriptors, which are currently stored in the smart card (Example for 2 Application Descriptors: Byte 1 Personalization Number for Application Descriptor 1 Byte 2 .. 3 Sequence Number for Application Descriptor 1 Byte 4 Error Counter for PersonalizationKey 1 Byte 5 Personalization Number for Application Descriptor 2 Byte 6 .. 7 Sequence Number for Application Descriptor 2 Byte 8 Error Counter for PersonalizationKey 2)

Erase in Progress	CardOS/M3 or higher provides the so-called life cycle phase “ERASE IN PROGRESS (29h)”. If the GET DATA command returns this value, it is recommended to run the ERASE FILES command once again, as this phase is usually passed during the run of the ERASE FILES command..
Prerequisites	**./.**
Options	P3=LE, Tr=ON, AC=none, SM=none

3.17 GIVE RANDOM

Command GIVE RANDOM

Function Transmit an external random number to the smart card

Standard / Use EN 726 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
80h	86h	00h	00h

Data

LC	Command Data Field	LE
	Random_Number	
*	(n bytes)	./.

* Length of subsequent command data field

Response ./.

Description The command transmits an external Random_Number of *n* bytes, which is used for the next **response-SM** calculation. Valid values for *n* are in the range **01h ... F8h**.

The external random number can be used only once. For the external random number one temporarily storage is reserved. This means that the external random number is valid during the smart card session until it is used by the response-SM or until it is overwritten by a new GIVE RANDOM command.

For storage of the external random number CardOS/M4 uses the XRAM of the smart card.

Prerequisites ./.

Options P3=LC, Tr=ON, AC=none, SM=none

3.18 INCREASE

Command INCREASE

Function Increase record value

Standard / Use EN 726 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
8xh	32h	00h	EF_ID

Data

LC	Command Data Field	LE
	Add_Value	
*	HI-LO (m bytes)	**

* Length of subsequent command data field
** Length of requested response data field

Response

Response Data Field	
New_Record_Value	Add_Value
HI-LO (n bytes)	HI-LO (m bytes)

Description The command increases the value of the current record of a *CYCLIC FIXED* file by the Add_Value.

Bits 2^3 to 2^7 of parameter P2 specify the file to be used for the command (i.e. *EF_ID*). Bits 2^0 to 2^2 of parameter P2 are not used (see table 3.18-1)

Table 3.18-1 EF_ID Byte P2

7	6	5	4	3	2	1	0	¬ **Bit Number / Meaning** ¯
0	*0*	*0*	*0*	*0*				Use current EF
p	*p*	*p*	*p*	*p*				Use SFI *'p pppp' ('p pppp' = '1 1111' not permitted)*
					x	x	x	Bits *xxx* not used (shall be set to 000)

Using a SFI the FID to be used is built adding the constant value **FE00h** to the specified SFI.

The record value is interpreted as unsigned integer in HI-LO format. If an overflow occurs during command execution an error code will be issued.

After successful command execution the result of the addition will be stored in the oldest record of the file or in an additional record which will be created automatically (depending on free memory/record(s) in the file). This record will become the *current record* and the *logical first record*.

The length *m* of the Add_Value must be smaller than or equal to the record length *n*. If the command is executed using SM then *n+m* must not exceed certain limits determined by the SM mode and SM algorithms used.

Prerequisites The command can only be executed, if the right referenced by the file’s *AC INCREASE* is granted in the *security status* of the *current DF*.

Options P3=LC, Tr=ON, AC=INCREASE (file), SM=INCREASE-IN (file) and/or SM=INCREASE-OUT (file)

3.19 INITIALIZE EEPROM

Command INITIALIZE EEPROM

Function Manufacturer use only

Standard / Use Proprietary/ ROM-masked; *INITIALIZATION*

Parameters

CLA	INS	P1	P2
80h / 84h	02h	MODE	00h / BLOCK-ID

Data

LC	Command Data Field	LE
*	(var. length)	./.

* Length of subsequent command data field

Response **./.**

Description

Using the INITIALIZE EEPROM command in life cycle phase *INITIALIZATION* it is possible to load the following data to the EEPROM area of the smart card:

- **filesystem structures**,
- **data structures**,
- **application descriptors,**
- **placeholders** and
- eventually **file data**.

The INITIALIZE EEPROM command can be executed with or without secure messaging (SM) using the *StartKey*. Using parameter MODE (P1) SM can be specified according to table 3.19-1, if CLA = 84h. Note that this is an extension to the standardized SM mode x4h of CardOS/M4.

Table 3.19-1 Usage of MODE (P1), if CLA = 84h

7	6	5	4	3	2	1	0	¬ **Bit Number / Meaning** ¯	
s								s=0/1:	SM without/with SIGning
	m							m=0/1:	SM without/with ENCryption
		p						p=1:	Alloc_P_Bit: Allocate Memory for Package Data
			t					t=1:	Alloc_Bit: Allocate Memory for Normal Data
				z	z	z	z		*zzzz* bits are rfu

If CLA = 80h, then P2 must be set to 00h.

INITIALIZE EEPROM supports the following 5 functionalities, which are described in table 3.19-2:

- **Allocate_Memory_Area**,
- **Write_Block,**
- **Finalize_Memory_Area**,
- **Allocate_&_Write,** and
- **Allocate_&_Write_&_Finalize.**

Table 3.19-2 Functionalities of INITIALIZE EEPROM with necessary parameter settings

Functionality	Alloc_Bit or Alloc_P_Bit exclusively set in P1	P2	LC	Command Data Field			Comment
Allocate_Memory_Area	yes	00h	02h	**Size**			Allocates a memory area in the EEPROM of the smart card:
				HI Byte	LO Byte		
				1 byte	1 byte		Size of memory area to be allocated in bytes
Write_Block	no	00h	n	**Offset**		**Data**	Writes a block of a allocated memory area:
				HI Byte	LO Byte	Block_Data	
				1 byte	1 byte	(n-2) bytes	Offset for the Block_Data starting from beginning of the allocated memory area is specified in bytes. Block_Data are the whole content of the allocated memory area or a part of it.
Finalize_Memory_Area	no	00h / FTAG		**Final_Size**			Finalizes a memory area: If P2≠00h, then P2 specifies the final TAG (FTAG) of the memory area to be finalized.
			00h	empty			If LC = 00h, then no Final_Size of the memory area will **not** be modified.
			02h	HI Byte	LO Byte		If LC = 02h, then the Final_Size of the memory area will be modified.
				1 byte	1 byte		
Allocate_&_Write	yes	00h	n	**Size**		**Data**	Allocates and writes a part or the whole memory area:
				HI Byte	LO Byte	Block_Data	
				1 byte	1 byte	(n-2) bytes	Size of memory area to be allocated in bytes. Block_Data are the whole content of the allocated memory area or a part of it.
Allocate_&_Write_&_Finalize	yes	FTAG	n	**Final_Size**		**Data**	Allocates, writes and finalizes a memory area:
				HI Byte	LO Byte	Block_Data	
				1 byte	1 byte	(n-2) bytes	FTAG specifies the final TAG of the memory area to be finalized. Final_Size specifies the size of the memory area to be finalized. Block_Data are the whole content of the memory area to be finalized.

During allocation of the memory area (e.g. Allocate_Memory_Area or Allocate_&_Write) the TAG of the memory area is set to 80h automatically.

INITIALIZE EEPROM in functionality Write_Block and Finalize_Block works with the first found memory area with TAG=80h. Therefore it is strongly recommended to change the TAG of the memory area to be finalized via specification of the FTAG parameter (e.g. Finalize_Block).

Note that an allocated memory block must be finalized always for consistency reasons.

Prerequisites The command can only be executed in the life cycle phase *INITIALIZATION*.

Options P3=LC, TR=OFF, AC=none, SM=none or mode x4h using the *StartKey*

3.20 INITIALIZE END

Command	INITIALIZE END
Function	Manufacturer use only
Standard / Use	Proprietary/ ROM-masked; *INITIALIZATION*

Parameters

CLA	INS	P1	P2
84h	00h	00h	00h

Data

LC	Command Data Field	LE
10h	(10h bytes) 8 bytes MAC + 8 bytes padding	./.

Response **./.**

Description

This command is used to finalize the life cycle phase *INITIALIZATION* and changes to the *PERSONALIZATION* life cycle phase automatically.

The command can only be executed using Secure Messaging (SM) mode x4h with the *StartKey*.

INITIALIZE END needs no command data. The data of the Command Data Field are enforced by SM mode x4h.

Prerequisites The command can only be executed in the life cycle phase *INITIALIZATION.*

Options P3=LC, TR=OFF, AC=none, SM= mode x4h using the *StartKey*

3.21 INTERNAL AUTHENTICATE

Command INTERNAL AUTHENTICATE

Function Perform cryptographic algorithms

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh	88h	00h	ID

Data

LC	Command Data Field	LE
	Input_Data	
*	(var. length)	**

* Length of subsequent command data field
** Length of requested response data field

Response

Response Data Field
Output_Data (MAC or SIGNATURE)
(var. length)

Description The command performs a MAC/SIGNATURE calculation using the Input_Data and padding data according to the key object referenced by the ID byte P2. Table 3.21-1 shows the ID Byte P2

INTERNAL AUTHENTICATE can only be used either with a

- **AUTH-MAC_Key**
  (i.e. AUTH object using tt='10' or tt='11' in the OPTIONS byte **and** using ALGO:
  - MAC (ALGO=80h) or
  - iMAC (ALGO=81h) or
  - MAC3 (ALGO=82h) or
  - iMAC3 (ALGO=83h))

or a

- **AUTH-Private_Key**
  (i.e. AUTH object using tt='10' in the OPTIONS byte **and** using ALGO:
  - RSA_SIG (ALGO=88h) or
  - DSA (ALGO=89h) (*) or
  - RSA_PURE_SIG (ALGO=8Ch) or
  - RSA_SIG_SHA-1 (ALGO=C8h) or
  - DSA_SHA-1 (ALGO=C9h) (*) or
  - RSA_PURE_SIG_SHA-1 (ALGO=CCh)).

(*) DSA algorithms are available only, if DSA_Package was loaded and activated before

Table 3.21-1 ID Byte P2

7	6	5	4	3	2	1	0	¬ **Bit Number / Meaning** ¯
z								z=0: Object search in MF
								z=1: Object search starting in current DF (backtracking)
	v	v	v	v	v	v	v	Object ID ‘*vvv vvvv*’:
								permitted values: ’000 0001’... ’111 1111’
0	0	0	0	0	0	0	0	Object referenced in CSE

In the Output_Data the MAC/SIGNATURE according to the AUTH object is returned.

If P2=00h, then CardOS/M4 “knows” which object shall be used. For this the AUTH object ID referenced in the CSE is used (IA component is byte 6 of the CSE). The backtracking mechanism for searching the AUTH object ID starts from the *current DF*.

Prerequisites The command can only be executed, if a AUTH object (i.e. the key usage bits *‘uuu’* of the object are *‘001’*) with the object ID specified in P2 exists (according to the backtracking mechanism of CardOS/M4) **and** the AUTH object is **not blocked** (see UseCounter and CurrentErrorCounter of objects) **and** the right referenced by the AUTH object’s *AC USE* is granted in the *security status* of the *current DF*.

Specials According to ISO 7816-4 the P2 (object ID) bits 2^5 ... 2^6 shall be set to ‘00’. For future compatibility it is recommended to take this restriction into account when defining object IDs.

Options P3=LC, Tr=ON, AC=USE (AUTH object referenced by object ID),
SM=USE-IN (AUTH object referenced by object ID) and/or
SM=USE-OUT (AUTH object referenced by object ID)

3.22 LOAD EXECUTABLE

Command LOAD EXECUTABLE

Function Activate a package

Standard / Use Proprietary / ROM-masked; *INITIALIZATION, ADMINISTRATION*

Parameters

CLA	INS	P1	P2
80h (0) / 84h (1)	20h	xxh	xxh

Data

LC	Command Data Field	LE
	Package_Checksum (0) / Internal information (1)	
*	(var. length)	./.

* Length of subsequent command data field
(0) Applies to CardOS/M4.00
(1) Applies to CardOS/M4.01

Response **./.**

Description The command activates a system package or an application package, which was installed in the smart card before. The parameters P1 and P2 may be used by the package as input parameters.

Installation of a System Package

For installation of a system package the following steps must be performed:

1st Changing to life cycle phase *INITIALIZATION*.
2nd Installing the image of the system package using the INITIALIZE EEPROM command.
3rd In order to activate the system the LOAD EXECUTABLE command must be performed finally.

Installation of an Application Package

In order to install an application the following steps must be performed:

1st Changing to life cycle phase *ADMINISTRATION*.
2nd Creation of a *CODE* file using the CREATE FILE command.
3rd Filling the *CODE* file with the application package data using the UPDATE BINARY command.
4th In order to activate an application package with a succeeding LOAD EXECUTABLE command the belonging *CODE* file must be selected.

Note **Generally during installation package data must not be changed anyhow as package activation via LOAD EXECUTABLE will fail, if package data are changed.**

Package LoadKey

The so-called *PackageLoadKey* is a system key of CardOS/M4 used for the Package_Checksum verification during the LOAD EXECUTABLE command. The *PackageLoadKey* is containing an error counter. The error counter of the *PackageLoadKey* is set to 0Ah during smart card production. If a Package_Checksum verification fails the error counter is always decreased. The error counter will never be increased, even after successful usage of the *PackageLoadKey*. This means that after 10 unsuccessful usages the command is blocked irreversibly.

The LOAD EXECUTABLE command checks the Package_Checksum using the image data (for system packages) respectively *CODE* file data (for application packages) and the *PackageLoadKey.*

- If the Package_Checksum verification fails the error counter of the *PackageLoadKey* will be decreased and an error code will be issued.
- If the Package_Checksum verification is successful the package will be activated for CardOS/M4.
    - In the case of an application package the belonging *CODE* file will be deactivated in order to prevent any change of the *CODE* file data. After deactivation of the *CODE* file a reactivation of the CODE file using the ACTIVATE FILE command is not possible.
    - In the case of a system package no additional action is performed by CardOS/M4.

Prerequisites

For activation of an application package (i.e. in life cycle phase *ADMINISTRATION*) the command can only be executed, if the error counter of the *PackageLoadKey* is not 00h **and** the right referenced by the MF's *AC EXECUTE* is granted.

For activation of a system package (i.e. in life cycle phase *INITIALIZATION*) the command can only be executed, if the error counter of the *PackageLoadKey* is not 00h.

Options

For CardOS/M4.00:
P3=LC, Tr=OFF, AC=EXECUTE (MF) for application packages, AC=none for system packages, SM=none

For CardOS/M4.01:
P3=LC, Tr=OFF, AC=EXECUTE (MF) for application packages, AC=none for system packages, SM=SM mode x4h with *PackageLoadKey*

3.23 MANAGE CHANNEL

Command MANAGE CHANNEL

Function Open or close a logical channel

Standard / Use ISO 7816-4 / CardOS/M4.00: Manage_Channel_Package, CardOS/M4.01: ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
00h	70h	MODE	CHANNEL-ID

Data

LC	Command Data Field	LE
*	empty	**

* Length of subsequent command data field
** Length of requested response data field (i.e. number of bytes to be read)

Response

Response Data Field
1 Byte / empty

Description This command opens or closes a logical channel to the smart card. During opening a channel a new smart card session will be established in the newly opened channel whereas the session of a channel will be closed during closing a channel.

The MODE parameter P1 specifies whether a logical channel shall be opened (P1=00h) or shall be closed (P1=80h). The CHANNEL-ID parameter P2 specifies which channel shall be opened or closed.

While opening a channel (P1=00h) the following applies:

- If P2=01h, 02h or 03h, then the respective channel will be opened and the response data field is empty.
- If P2=00h, then CardOS/M4 selects an unused channel and returns the channel ID of the selected channel in the response data field.

Depending on the current channel in which the MANAGE CHANNEL command is executed the following applies during opening a channel:

- If the current channel is channel 00h (i.e. the default channel), then the smart card session in the newly opened channel is created as after a reset of the smart card, i.e. no access rights are granted, no random numbers exist and the current file is the MF.
- If the current channel is **not** channel 00h, then the access rights, the current file and the current record are copied from the session of the current channel to the session of the newly opened channel. In the session of the newly opened channel **no random numbers** exist.

Access rights belonging to **TEST** objects with an **ARA_COUNTER > 00h** (i.e. limited usage of a granted access right), are **not** copied from the session of the current channel to the session of the newly opened channel for **security reasons**.

The MANAGE CHANNEL command allocates and uses the XRAM of the smart card. The XRAM will be allocated until the channel will be closed.

If a channel shall be closed, the CHANNEL-ID parameter P2 specifies the logical channel to be closed. The logical channel to be closed **must not** be the same as the one, in which the MANAGE CHANNEL command is executed. The default channel (channel 00h) **cannot** be closed. After having closed a channel the XRAM, which was used by the session of the closed channel is freed.

Note

In combination with the MANAGE CHANNEL command and during execution the DELETE FILE, ERASE FILES, PUT DATA or UNINSTALL PACKAGE commands

- **must be performed in logical channel 00h** and
- **other logical channels must not be opened**.

This usage of the DELETE FILE and PUT DATA commands in combination with the MANAGE CHANNEL command

- is mandatory for CardOS/M4.01 and
- may otherwise lead to inconsistent data in a CardOS/M4.00 smart card. This problem for CardOS/M4.00 can be solved, if other security mechanisms of CardOS/M4 are used to prohibit other usages of DELETE FILE respectively PUT DATA in combination with MANAGE CHANNEL as described above.

Prerequisites

The command can only be executed, if enough XRAM is available for a new session in the smart card.

Options

P3=LE, Tr=ON, AC=none, SM=none

3.24 MANAGE SECURITY ENVIRONMENT (MSE)

Command MANAGE SECURITY ENVIRONMENT (MSE)

Function Load a *CSE* (RESTORE) or set a component of the *CSE* (SET*)*

Standard / Use ISO 7816-8 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
00h	22h	Mode	SE Object ID / CSE-Component TAG

Data

LC	Command Data Field			LE
	Empty / CSE-Component Control_Reference_Template (CRT)			
*	0 bytes /			./.
	T (TAG)	L (Length)	V (Value)	
	1 byte	1 byte	1 byte	

* Length of subsequent command data field

Response ./.

Description Using the MANAGE SECURITY ENVIRONMENT (MSE) command a *CSE* can be **loaded** (mode RESTORE) or one component of the *CSE* can be set (mode SET) (see table 3.24-1).

Table 3.24-1 MSE command using different modes

MSE Command			
Mode	**P1**	**P2**	**Command Data Field**
RESTORE	x3h	SE Object ID	empty
SET	x1h	CSE-Component TAG	CSE-Component CRT

The ***CSE*** *(current security environment)* is a temporarily variable in RAM, which **must be present**, if for example a PSO command shall be executed. This means that after a smart card reset no *CSE* exists. The *CSE* consist of the following 6 *CSE* components:

- **CC** Component (used by PSO_CCC, PSO_VCC),
- **DS** Component (used by (PSO_CDS, PSO_VDS),
- **HASH** Component (used by PSO_H),
- **CON** Component (used by PSO_DEC, PSO_ENC),
- **TEST** Component (used by EXTERNAL AUTHENTICATE, VERIFY),
- **IA** Component (used by INTERNAL AUTHENTICATE)

MSE Mode RESTORE
In order to get a *CSE* the command MANAGE SECURITY ENVIRONMENT (MSE) in mode RESTORE must be executed.

- For this MSE searches (according to the backtracking mechanism of CardOS/M4) for an SE object with an object ID specified in P2.

- If the specified SE object is found, the SE object data are copied to the *CSE*. After this the *CSE* can be used for subsequent commands using the CSE (i.e.
  - all PSO commands
  - optionally EXTERNAL AUTHENTICATE
  - optionally INTERNAL AUTHENTICATE and
  - optionally VERIFY).
- In order to perform the MSE command successfully the referenced SE object must have been installed before. Using the PUT DATA_SECI command a single SE object, which is **located** in the *current DF*, can be **installed**.

MSE Mode SET
If a CSE exists, then a CSE component (specified by it's CSE-Component TAG) can be modified using MSE in SET mode. In the MSE command data field the belonging CSE-Component CRT (Control_Reference_Template) must be specified then. CRTs are TLV coded and contain the ID of the referenced BS object in the value filed.

Table 3.24-2 shows the relations between the CSE components, the belonging CSE-Component TAG in parameter P2, the belonging CSE-Component CRTs, and the commands using the CSE components.

Table 3.24-2 Relations between CSE components, MSE and commands using the CSE

CSE	**MSE command**				**Commands using CSE component**
CSE component	**CSE-Component TAG (P2)**	**CSE-Component CRT (Command Data Field)**			
		T	**L**	**V**	
CC component	B4h	83h / 84h	01h	PSO Object ID	PSO_CCC, PSO_VCC
DS component	B6h	83h / 84h	01h	PSO Object ID	PSO_CDS, PSO_VDS
HASH component	AAh	83h / 84h	01h	PSO Object ID	PSO_H
CON component	B8h	83h / 84h	01h	PSO Object ID	PSO_DEC, PSO_ENC
TEST component	A4h	83h / 84h	01h	TEST Object ID	EXTERNAL AUTENTICATE, VERIFY
IA component	A2h	83h / 84h	01h	AUTH Object ID	INTERNAL AUTHENTICATE

Prerequisites The MSE command in RESTORE mode can only be executed, if the right referenced by the SE object's *AC RESTORE* is granted in the *security status* of the *current DF*.

The MSE command in SET mode is not secured by an AC.

Options P3=LC; Tr=ON;

RESTORE Mode:	AC=RESTORE (SE object), SM=none
SET Mode:	AC=none, SM=none

3.25 PERFORM SECURITY OPERATION (PSO)

Command PERFORM SECURITY OPERATION (PSO)

Function Perform a cryptographic operation

Standard / Use ISO 7816-8 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh / 1xh	2Ah	Selection_Part1	Selection_Part2

Data

LC	Command Data Field	LE
*	var. length or empty	**

* Length of subsequent command data field
** Length of requested response data field

Response

Response Data Field
var. length or empty

Description The PERFORM SECURITY OPERATION (PSO) command executes a cryptographic operation. For the 7 existing cryptographic operations PSO supports 7 different PSO modes (see table 3.25-1), which are described in the following sections. The PSO mode (or cryptographic operation to be executed) is selected by P1 and P2 according to table 3.25-1.

Table 3.25-1 Supported PSO modes

P1	P2	Command Data	Response Data	Needed CSE component	Description
00h	A2h	Input_Data, MAC	-	CC component	**PSO_VCC:** VERIFY CRYPTOGRAPHIC CHECKSUM Mode: The command verifies the MAC given in the command data field. The result of the verification is given in the status code SW1/SW2.
8Eh	80h	Input_Data (plain)	MAC	CC component	**PSO_CCC:** COMPUTE CRYPTOGRAPHIC CHECKSUM Mode: The command calculates the MAC for the Input_Data. The result is given in the response data.
00h	A8h	DS / Input_Data, DS		DS component	**PSO_VDS:** VERIFY DIGITAL SIGNATURE Mode: The command verifies the DS (digital signature) given in the command data field. The result of the verification is given in the status code SW1/SW2.
9Eh	9Ah	Input_Data (plain)	DS (Digital Signature)	DS component	**PSO_CDS:** COMPUTE DIGITAL SIGNATURE Mode: The command calculates a DS for the Input_Data. The result is given in the response data.
86h	80h	Input_Data (Plain)	Enciphered Input	CON component	**PSO_ENC:** ENCIPHER Mode: The command enciphers the Input_Data using a DES_Key or a Public_Key. The result is given in the response data.
80h	86h	Input_Data (enciphered)	Deciphered Input	CON component	**PSO_DEC:** DECIPHER Mode: The command deciphers the Input data using a DES_Key or a Private_Key. The result is given in the response data.
90h	80h	Input_Data	Hash / none	HASH component	**PSO_H:** HASHING Mode The command calculates a hash value for the Input_Data and stores it internally. The result is given in the response data when the last PSO_H command of a command chain was entered.

For successful command execution PSO needs a CSE, which was loaded by the MSE or SELECT FILE (only in the case of the default SE object) command before. The CSE itself must reference an appropriate BS object, which **must** be a PSO object.

According to the backtracking mechanism of CardOS/M4 PSO searches for the CSE first and then searches for the BS object.

- This means that first the CSE is searched for in the path back from the *current DF* to the MF starting from the *current DF*.
- If the CSE is found, then the PSO command searches for the BS object, which is referenced by the found CSE. For this second step PSO searches for the PSO object in the path back from the *current DF* to the MF starting from the *current DF*.
- As a consequence of the behavior described the PSO object needed by PSO may not be located in the *current DF*.

The PSO command supports command chaining, which is specified by CLA = 1xh. Command chaining is supported using several PSO commands only. If the PSO command chain is interrupted by another command the internally generated hash value is withdrawn immediately.

3.25.1 PSO_VCC

Command PSO (PERFORM SECURITY OPERATION)

Function Perform the VERIFY CRYPTOGRAPHIC CHECKSUM operation

Standard / Use ISO 7816-8 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh	2Ah	00h	A2h

Data

LC	Command Data Field						LE
	Input_Data_Object			MAC_Data_Object			
	T	L	V	T	L	V	
*	80h	n	Input_Data	8Eh	08h	MAC_Data	./.
	1 byte	1 byte	n bytes	1 byte	1 byte	8 bytes	

* Length of subsequent command data field
Note: Input_Data_Object and MAC_Data_Object can be specified in any order inside the command data field.

Response **./.**

Description PSO_VCC verifies the MAC data given in the MAC_Data_Object with a MAC calculated for the Input_Data. The result of the verification is given in the status code SW1/SW2.

For command execution a *CSE* must exist. The CC component must reference a

- **PSO-MAC_Key**
  (i.e. a PSO object using tt='01' or tt='11' in the OPTIONS byte **and** using ALGO:
  - MAC (ALGO=80h) or
  - iMAC (ALGO=81h) or
  - MAC3 (ALGO=82h) or
  - iMAC3 (ALGO=83h)).

Prerequisites The PSO_VCC command can only be executed, if the right referenced by the PSO object's *AC USE* is granted in the *security status* of the *current DF*.

Options P3=LC, Tr=ON, AC=USE (PSO object referenced by the *CSE's* CC component), SM=USE-IN (PSO object referenced by the *CSE's* CC component)

3.25.2 PSO_CCC

Command PSO (PERFORM SECURITY OPERATION)

Function Perform the COMPUTE CRYPTOGRAPHIC CHECKSUM operation

Standard / Use ISO 7816-8 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh	2Ah	8Eh	80h

Data

LC	Command Data Field	LE
	Input_Data (plain)	
*	(var. length)	**

* Length of subsequent command data field
** Length of requested response data field

Response

Response Data Field
MAC
8 bytes

Description PSO_CCC calculates the MAC for the Input_Data. The MAC is given in the response data.

For command execution a *CSE* must exist. The *CSE's* CC component must reference a

- **PSO-MAC_Key**
  (i.e. a PSO object using tt='10' or tt='11' in the OPTIONS byte **and** using ALGO:
  - MAC (ALGO=80h) or
  - iMAC (ALGO=81h) or
  - MAC3 (ALGO=82h) or
  - iMAC3 (ALGO=83h)).

Prerequisites The PSO_CCC command can only be executed, if the right referenced by the PSO object's *AC USE* is granted in the *security status* of the *current DF*.

Options P3=LC, Tr=ON, AC=USE (PSO object referenced by the *CSE's* CC component),
SM=USE-IN (PSO object referenced by the *CSE's* CC component) and/or
SM=USE-OUT (PSO object referenced by the *CSE's* CC component)

3.25.3 PSO_VDS

Command PSO (PERFORM SECURITY OPERATION)

Function Perform the VERIFY DIGITAL SIGNATURE operation

Standard / Use ISO 7816-8 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh / 1xh	2Ah	00h	A8h

Data

LC	Command Data Field						LE
	Input_Data_Object			DS_Data_Object			
	T	L	V	T	L	V	
*	9Ah	n	Input_Data	9Eh	m	DS_Data	./.
	1 byte / empty	1 byte / empty	n bytes / empty	1 byte / empty	1 byte / empty	m bytes / empty	

* Length of subsequent command data field
Note: Input Data Object and DS Data Object can be specified in any order inside the command data field.

Response **./.**

Description PSO_VDS verifies the DS_Data (Digital Signature Data) given in the DS_Data_Object 9Eh. For this first the belonging DS hash value is calculated from the DS_Data using a Public_Key. Then either an internal hash value (mode1 or 3) or a hash value given in the Input_Data (mode2) or a hash value calculated for the Input_Data (mode4) is verified with the DS hash value. The result of the verification is given in the status code SW1/SW2.

For command execution a *CSE* must exist. The *CSE's* DS component must reference a

- **PSO-Public_Key**
  (i.e. PSO object using tt='01' in the OPTIONS byte **and** using ALGO:
  - RSA_SIG (ALGO=88h) or
  - DSA (ALGO=89h) $^{(*)}$ or
  - RSA_PURE_SIG (ALGO=8Ch) or
  - RSA_SIG_SHA-1 (ALGO=C8h) or
  - DSA_SHA-1 (ALGO=C9h) $^{(*)}$ or
  - RSA_PURE_SIG_SHA-1 (ALGO=CCh)).

$^{(*)}$ DSA algorithms are available only, if DSA_Package was loaded and activated before

According to table 3.25.3-1 the PSO_VDS command can be used in 4 different modes, i.e. depending on the used algorithm for the necessary PSO-Public_Key the Input_Data are interpreted and used differently.

The internal hash value used in the modes 1 and 3 (see table 3.25.3-1) may be calculated using a single PSO_H command or a finished PSO_H command chain. The specified hash value in mode 2 may be calculated in the smart card before (i.e. using PSO_H command(s)) or may be calculated outside the smart card.

The DS_Data_Object must be specified for all commands of mode 1 to 3. For mode 4, where PSO_VDS command chaining is permitted (see table 3.25.3-1), only the **last** command of the chain contains the DS_Data_Object.

For all PSO_VDS commands in mode 4 using CLA=1xh the length n of the Input_Data must be a multiple of the used hash algorithm's specific data length (e.g. n must be a multiple of 40h for the SHA-1 algorithm). For the last command of a PSO_VDS chain (i.e. PSO_VDS command using CLA=0xh) n may have any length, if the length of the DS_Data_Object and the Input_Data_Object does not exceed the I/O buffer size of CardOS/M4.

Table 3.25.3-1 Different modes of the PSO_VDS command

Used Public Key Algorithm	Mode	Input_ Data_ Object 9Ah specified (y/n)	Input_Data interpreted as	Command Chaining permitted (y/n)	Description
RSA_SIG or DSA or RSA_PURE_SIG	1	n	-	n	**Internal** hash value (i.e. hash value calculated with preceding PSO_H command(s)) is used for DS hash value verification.
	2	y	Hash value	n	**Specified** hash value (i.e. hash value specified in the input data object 9Ah) is used for DS hash value verification.
RSA_SIG_SHA-1 or DSA_SHA-1 or RSA_PURE_SIG_SHA-1	3	n	-	n	**Internal** hash value (i.e. hash value calculated with preceding PSO_H command(s)) is used for DS hash value verification.
	4	y	Input Data (plain)	y	**Specified input data** (in plain text) are hashed internally. The internal hash value is used for DS hash value verification during the last command of a PSO_VDS command chain (i.e. CLA=0xh for the last PSO_VDS command in the chain.).

(*) DSA algorithms are available only, if DSA_Package was loaded and activated before

Note that PSO_VDS may be used with DSI objects.

- DSI objects are supported by CardOS/M4.01 in any case.
- DSI objects are **not** supported by CardOS/M4.00 as long as the algorithms RSA_SIG, DSA or RSA_PURE_SIG are used. In order to support DSI objects with these algorithms the so-called DSI_Extension_Package must have been loaded and activated for CardOS/M4.00.

Prerequisites The PSO_VDS command can only be executed, if the right referenced by the PSO object's *AC USE* is granted in the *security status* of the *current DF.*

Options P3=LC, Tr=ON, AC=USE (PSO object referenced by the *CSE's* DS component), SM=USE-IN (PSO object referenced by the *CSE's* DS component)

3.25.4 PSO_CDS

Command PSO (PERFORM SECURITY OPERATION)

Function Perform the COMPUTE DIGITAL SIGNATURE operation

Standard / Use ISO 7816-8 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
00h / 10h	2Ah	9Eh	9Ah

Data

LC	Command Data Field	LE
	Input_Data	
*	n bytes	**

* Length of subsequent command data field
** Length of requested response data field

Response

Response Data Field
DS_Data
m bytes

Description PSO_CDS calculates the DS (digital signature) for the Input_Data, which are interpreted as plain text or as hash value. The DS_Data are given in the response data.

For command execution a *CSE* must exist. The *CSE's* DS component must reference a

- **PSO-Private_Key**
  (i.e. PSO object using tt='10' in the OPTIONS byte **and** using ALGO:
  - RSA_SIG (ALGO=88h) or
  - DSA (ALGO=88h) (*) or
  - RSA_PURE_SIG (ALGO=8Ch) or
  - RSA_SIG_SHA-1 (ALGO=C8h) or
  - DSA_SHA-1 (ALGO=C8h) (*) or
  - RSA_PURE_SIG_SHA-1 (ALGO=CCh)).

(*) DSA algorithms are available only, if DSA_Package was loaded and activated before

According to table 3.25.4-1 the PSO_CDS command can be used in 4 different modes, i.e. depending on the used algorithm for the necessary PSO-Private_Key the Input_Data are interpreted and used differently.

Table 3.25.4-1 Different modes of the PSO_CDS command

Used Private Key Algorithm	Mode	Input Data interpreted as	Command Chaining permitted (y/n)	Description
RSA_SIG or DSA or RSA_PURE_SIG	1	-	n	**Internal** hash value (i.e. hash value calculated with preceding PSO_H command(s)) is used for DS hash value calculation.
	2	Hash value	n	**Specified** hash value (i.e. hash value specified in the input data) is used for DS calculation.
RSA_SIG_SHA-1 or DSA_SHA-1 or RSA_PURE_SIG_SHA-1	3	-	n	**Internal** hash value (i.e. hash value calculated with preceding PSO_H command(s)) is used for DS hash value calculation.
	4	Input Data (plain)	y	**Specified input data** (in plain text) are hashed internally. The internal hash value is used for DS calculation during the last command of a PSO_CDS command chain (i.e. CLA=0xh for the last PSO_CDS command in the chain.).

(*) DSA algorithms are available only, if DSA_Package was loaded and activated before

Note Modes 1 and 3 are **not** supported for CardOS/M4.00 whereas for CardOS/M4.01 all modes are supported.

PSO_CDS may be used with DSI objects.

- DSI objects are supported by CardOS/M4.01 in any case.
- DSI objects are **not** supported by CardOS/M4.00 as long as the algorithms
    - **RSA_SIG**,
    - **DSA** or
    - **RSA_PURE_SIG**

  are used. In order to support DSI objects with these algorithms the so-called **DSI_Extension_Package** must have been loaded and activated for CardOS/M4.00.

The specified hash value in mode 2 (see table 3.25.4-1) may be calculated in the smart card before (i.e. using PSO_H command(s)) or may be calculated outside the smart card.

For all PSO_CDS commands in mode 4 using CLA=1xh the length n of the Input_Data must be a multiple of the used hash algorithm's specific data length (e.g. n must be a multiple of 40h for the SHA-1 algorithm). For the last command of a PSO_CDS chain (i.e. PSO_CDS command using CLA=0xh) n may have any length, if the length of the Input_Data does not exceed the I/O buffer size of CardOS/M4.

Prerequisites The PSO_CDS command can only be executed, if the right referenced by the PSO object's *AC USE* is granted in the *security status* of the *current DF*.

Options P3=LC, Tr=ON, AC=USE (PSO object referenced by the *CSE's* DS component), SM=USE-IN (PSO object referenced by the *CSE's* DS component), SM=USE-OUT (PSO object referenced by the *CSE's* DS component)

3.25.5 PSO_ENC

Command	PSO (PERFORM SECURITY OPERATION)
Function	Perform the ENCIPHER operation
Standard / Use	ISO 7816-8 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh	2Ah	86h	80h

Data

LC	Command Data Field	LE
	Input_Data (plain)	
*	n bytes	**

* Length of subsequent command data field
** Length of requested response data field

Response

Response Data Field	
Padding Indicator	Enciphered_Input_Data
00h	
1 byte	m bytes

Description

PSO_ENC enciphers the Input_Data given as plain text using a symmetric or asymmetric key. The result is given in the response data field as Enciphered_Input_Data preceded by a fixed padding indicator.

For command execution a *CSE* must exist. The *CSE's* CON component must reference a

- **PSO-DES_Key**
  (i.e. PSO object using tt='01' or tt='11' in the OPTIONS byte **and** using ALGO:
  - DES_ECB (ALGO=00h) or
  - DES_CBC (ALGO=01h) or
  - DES3_CBC(ALGO=03h))

or a

- **PSO-Public_Key**
  (i.e. PSO object using tt='01' in the OPTIONS byte and using ALGO:
  - RSA (ALGO=08h) or
  - RSA_PURE (ALGO=0Ch)).

For the data lengths the following is valid:
m ≥ n for PSO-DES_Keys
m > n for PSO-Public_Keys
(as the length of the plain input data +1 ≤ modulus length)

Prerequisites The PSO_ENC command can only be executed, if the right referenced by the PSO object’s *AC USE* is granted in the *security status* of the *current DF*.

Options P3=LC, Tr=ON, AC=USE (PSO object referenced by the *CSE’s* CON component), SM=USE-IN (AUTH object referenced by the *CSE’s* CON component) and/or SM=USE-OUT IN (AUTH object referenced by the *CSE’s* CON component)

3.25.6 PSO_DEC

Command PSO (PERFORM SECURITY OPERATION)

Function Perform the DECIPHER operation

Standard / Use ISO 7816-8 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh	2Ah	80h	86h

Data

LC	Command Data Field		LE
	Padding Indicator	Enciphered_Input_Data	
*	00h		**
	1 byte	m bytes	

* Length of subsequent command data field
** Length of requested response data field

Response

Response Data Field
Input_Data (plain)
n bytes

Description PSO_DEC deciphers the Enciphered_Input_Data preceded by a fixed padding indicator using a symmetric or asymmetric key. The result is given in the response data field as Input_Data in plain text.

For command execution a *CSE* must exist. The *CSE's* CON component must reference a

- **PSO-DES_Key**
  (i.e. PSO object using tt='10' or tt='11' in the OPTIONS byte **and** using ALGO:
  - DES_ECB (ALGO=00h) or
  - DES_CBC (ALGO=01h) or
  - DES3_CBC(ALGO=03h))

or a

- **PSO-Private_Key**
  (i.e. PSO object using tt='10' in the OPTIONS byte and using ALGO:
  - RSA (ALGO=08h) or
  - RSA_PURE (ALGO=0Ch)).

For the data lengths the following is valid:
m $\geq$ n for PSO-DES_Keys
m > n for PSO-Private_Keys

Prerequisites The PSO_DEC command can only be executed, if the right referenced by the PSO object's *AC USE* is granted in the *security status* of the *current DF*.

Options P3=LC, Tr=ON, AC=USE (PSO object referenced by the *CSE's* CON component), SM=USE-IN (PSO object referenced by the *CSE's* CON component) an/or SM=USE-OUT (PSO object referenced by the *CSE's* CON component)

3.25.7 PSO_H

Command PSO (PERFORM SECURITY OPERATION)

Function Perform the HASHING operation

Standard / Use ISO 7816-8 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh / 1xh	2Ah	90h	80h

Data

LC	Command Data Field	LE
	Input_Data (plain)	
*	n bytes	**

* Length of subsequent command data field
** Length of requested response data field

Response

Response Data Field
Hash_Value / empty
m bytes / 0 bytes

Description

PSO_H calculates a hash value for the Input_Data given in plain text. For any PSO_H command the result is **always** stored internally.

For command execution a *CSE* must exist. The *CSE's* HASH component must reference a

- **PSO-HASH_Key**
  (i.e. PSO object using tt='01', tt='10' or tt='11' in the OPTIONS byte **and** using ALGO:
  - SHA-1 (ALGO=CFh).

Command chaining is permitted. Only for the last PSO_H command of a command chain or a single PSO_H command (i.e. CLA = 0xh) the result is given in the response data field as Hash_Value. For all PSO_H commands using CLA = 1xh (i.e. not last command of a PSO_H command chain) no result is returned, i.e. the response data field is empty.

Inside a command chain the internally stored hash value is taken into account for subsequent PSO_H commands. Therefore a PSO_H command chain can be used for hashing large amounts of input data exceeding the I/O buffer size of a CardOS/M4 smart card.

For all PSO_H commands using CLA = 1xh the input data length n must be a multiple of the used hash algorithm's specific data length (e.g. n must be a multiple of 40h for the SHA-1 algorithm (ALGO = CFh)). For PSO_H using CLA = 0xh n may have any length not exceeding the I/O buffer size of CardOS/M4. Therefore the input data of the last command of a PSO_H command chain or a single PSO_H command may have any length.

The response data length is hash algorithm-specific (e.g. m = 20 for SHA-1 (ALGO = CFh)).

Prerequisites The PSO_H command can only be executed, if the right referenced by the PSO object's *AC USE* is granted in the *security status* of the *current DF*.

Options P3=LC, Tr=ON, AC=USE (PSO object referenced by the *CSE's* HASH component), SM=USE-IN (PSO object referenced by the *CSE's* HASH component) an/or SM=USE-OUT (PSO object referenced by the *CSE's* HASH component)

3.26 PERSONALIZE

Command PERSONALIZE

Function Personalizer use only

Standard / Use Proprietary / ROM-masked; *PERSONALIZATION*

Parameters

CLA	INS	P1	P2
80h / 84h	08h	PERS_NO	00h

Data

LC	Command Data Field		LE
	Personalization_Data		
*	Sequence Number	Personalization Data	./.
	2 bytes	n bytes	

* Length of subsequent command data field

Response **./.**

Description The card issuer has defined a file system structure during the life cycle phase *INITIALIZATION* and knows exactly where individual data have to be written. Therefore during *INITIALIZATION* placeholders were loaded at each requested place in the file system.

Using the PERSONALIZE command the real data will be “plugged in” during the *PERSONALIZATION* life cycle phase at a location where the belonging placeholder was loaded during *INITIALIZATION phase.*

The placeholders contain flags in a control byte, which define, whether SM must be used or not.

If the PERSONALIZE command has replaced the **last placeholder** of the placeholder list, which is specified by the personalization number PERS_NO (P1), the belonging application descriptor will be deleted automatically. If the **last application descriptor** of the smart card is deleted, the life cycle phase will be changed to *OPERATIONAL* (i.e. the *PERSONALIZATION* phase is finished) automatically.

The command can be executed unsecured (CLA=80h) and secured (CLA=84h; encrypted and protected) using Secure Messaging (SM) mode x4h with the belonging PersonalizationKey.

Prerequisites The card must be in the life cycle phases *PERSONALIZATION.*

Options P3=LC, Tr=OFF, AC=none, SM=SM mode x4 with PersonalizationKey belonging to PERS_NO.

3.27 PHASE CONTROL

Command PHASE CONTROL

Function Changes from life cycle phase *ADMINISTRATION* to *OPERATIONAL* and vice versa.

Standard / Use Proprietary / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
80h	10h	00h	00h

Data

LC	Command Data Field	LE
*	empty	./.

* Length of subsequent command data field

Response ./.

Description The command changes from life cycle phase *ADMINISTRATION* to *OPERATIONAL*. This change is permanently valid, i.e. the life cycle phase is still valid after a reset of the smart card.

The life cycle phase change from *OPERATIONAL* to *ADMINISTRATION* is only temporary, i.e. the life cycle phase is set back to *OPERATIONAL* after a reset of the smart card. It is also possible to change back to *OPERATIONAL* directly with the PHASE CONTROL command.

When using the PHASE CONTROL command in combination with the MANAGE CHANNEL command note that for CardOS/M4.00 PHASE CONTROL works with channel 0 only, for CardOS/M4.01 PHASE CONTROL works with all channels.

Prerequisites Changing from *ADMINISTRATION* to *OPERATIONAL* is not protected. Changing from *OPERATIONAL* to *ADMINISTRATION* is controlled by the right referenced by the current DF's *AC LCYCLE*.

The command can be executed in the life cycle phases *ADMINISTRATION* and *OPERATIONAL*.

Options P3=LC, Tr=OFF, AC=none or AC=LCYCLE (current DF), SM=none

3.28 PUT DATA

Command PUT DATA

Function Install / administrate / overwrite different objects

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION*

Parameters

CLA	INS	P1	P2
0xh	DAh	MODE	Object ID / Template ID

Data

LC	Command Data Field	LE
*	(var. length)	./.

* Length of subsequent command data field

Response ./.

Description Depending on P1 and P2 (see table 3.28-1) the command supports the 5 modes

- **PUT DATA_ATR,**
- **PUT DATA_DSI,**
- **PUT DATA_FCI,**
- **PUT DATA_OCI,** and
- **PUT DATA_SECI**

which are described in the following sections.

Table 3.28-1 MODE byte P1 and Object ID / Template ID byte P2

P1	P2	Description
02h	C0h	**PUT DATA_ATR** **Installation / Overwriting** of the smart card's ATR via TLV data
02h	C2h	**PUT DATA_DSI** **Installation / Overwriting** of DSI objects
01h	6Fh	**PUT DATA_FCI** Installation / Administration of files via FCIs (File Control Information) **Installation** of the following FCI: AID (DF Name) **Administration** of the following FCIs: AC Definitions SM Definitions AID (DF Name)
01h	6Eh	**PUT DATA_OCI** **Installation / Overwriting** of the following objects via OCIs (Object Control Information): **BS Objects:** TEST Object AUTH Object SM Object PSO Object
01h	6Dh	**PUT DATA_SECI** **Installation / Overwriting** of the following objects via SECIs (Security Environment Control Information): **SE Objects**

PUT DATA never checks consistency of the ATR, DSI, FCI, OCI or SECI data.

Note

In combination with the MANAGE CHANNEL command and during execution the PUT DATA command

- **must be performed in logical channel 00h** and
- **other logical channels must not be opened**.

This usage of the PUT DATA command in combination with the MANAGE CHANNEL command

- is mandatory for CardOS/M4.01 and
- may otherwise lead to inconsistent data in a CardOS/M4.00 smart card. This problem for CardOS/M4.00 can be solved, if other security mechanisms of CardOS/M4 are used to prohibit other usages of PUT DATA in combination with MANAGE CHANNEL as described above.

3.28.1 PUT DATA_ATR

Command PUT DATA

Function Install / overwrite the smart card's ATR

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION*

Parameters

CLA	INS	P1	P2
0xh	DAh	02h	C0h

Data

LC	Command Data Field		LE
	ATR_Object_Data		
*	n+1 bytes		./.
	PROT	ATR_String (without Initial Character and with checksum according to ISO 7816-3)	
	1 byte	n bytes	

* Length of subsequent command data field

Response ./.

Description PUT DATA_ATR installs or overwrites a EEPROM-ATR. The object installed with this command needs (n+3) bytes in EEPROM.

CardOS/M4 has a default ATR, which is stored in ROM. The ROM-ATRs of CardOS/M4.00 and CardOS/M4.01 are listed in table 3.28.1-1.

Table 3.28.1-1 ROM-ATR Strings for CardOS/M4.00 and CardOS/M4.01

CardOS Version	ROM-ATR String
CardOS/M4.00	**3Bh E2h 00h FFh C1h 10h 31h FEh 55h C8h 02h 9Ch**
CardOS/M4.01	**3Bh F2h 98h 00h FFh C1h 10h 31h FEh 55h C8h 03h 15h**

The last byte of each ROM-ATR string denotes the checksum, which must be set according to ISO 7816-3, if the CardOS/M4 smart card shall be used with ISO compliant smart card readers.

For conformity reasons with ISO 7816-3 it is recommended to change the historical bytes of the ROM-ATR only.

If only a ROM-ATR is present in the smart card, then the ROM-ATR is sent after a reset of the smart card. If an EEPROM-ATR exists (beside the ROM-ATR), then the EEPROM-ATR is sent after a reset of the smart card.

In order to modify the default ROM-ATR of the smart card according to the needs of the application it is possible to install a user-defined ATR in EEPROM. This EEPROM-ATR is installed and can be overwritten using the PUT DATA_ATR command.

Using the PROT byte the behavior of the smart card can be modified.

- If PROT=00h and a T=0 transmission package is activated in the smart card, then the smart card starts with the T=0 protocol independently of what is specified in the ATR string.
- If PROT=01h, then the smart card starts with the T=1 protocol independently of what is specified in the ATR_String.

As the smart card does not check consistency between the setting in the PROT byte and in the ATR_String the user of PUT DATA_ATR must take care of consistency. **If the data of the PROT byte and the ATR_String are not consistent the smart card may not work correctly.**

The ATR_String must contain a complete ATR string including the checksum and excluding the initial character (see ISO 7816-3).

The EEPROM-ATR must be installed in the MF (i.e. the MF must be the *current DF* when running PUT DATA_ATR).

Note

In combination with the MANAGE CHANNEL command and during execution the PUT DATA_ATR command

- **must be performed in logical channel 00h** and
- **other logical channels must not be opened**.

This usage of the PUT DATA_ATR command in combination with the MANAGE CHANNEL command

- is mandatory for CardOS/M4.01 and
- may otherwise lead to inconsistent data in a CardOS/M4.00 smart card. This problem for CardOS/M4.00 can be solved, if other security mechanisms of CardOS/M4 are used to prohibit other usages of PUT DATA_ATR in combination with MANAGE CHANNEL as described above.

Prerequisites

PUT DATA_ATR can only be executed, if the right referenced by the *current DF*'s *AC APPEND* (for installation of a EEPROM-ATR) or *AC UPDATE* (for administration of a EEPROM-ATR) is granted in the *security status* of the *current DF*.

Specials

The EEPROM-ATR must be installed in the MF (i.e. MF = *current DF*).

Options

P3=LC, Tr=OFF, AC=APPEND/UPDATE (*current DF*),
SM=APPEND/UPDATE (*current DF*)

3.28.2 PUT DATA_DSI

Command PUT DATA

Function Install / overwrite a DSI object

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION*

Parameters

CLA	INS	P1	P2
0xh	DAh	02h	C2h

Data

LC	Command Data Field	LE
	DSI_Object_Data	
*	(d bytes; var. length)	./.

* Length of subsequent command data field

Response ./.

Description Using the PUT DATA_DSI command a so-called DSI (Digital Signature Information) object can be installed or overwritten in a DF. **Per DF only one DSI object** can be installed. The DSI objects needs (d+2) bytes in EEPROM.

When using the signing algorithms, i.e.

- **RSA_SIG (ALGO=88h)**,
- **RSA_PURE_SIG (ALGO=8Ch)**,
- **RSA_SIG_SHA-1**,
- **RSA_PURE_SIG_SHA-1,**

CardOS/M4 searches for a DSI object according to the backtracking mechanism of CardOS/M4. If a DSI object was found the DSI_Object_Data is taken into account during the input data are padded (see chapter 2.4.2.2).

Note DSI objects may be used with the commands PSO_CDS and PSO_VDS. In this context the following applies:

- DSI objects are supported by CardOS/M4.01 in any case.
- DSI objects are **not** supported by CardOS/M4.00 as long as the algorithms
    - o **RSA_SIG**,
    - o **DSA** or
    - o **RSA_PURE_SIG**

  are used. In order to support DSI objects with these algorithms the so-called **DSI_Extension_Package** must have been loaded and activated for CardOS/M4.00.

Using a DSI object with the signing algorithms a DigestInfo scheme according to [16] can be realized. If the hashing algorithm SHA-1 is used for instance, the DSI_Object_Data looks as the DSI_Object_Data which is recommended in [18]. For both approaches the corresponding DSI_Object_Data with the SHA-1 hash algorithm looks as follows:

DSI_Object_Data String
30h 21h 30h 09h 06h 05h 2Bh 0Eh 03h 02h 1Ah 05h 00h 04h 14h

Note

In combination with the MANAGE CHANNEL command and during execution the PUT DATA_DSI command

- **must be performed in logical channel 00h** and
- **other logical channels must not be opened**.

This usage of the PUT DATA_DSI command in combination with the MANAGE CHANNEL command

- is mandatory for CardOS/M4.01 and
- may otherwise lead to inconsistent data in a CardOS/M4.00 smart card. This problem for CardOS/M4.00 can be solved, if other security mechanisms of CardOS/M4 are used to prohibit other usages of PUT DATA_DSI in combination with MANAGE CHANNEL as described above.

Prerequisites

PUT DATA_DSI can only be executed, if the right referenced by the *current DF*'s *AC APPEND* (for installation of a DSI object) or *AC UPDATE* (for administration of a DSI object) is granted in the *security status* of the *current DF*.

Options

P3=LC, Tr=OFF, AC=APPEND/UPDATE (*current DF*),
SM=APPEND/UPDATE (*current DF*)

3.28.3 PUT DATA_FCI

Command PUT DATA

Function Install / administrate files via FCIs

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION*

Parameters

CLA	INS	P1	P2
0xh	DAh	01h	6Fh

Data

LC	Command Data Field									LE
	VALUE field of FCI template									
*	n bytes									./.
	FCI 1			FCI 2 / empty			FCI 3 / empty			
	T	L	V	T	L	V	T	L	V	
	For TLV data of FCIs see tables below									

* Length of subsequent command data field

Response ./.

Description PUT DATA_FCI performs one of the following actions on the *current EF* or *current DF*:

- Installation of a AID-FCI (for DFs only) or
- Administration of one or several of the following FCIs:
  - AC Definitions
  - SM Definitions
  - AID (DFs only; can be administrated separately only).

During installation of an AID-FCI the AID data are stored in the *current DF*. During administration existing FCIs of the *current EF* or *current DF* can be changed.

In order to perform the command a TLV coded FCI template (FCIT) is given to the PUT DATA_FCI command (T=P2, L=LC, V=Command Data Field). The FCIT's VALUE field contains all FCIs, which are TLV coded also according to tables 3.28.3-1, 3.28.3-2 and 3.28.3-3

Table 3.28.3-1 Description of FCIs for the PUT DATA_FCI command

T	L	V (Value)	m /o	Description
84h	p	DF Name (for DFs only)	o	DF Name
86h	q	Definitions for Access Conditions	o	AC Definitions (see table 3.28.3-2)
8Bh	r	Definitions for Secure Messaging	o	SM Definitions (see table 3.28.3-3)

m/o:	mandatory / optional
p:	Length of DF name in bytes
q:	Length of AC definitions in bytes
r:	Length of SM definitions in bytes

Table 3.28.3-2 AC Definitions in FCI 86h

Byte No.	AC definitions (possible values)	Meaning for MF	Meaning for DF	Meaning for EF
1	00h ... 7Fh, FFh	AC LCYCLE	AC LCYCLE	AC READ (Data)
2	00h ... 7Fh, FFh	AC UPDATE (Objects)	AC UPDATE (Objects)	AC UPDATE (Data)
3	00h ... 7Fh, FFh	AC APPEND (Objects)	AC APPEND (Objects)	AC APPEND (Record) / rfu
4	00h ... 7Fh, FFh	AC DEACTIVATE (MF)	AC DEACTIVATE (DF)	AC DEACTIVATE (EF)
5	00h ... 7Fh, FFh	AC ACTIVATE (MF)	AC ACTIVATE (DF)	AC ACTIVATE (EF)
6	00h ... 7Fh, FFh	AC ERASE (MF)	AC DELETE (DF)	AC DELETE (EF)
7	00h ... 7Fh, FFh	AC ADMIN (MF)	AC ADMIN (DF)	AC ADMIN (EF)
8	00h ... 7Fh, FFh	AC CREATE (Files)	AC CREATE (Files)	AC INCREASE / rfu
9	00h ... 7Fh, FFh	AC EXECUTE	rfu	AC DECREASE / rfu

Meaning of possible values:

00h:	Access action referenced by AC can **always** be performed (i.e. right is granted).
01h ... 7Fh	Access action referenced by AC can be performed, if right 01h ... 7Fh is granted.
FFh:	Access action referenced by AC can **never** be performed.

Since FCI 86h is optional in the FCI template neither FCI 86h nor any of the AC definitions must be specified.

- If FCI 86h is specified in the FCI template and none/not all of the described AC definitions are specified with FCI 86h, then the missing AC definitions are supposed to be specified with **FFh** values **implicitly** (i.e. the access action referenced by the AC can **never** be performed).
- If FCI 86h is not specified in the FCI template, then the AC definitions will not be changed.

Table 3.28.3-3 SM Definitions in FCI 8Bh

Byte No.	SM definitions (possible values)	Meaning for MF	Meaning for DF	Meaning for EF
1	00h ... FFh	rfu	rfu	ENC READ-OUT (Data)
2	00h ... FFh	rfu	rfu	SIG READ-OUT (Data)
3	00h ... FFh	ENC UPDATE (0) / ENC UPDATE/APPEND (1)	ENC UPDATE (0) / ENC UPDATE/APPEND (1)	ENC UPDATE (Data)
4	00h ... FFh	SIG UPDATE (0) / SIG UPDATE/APPEND (1)	SIG UPDATE (0) / SIG UPDATE/APPEND (1)	SIG UPDATE (Data)
5	00h ... FFh	ENC APPEND (0) / rfu (1)	ENC APPEND (0) / rfu (1)	ENC APPEND (Record) / rfu
6	00h ... FFh	SIG APPEND (0) / rfu (1)	SIG APPEND (0) / rfu (1)	SIG APPEND (Record) / rfu
7	00h ... FFh	rfu	rfu	rfu
8	00h ... FFh	rfu	rfu	rfu
9	00h ... FFh	rfu	rfu	rfu
10	00h ... FFh	rfu	rfu	rfu
11	00h ... FFh	rfu	ENC DELETE (DF)	ENC DELETE (EF)
12	00h ... FFh	rfu	SIG DELETE (DF)	SIG DELETE (EF)
13	00h ... FFh	ENC ADMIN (MF)	ENC ADMIN (DF)	ENC ADMIN (EF)
14	00h ... FFh	SIG ADMIN (MF)	SIG ADMIN (DF)	SIG ADMIN (EF)
15	00h ... FFh	ENC CREATE (Files)	ENC CREATE (files)	ENC INCREASE-IN (Data) / rfu
16	00h ... FFh	SIG CREATE (Files)	SIG CREATE (files)	SIG INCREASE-IN (Data) / rfu
17	00h ... FFh	rfu	rfu	ENC DECREASE-IN (Data) / rfu
18	00h ... FFh	rfu	rfu	SIG DECREASE-IN (Data) / rfu
19	00h ... FFh	ENC GENKEY	ENC GENKEY	ENC INCREASE-OUT (Data) / rfu
20	00h ... FFh	SIG GENKEY	SIG GENKEY	SIG INCREASE-OUT (Data) / rfu
21	00h ... FFh	rfu	rfu	ENC DECREASE-OUT (Data) / rfu
22	00h ... FFh	rfu	rfu	SIG DECREASE-OUT (Data) / rfu
23	00h ... FFh	rfu	rfu	ENC READ-IN (DATA) / rfu
24	00h ... FFh	rfu	rfu	SIG READ-IN (Data) / rfu

Meaning of possible values:

00h ... FEh	ID of SM-key to be used for referenced access action, ENC/SIG and communication direction.
FFh:	**No** SM.
(0):	CardOS/M4.00 (Objects)
(1):	CardOS/M4.01 (Objects)

Since FCI 8Bh is optional neither FCI 8Bh nor any of the SM definitions must be specified.

- If FCI 8Bh is specified in the FCI template and none/not all of the described SM definitions are specified with FCI 8Bh, then the missing SM definitions are supposed to be specified with **FFh** values **implicitly** (i.e. **no SM** is specified).
- If FCI 8Bh is not specified in the FCI template, then the SM definitions will not be changed.

Note

In combination with the MANAGE CHANNEL command and during execution the PUT DATA_FCI command

- **must be performed in logical channel 00h** and
- **other logical channels must not be opened**.

This usage of the PUT DATA_FCI command in combination with the MANAGE CHANNEL command

- is mandatory for CardOS/M4.01 and
- may otherwise lead to inconsistent data in a CardOS/M4.00 smart card. This problem for CardOS/M4.00 can be solved, if other security mechanisms of CardOS/M4 are used to prohibit other usages of PUT

DATA_FCI in combination with MANAGE CHANNEL as described above.

Prerequisites PUT DATA_FCI can only be executed, if the right referenced by file’s *AC ADMIN* is granted in the *security status* of the *current DF*.

Options P3=LC, Tr=OFF, AC=ADMIN (file), SM=ADMIN (file)

3.28.4 PUT DATA_OCI

Command PUT DATA

Function Install / overwrite BS objects in *current DF*

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION*

Parameters

CLA	INS	P1	P2
0xh	DAh	01h	6Eh

Data

LC	Command Data Field															LE
	VALUE field of OCI template															
*	n bytes															./.
	OCI 1			OCI 2			OCI 3			OCI 4			OCI 5 / empty			
	T	L	V	T	L	V	T	L	V	T	L	V	T	L	V	
	For TLV data of OCIs see tables below															

* Length of subsequent command data field

Response ./.

Description The command installs or overwrites an BS object in the *current DF*. During installation of a BS object the object data are stored in the *current DF*. The BS object needs (s+14) bytes in EEPROM (see table 3.28.4-1).

In order to perform the command a TLV coded OCI template (OCIT) is given to the PUT DATA_OCI command (*T*=P2, *L*=LC, *V*=Command Data Field). The OCIT's VALUE field contains all OCIs, which are TLV coded also according to tables 3.28.4-1 ... 3.28.4-6.

Table 3.28.4-1 Description of OCIs for the PUT DATA_OCI command

T	L	V (Value)	m /o	Description
83h	02h	Byte 1: Object Class Byte 2: Object ID	m	Object Address
85h	08h	Byte 1: OPTIONS Byte 2: FLAGS Byte 3: ALGO Byte 4: ERRCNT Byte 5: USECOUNT Byte 6: DEK Byte 7: ARA_COUNTER Byte 8: MINLEN	m	Object Parameters
86h	q	Definitions for Access Conditions	m	AC Definitions
8Bh	r	Definitions for Secure Messaging	o	SM Definitions
8Fh	s	Object Data	m	Object Data

m/o:	mandatory / optional
q:	Length of AC definitions in bytes
r:	Length of SM definitions in bytes
s:	Length of Object Data in bytes

Table 3.28.4-2 Object Address (OCI 83h)

Byte No.	Bit No.								Description		
	7	6	5	4	3	2	1	0			
1									**Object Class Byte**		
	0								**BS Objects:**	fixed to '0'	
		p							**BS Objects:**	p='0'	
			u	u	u				**BS Objects:**	uuu = Object Usage	
											Class Byte for kkk='000'
									uuu='000':	TEST Object	(00h)
									uuu='001':	AUTH Object	(08h)
									uuu='010':	SM Object	(10h)
									uuu='100':	PSO Object	(20h)
						k	k	k	**BS Objects:**	kkk = Object Component	
									ALGO	*ALGO-ID*	*Meaning*
									kkk='000':		
									DES_ECB	(00h)	Single Component
									DES_CBC	(01h)	Single Component
									DES3_CBC	(03h)	Single Component
									Logical	(7Fh)	Single Component
									MAC	(80h)	Single Component
									iMAC	(81h)	Single Component
									MAC3	(82h)	Single Component
									iMAC3	(83h)	Single Component
									PIN (CHV)	(87h)	Single Component
									DSA (*)	(89h)	Prime P
									DSA_SHA-1 (*)	(C9h)	Prime P
									RSA	(08h)	Modulus N
									RSA_PURE	(0Ch)	Modulus N
									RSA_SIG	(88h)	Modulus N
									RSA_PURE_SIG	(8Ch)	Modulus N
									RSA_SIG_SHA-1	(C8h)	Modulus N
									RSA_PURE_SIG_SHA-1	(CCh)	Modulus N
									kkk='001':		
									DSA (*)	(89h)	Prime Q
									DSA_SHA-1 (*)	(C9h)	Prime Q
									RSA	(08h)	Exponent E or D
									RSA_PURE	(0Ch)	Exponent E or D
									RSA_SIG	(88h)	Exponent E or D
									RSA_PURE_SIG	(8Ch)	Exponent E or D
									RSA_SIG_SHA-1	(C8h)	Exponent E or D
									RSA_PURE_SIG_SHA-1	(CCh)	Exponent E or D
									kkk='010':		
									DSA (*)	(89h)	Base G
									DSA_SHA-1 (*)	(C9h)	Base G
									kkk='011':		
									DSA (*)	(89h)	Factor X or Y
									DSA_SHA-1 (*)	(C9h)	Factor X or Y)

(*): DSA algorithms are available only, if DSA_Package was loaded and activated before.

Table 3.28.4-2 Object Address (OCI 83h) continued

Byte No.	Bit No.								Description	
	7	6	5	4	3	2	1	0		
2									**Object ID Byte**	
	v	v	v	v	v	v	v	v	vvvv vvvv = ‘1111 1111’ (FFh) not permitted other valid values depend on the type of object:	
									Object	*Value Range*
									BS Objects:	
									TEST Object used for: EXTERNAL AUTHENTICATE SELECT FILE (i.e. for implicit logical test) VERIFY	01h ... 7Fh
									AUTH Object	01h ... FEh
									SM Object	01h ... FEh
									PSO Object	01h ... FEh

Table 3.28.4-3 Object Parameters (OCI 85h)

Byte No.	Bit No.								Description	
	7	6	5	4	3	2	1	0		
1									**OPTIONS Byte**	
	p								**TEST Object**: p='0': Right withdrawn after failed test p='1': Right remains after failed test **SM Object**: p='0' Using **no** random number with SM p='1' Using random number with SM **Other Objects**: no meaning (shall be set to '0')	
		n							**Enable-Bit n** for Generate-Bit (see FLAGS Byte, g Bit) n='1' Generate-Bit enabled n='0' Generate-Bit disabled	
			q						**BS Objects** (depending on used algorithm): q='0' Last or single Object Component q='1' Object Component following	
				0					Must be set to '0'	
					r				r='0' Object will be passed on (i.e. object can be used in the DF where it was installed and in all child-DFs.) r='1' Object will not be passed on (i.e. object can only be used in the DF where it was installed.)	
						0			Shall be set to '0'	
							t	t	**BS Objects:** The tt settings depend on the BS object and the action to be performed with the BS object:	
									Commands	**Possible tt values**
									EXTERNAL AUTHENTICATE [TEST-MAC_Key]	01, 11
									EXTERNAL AUTHENTICATE [TEST-Public_Key]	01
									INTERNAL AUTHENTICATE [AUTH-MAC_Key]	10, 11
									INTERNAL AUTHENTICATE [AUTH-Private_Key]	10
									PSO_CCC [PSO-MAC_Key]	10, 11
									PSO_CDS [PSO-Private_Key]	10
									PSO_DEC [PSO-DES_Key]	10, 11
									PSO_DEC [PSO-Private_Key]	10
									PSO_ENC [PSO-DES_Key]	01, 11
									PSO_ENC [PSO-Public_Key]	01
									PSO_H [PSO-Hash_Key]	don't care
									PSO_VCC [PSO-MAC_Key]	01, 11
									PSO_VDS [PSO-Public_Key]	01
									VERIFY [PIN TEST]	10, 11
									SM (Mode xCh only)	
									Command-SM_ENC	10, 11
									Command-SM_SIG	10, 11
									Command-SM_ENC_SIG	10, 11
									Response-SM_ENC	01, 11
									Response-SM_SIG	01, 11
									Response-SM_ENC_SIG	01, 11

Table 3.28.4-3 Object Parameters (OCI 85h) continued

Byte No.	Bit No.								Description
	7	6	5	4	3	2	1	0	
2									**FLAGS Byte**
	x								**DeathBit x** [(*)]**:** x='0' CardOS/M4 does *not* reach life cycle phase DEATH if USECOUNT reaches 00h. x='1' CardOS/M4 reaches life cycle phase DEATH if USECOUNT reaches 00h and every DeathEnableBit in the complete DF/MF hierarchy is set.
		u							**TEST Objects**: u='0' Referenced right **not** granted during selection of the DF where the object is installed (shall be default). u='1' Referenced right granted during selection of the DF where the object is installed (e.g. for GSM). **Other Objects**: no meaning (shall be set to '0')
			g						**Generate-Bit g:** The GenerateBit g is active only if the EnableBit n in the OPTIONS Byte is set to '1' g='1' The key can be generated (i.e. overwritten) using the GENERATE KEY PAIR command. After successful execution of GENERATE KEY PAIR GenerateBit g is set to '0'. g='0' The key cannot be overwritten using the GENERATE KEY PAIR command.
				v					Must be set to v='0'
					x	x	x	x	**BS Objects**: MaxErrorCounter: The CurrentErrorCounter (see ERRCNT byte) is set to the value of the MaxErrorCounter after a successful test. MaxErrorCounter='0000' deactivates the CurrentErrorCounter.

(*): DeathBit is available only, if in CardOS/M4.00 UseCounter_Package was loaded and activated before. Always available in CardOS/M4.01.

Table 3.28.4-3 Object Parameters (OCI 85h) continued

Byte No.	Bit No.								Description
	7	6	5	4	3	2	1	0	
3									**ALGO Byte**
	y	y	y	y	y	y	y	y	**BS Objects**: The byte specifies the following algorithms: *ALGO-ID* *ALGO* 00h DES_ECB 01h DES_CBC 03h DES3_CBC 08h RSA 0Ch RSA_PURE 7Fh Logical 80h MAC 81h iMAC 82h MAC3 83h iMAC3 87h PIN (CHV) 88h RSA_SIG 89h DSA (*) 8Ch RSA_PURE_SIG C8h RSA_SIG_SHA-1 C9h DSA_SHA-1 (*) CCh RSA_PURE_SIG_SHA-1 CFh SHA-1
4									**ERRCNT Byte**
	0								**PureResetRetry r:** **(applies to CardOS/M4.01 only)** r='1' RESET RETRY COUNTER is *not* allowed to change the PIN r='0' RESET RETRY COUNTER is allowed to change the PIN
		w							**BS Objects**: w='0': CurrentErrorCounter decreased **after** test execution (POST-Decrement) w='1': CurrentErrorCounter decreased **before** test execution (PRE-Decrement)
			z						**BS Objects**: z='0': CurrentErrorCounter set to value of MaxErrorCounter after successful test (default). z='1': CurrentErrorCounter **not** set to value of MaxErrorCounter after successful test.
				m					**TerminateBit m:** **(applies to CardOS/M4.01 only)** m='1' If CurrentErrorCounter reaches '0000' then the DF which contains the BS object is terminated, i.e. an "implicit" TERMINATE DF operation is performed. This applies to keys using a CurrentErrorCounter (e.g. SM keys, C/R test keys,...) m='0' If the CurrentErrorCounter reaches '0000' the DF which contains the BS object is **not** terminated.
					x	x	x	x	**BS Objects**: CurrentErrorCounter: The CurrentErrorCounter is set to the value of the MaxErrorCounter (see FLAGS byte) after a successful test and is decreased after a failed test. If CurrentErrorCounter='0000', then the BS object is blocked.

(*): DSA algorithms are available only, if DSA_Package was loaded and activated before.

Table 3.28.4-3 Object Parameters (OCI 85h) continued

Byte No.	Bit No.								Description
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
5									**USECOUNT Byte**
	y	y	y	y	y	y	y	y	**BS Objects:** The byte represents the **UseCounter**, which is decreased each time the object is used. So the number of usages of an object can be limited absolutely. UseCounter=00h: Object blocked UseCounter=FFh Number of Object Usages not limited.
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
6									**DEK Byte**
	1	1	1	1	1	1	1	1	**must** be set to FFh
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
7									**ARA_COUNTER Byte**
	v	v	v	v	v	v	v	v	**TEST Objects:** This byte limits the usage of a granted access right by a counter, which is decreased each time the access right is used (e.g. as AC for file access, USE of SM keys,...) ARA_COUNTER = 00h or = FFh: unlimited usage ARA_COUNTER > 00h and < FFh: limited usage **Other Objects:** no meaning (shall be set to 00h)
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
8									**MINLEN Byte**
	r	r	r	r	r	r	r	r	**TEST C/R or SM SIG_Objects**: MINLEN is the minimum length of the random number to be used. **PIN TEST** object: MINLEN is the minimum length of the PIN data. **Other Objects**: CardOS/M4.01: shall be set to 00h CardOS/M4.00: shall be set to 00h or eventually to the length of the object data (see CHANGE REFERENCE DATA)

Table 3.28.4-4 AC Definitions in OCI 86h

Byte No.	AC definitions (possible values)	Meaning
1	00h ... 7Fh, FFh	AC USE (Object)
2	00h ... 7Fh, FFh	AC CHANGE (Object Data)
3	00h ... 7Fh, FFh	AC UNBLOCK (BS Objects)
4	00h ... 7Fh, FFh	rfu
5	00h ... 7Fh, FFh	rfu
6	00h ... 7Fh, FFh	rfu
7	00h ... 7Fh, FFh	AC GENKEY
8	00h ... 7Fh, FFh	AC SIGNDEC (for Sign_By_Descryption_Key_Package only)

Meaning of possible values:

00h:	Access action referenced by AC can **always** be performed (i.e. right is granted).
01h ... 7Fh	Access action referenced by AC can be performed, if right 01h ... 7Fh is granted.
FFh:	Access action referenced by AC can **never** be performed.

Unlike the OCI 86h which is mandatory in the OCI template neither the OCI 8Bh nor any of the AC/SM definitions must be specified explicitly.

- If none/not all of the described AC definitions are specified with OCI 86h, then the missing AC definitions are supposed to be specified with **FFh** values **implicitly** (i.e. the access action referenced by the AC can **never** be performed).
- If either OCI 8Bh is missing or none/not all of the described SM definitions are specified with OCI 8Bh, then the missing SM definitions are supposed to be specified with **FFh** values **implicitly** (i.e. **no SM** is specified).

Table 3.28.4-5 SM Definitions in OCI 8Bh

Byte No.	SM definitions (possible values)	Meaning
1	00h ... FFh	ENC USE-IN
2	00h ... FFh	SIG USE-IN
3	00h ... FFh	ENC CHANGE (Object Data)
4	00h ... FFh	SIG CHANGE (Object Data)
5	00h ... FFh	ENC UNBLOCK (BS Objects)
6	00h ... FFh	SIG UNBLOCK (BS Objects)
7	00h ... FFh	ENC rfu
8	00h ... FFh	SIG rfu
9	00h ... FFh	ENC rfu
10	00h ... FFh	SIG rfu
11	00h ... FFh	ENC rfu
12	00h ... FFh	SIG rfu
13	00h ... FFh	ENC rfu
14	00h ... FFh	SIG rfu
15	00h ... FFh	ENC USE-OUT
16	00h ... FFh	SIG USE-OUT

Meaning of possible values:

00h ... FEh	ID of SM-key to be used for referenced access action, ENC/SIG and communication direction.
FFh:	**No** SM.

The object data are specified in OCI 8Fh.

- Object data of BS objects, their length and their format depend on the object algorithms (ALGO byte of OCI 85h) and can vary therefore.
- Especially for PSO objects, which shall represent a Hash_Key the object data shall be 1 byte in length and must be used as shown in table 3.28.4-6.

Table 3.28.4-6 Object Data in OCI 8Fh for a Hash_Key (PSO object) only

Byte No.	Recommended value	Meaning
1	FFh	Dummy Byte

The Dummy Byte is needed for object data compatibility only.

Note

In combination with the MANAGE CHANNEL command and during execution the PUT DATA_OCI command

- **must be performed in logical channel 00h** and
- **other logical channels must not be opened**.

This usage of the PUT DATA_OCI command in combination with the MANAGE CHANNEL command

- is mandatory for CardOS/M4.01 and
- may otherwise lead to inconsistent data in a CardOS/M4.00 smart card. This problem for CardOS/M4.00 can be solved, if other security mechanisms of CardOS/M4 are used to prohibit other usages of PUT DATA_OCI in combination with MANAGE CHANNEL as described above.

Prerequisites

PUT DATA_OCI can only be executed, if the right referenced by *current DF*'s *AC APPEND* (in the case of installation) or *AC UPDATE* (in the case of administration) is granted in the *security status* of the *current DF*.

Options

P3=LC, Tr=OFF, AC=APPEND/UPDATE (*current DF*),
SM=APPEND/UPDATE (*current DF*)

3.28.5 PUT DATA_SECI

Command PUT DATA

Function Install / overwrite SE objects via SECIs

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION*

Parameters

CLA	INS	P1	P2
0xh	DAh	01h	6Dh

Data

LC	Command Data Field									LE
	VALUE field of SECI template									
*	n bytes									./.
	SECI 1			SECI 2			SECI 3			
	T	L	V	T	L	V	T	L	V	
	For TLV data of SECIs see tables below									

* Length of subsequent command data field

Response ./.

Description The command installs or overwrites an SE object in the *current DF*. During installation of an SE object the object data are stored in the *current DF*. A SE object needs 11 bytes in EEPROM always

In order to perform the command a TLV coded SECI template (SECIT) is given to the PUT DATA_SECI command (T=P2, L=LC, V=Command Data Field). The SECIT's VALUE field contains all SECIs, which are TLV coded also according to tables 3.28.5-1, 3.28.5-2 and 3.28.5-3.

Table 3.28.5-1 Description of SECIs for the PUT DATA_SECI command

T	L	V (Value)	m /o	Description
83h	01h	SE Object ID: 00h ... FEh: ID range 01h: Default SE Object	m	ID of SE Object
86h	q	Definitions for Access Conditions of SE Object	m	AC Definitions (see table 3.28.5--2)
8Fh	06	SE Object Data	m	Data of SE Object (see table 3.28.5--3)

m/o: mandatory / optional
q: Length of AC definitions in bytes

Table 3.28.5-2 AC Definitions in SECI 86h

Byte No.	AC definitions (possible values)	Meaning for MF
1	00h ... 7Fh, FFh	AC RESTORE (SE Object)
2	00h ... 7Fh, FFh	AC STORE (SE Object) rfu

Meaning of possible values:

00h:	Access action referenced by AC can **always** be performed (i.e. right is granted).
01h ... 7Fh	Access action referenced by AC can be performed, if right 01h ... 7Fh is granted.
FFh:	Access action referenced by AC can **never** be performed.

For the TEST object, which references the AC RESTORE (SE Object) it is strongly recommended to use the setting ARA_COUNTER=00h, because with each DF context change via SELECT FILE the ARA_COUNTER would be decreased implicitly. So evaluation of a functioning ARA_COUNTER setting especially for the AC RESTORE would be a very difficult and time-consuming process.

Unlike the OCI 86h which is mandatory in the OCI template none of the AC definitions must be specified explicitly.

- If none/not all of the described AC definitions are specified with OCI 86h, then the missing AC definitions are supposed to be specified with **FFh** values **implicitly** (i.e. the access action referenced by the AC can **never** be performed).

Table 3.28.5-3 SE Object Data in SECI 8Fh

Byte No.	Possible values	Referenced by MSE (Mode SET) using CRT_TAG	Meaning
1	00h ... FEh	B4h	BS Object ID for **CC component** Commands: PSO_CCC, PSO_VCC BS Object has object class 20h (i.e. **PSO Object**)
2	00h ... FEh	B6h	BS Object ID for **DS component** Commands: PSO_CDS, PSO_VDS BS Object has object class 20h (i.e. **PSO Object**)
3	00h ... FEh	AAh	BS Object ID for **HASH component** Command: PSO_H BS Object has object class 20h (i.e. **PSO Object**)
4	00h ... FEh	B8h	BS Object ID for **CON component** Commands: PSO_DEC, PSO_ENC BS Object has object class 20h (i.e. **PSO Object**)
5	00h ... 7Fh	A4h	BS Object ID for **TEST component** Commands: EXTERNAL AUTHENTICATE, VERIFY BS Object has object class 00h (i.e. **TEST Object**)
6	00h ... FEh	A2h	BS Object ID for **IA component** Command: INTERNAL AUTHENTICATE BS Object has object class 08h (i.e. **AUTH Object**)

IA Internal Authentication

Note

In combination with the MANAGE CHANNEL command and during execution the PUT DATA_SECI command

- **must be performed in logical channel 00h** and
- **other logical channels must not be opened**.

This usage of the PUT DATA command in combination with the MANAGE CHANNEL command

- is mandatory for CardOS/M4.01 and
- may otherwise lead to inconsistent data in a CardOS/M4.00 smart card. This problem for CardOS/M4.00 can be solved, if other security mechanisms of CardOS/M4 are used to prohibit other usages of PUT DATA_SECI in combination with MANAGE CHANNEL as described above.

Prerequisites PUT DATA_SECI can only be executed, if the right referenced by *current DF*'s *AC APPEND* (in the case of installation) or *AC UPDATE* (in the case of administration) is granted in the *security status* of the *current DF*.

Options P3=LC, Tr=OFF, AC=APPEND/UPDATE (*current DF*),
SM=APPEND/UPDATE (*current DF*)

3.29 READ BINARY

Command READ BINARY

Function Read a *BINARY* file

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh	B0h	HI	LO

Data

LC	Command Data Field	LE
*	empty	**

* Length of subsequent command data field
** Length of requested response data field (i.e. number of bytes to be read)

Response

Response Data Field
(var. length)

Description This command reads data from a *BINARY* file. The start offset is specified in HI-LO. The start offset specifies the starting point of the read access.

LE specifies the number of bytes to be read. If LE=00h the maximum number of available bytes is assumed. The maximum number for LE is limited either by the number of available I/O buffer bytes (i.e. 256 bytes) or by the number of bytes in between starting point and the end of file.

If bit 2^7 is set in HI, HI is considered the SFI for selecting the desired file (bits 2^0 to 2^4). 00h is then assumed to be the high portion of the start offset (see table 3.29-1).

Table 3.29-1 HI Byte P1

7	6	5	4	3	2	1	0	¬ **Bit Number / Meaning** ¯
0	*x*	*x*	*x*	*x*	*x*	*x*	*x*	Use current EF and HI-value=*'0xxx xxxx'*
1	*y*	*y*	*p*	*p*	*p*	*p*	*p*	Use SFI '*p pppp*' *('p pppp' = '1 1111' not permitted)* and
								HI-value=00h; yy-Bits are don't care (shall be set to '00')

Using a SFI the FID to be used is built adding the constant value **FE00h** to the specified SFI.

Prerequisites The command can only be executed, if the right referenced by the file's *AC READ* is granted in the *security status* of the *current DF*.
READ BINARY can only be used for *BINARY* and *CODE* files.

Options P3=LE, Tr=ON, AC=READ (file), SM=READ-OUT (file) and/or SM=READ-IN (file)

3.30 READ RECORD

Command READ RECORD

Function Read a record oriented file

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh	B2h	REC	EF-ID / MODE

Data

LC	Command Data Field	LE
*	empty	**

* Length of subsequent command data field
** Length of requested response data field (i.e. number of bytes to be read)

Response

Response Data Field
Record_Data (var. length)

Description This command reads a record from a *LINEAR FIXED*, *CYCLIC FIXED* or *LINEAR TLV* file. Byte P2 specifies *EF-ID* (i.e. file to be used) and *MODE* (i.e. access mode) according to table 3.30-1.

Table 3.30-1 EF-ID / MODE Byte P2

7	6	5	4	3	2	1	0	¬ **Bit Number / Meaning** ¯
0	*0*	*0*	*0*	*0*				Use current EF
p	*p*	*p*	*p*	*p*				Use SFI *'p pppp' ('p pppp' = '1 1111' not permitted)*
					0	0	0	**FIRST** mode: Read first record REC contains the record identifier
					0	0	1	**LAST** mode: Read last record REC contains the record identifier
					0	1	0	**NEXT** mode: Read the next record REC contains the record identifier
					0	1	1	**PREV** mode: Read the previous record REC contains the record identifier
					1	0	0	**CURRENT / ABSOLUTE** mode: Read the current record (REC=0) or read via the record number (REC contains the logical number of the record)

Using a SFI the FID to be used is built adding the constant value **FE00h** to the specified SFI.

The modes *FIRST*, *LAST*, *NEXT* and *PREV* affect the internal record pointer, which references the last record read. The *TAG* of the desired record can be specified as record identifier for these modes, when *LINEAR TLV* files are involved. Relative specifications (i.e., *NEXT*, *PREV*) always refer to all records with the *TAG* specified in *REC*. If *REC* contains 00h, then all records are taken into account. For all other files only 00h is permitted.

If the internal record pointer points to the last record of a non-cyclic file and a READ RECORD in *NEXT* mode is issued or the internal record pointer points to the first record of a non-cyclic file and a READ RECORD in *PREV* mode is issued, then the error code 6A83h (i.e., record not found) is returned.

In mode *CURRENT*, either the last record read with one of the modes *FIRST*, *LAST*, *NEXT* or *PREV* is read again (*REC* = 00h), or the record with the logical number specified in *REC* is read.

With the exception of *CYCLIC FIXED* files, when a file is selected again, *NEXT* reads the first record and *PREV* reads the last record. *CURRENT* will cause an error.

When *CYCLIC FIXED* files are involved, the internal record pointer is already pointing to the first record during selection (i.e., *NEXT* reads the second and *PREV* reads the last but one record. *CURRENT* can be used to read the logically first record).

The length of the data to be read is specified by LE and by the record length. If LE = 00h, all records up to the end of the file are read.

Prerequisites The command can only be executed, if the right referenced by the file's *AC READ* is granted in the *security status* of the *current DF*.
READ RECORD can only be used for *LINEAR TLV*, *LINEAR FIXED* and *CYCLIC FIXED* files.

Options P3=LE, Tr=ON, AC=READ (file), SM=READ-OUT (file) and/or SM=READ-IN (file)

3.31 RESET RETRY COUNTER

Command RESET RETRY COUNTER

Function Reset error counter of a BS object to its maximum

Standard / Use ISO 7816-8 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh	2Ch	Object Parms / MODE	ID

Data

LC	Command Data Field		LE
	Unblock_CHV_Data	New_CHV_Data	
*	m bytes / empty	n bytes / empty	./.

* Length of subsequent command data field

Response ./.

Description The command sets the current error counter (i.e. *CurrentErrorCounter* field in the *ERRCNT* byte) of a **BS object** (except logical tests) to its maximum value (i.e. *MaxErrorCounter* field in *FLAGS* byte).

The command cannot be used for logical tests as for logical tests the current error counter has no meaning.

Table 3.31-1 explains the meaning of the Object-Parms/MODE byte P1.

Table 3.31-1 Object-Parms/MODE Byte P1

7	6	5	4	3	2	1	0	¬ **Bit Number / Meaning** ¯
0	0							Bits 2^6 ... 2^7 must be set to '00'
		U	u	u				Object Usage Bits *'uuu'*
					0	0	0	MODE **'000'**
								Command data contain Unblock CHV and New CHV
								Only if P2 references a PIN TEST object
					0	0	1	MODE **'001'**
								Command data contain Unblock CHV only
								Only if AC UNBLOCK (Object) references a PIN TEST object
					0	1	1	MODE **'011'**
								Command data field is empty
								Possible for BS objects referenced by P2

Table 3.31-2 explains the meaning of the ID byte P2.

Table 3.31-2 ID Byte P2

7	6	5	4	3	2	1	0	¬ Bit Number / Meaning ¯
z								z=0: Object search in MF
								z=1: Object search starting in current DF (backtracking)
	v	V	v	v	v	v	v	Object ID ‘*vvv vvvv*’:
								permitted values: ’000 0001’... ’111 1111’

First the command searches for a BS object using an object class according to the object usage bits ‘uuu’ given in P1 and the object ID given in P2. If such a BS object is found the command distinguishes between the three modes ‘000’, ‘001’ and ‘011’. The command searches for the BS object according to the backtracking mechanism of CardOS/M4.

MODE ‘000’ can only be used with PIN TEST objects (i.e. the BS object referenced by P2 and found via backtracking must be a PIN TEST object).

MODE ‘001’ and MODE ‘011’ can be used with **any** BS object referenced by P2. For these modes the BS object data are **not** changed by RESET RETRY COUNTER.

If MODE = ‘000’ or MODE = ‘001’ **and** the AC *UNBLOCK* of the BS object, which is referenced by P2, references a PIN TEST, then the Unblock_CHV_Data are verified with the PIN TEST referenced by the AC *UNBLOCK* implicitly. If the AC *UNBLOCK* of the BS object, which is referenced by P2, is set to always, CardOS/M4 searches for the PIN TEST via the CSE.

- If the verification is successful or the AC *UNBLOCK* of the PIN TEST object is set to ‘00h’ (i.e. always) command execution proceeds. Otherwise command execution is aborted.
- After this the CurrentErrorCounter of the BS object referenced by P2 is set to its maximum.
- For MODE = ‘000’ the following applies additionally:
    - The PIN TEST data (i.e. BS object data) is updated with the New_CHV_Data regardless of what is specified in the *AC CHANGE* of the PIN TEST object. The length *n* of the New_CHV_Data **need not** fit the data length of the PIN TEST object to be updated.
    - The UseCounter of the PIN TEST object (i.e. BS object referenced by P2) will not be changed except the AC *CHANGE* of the PIN TEST object references itself. Possibly the error counter of the PIN TEST object referenced by *AC UNBLOCK* is changed.

In CardOS/M4.01 with the RESET RETRY COUNTER command a PIN can be changed if old verification data is present or if the right **AC UNBLOCK** referring a LOGICAL TEST (i.e. TEST objects with ALGO= 7Fh)) is granted. CardOS/M4.01 iterates (from first to last byte) through the logical expression data of the LOGICAL TEST to find a valid PIN TEST as follows:

- Operator bytes (0x00 and 0xFF) are skipped.
- The AC number must correspond to a PIN TEST
- The PIN TEST must be usable (GenerateBit clear, MaxErrorCounter=0 or CurrentErrorCounter>0, USECOUNT>0).

The first matching PIN TEST is taken as reference for the old verification data. The search for this PIN TEST starts in the DF of the LOGICAL TEST.

In CardOS/M4.01 with Mode ´000´ given in P1 the PIN TEST data (i.e. BS object data) is updated with the New_CHV_Data only if the **PureResetRetry** Bit in the PIN TEST is set to **zero**. Otherwise the command is rejected.
This Bit is located in the ERRCNT Byte of the Object Parameters of the PIN TEST (refer the PUT DATA_OCI command Tag 85h).

MODE ‘011’ can be used with any BS object except TEST-Logical objects. With this MODE the command data field is empty, i.e. no implicit verification is performed. In order to perform the command in this MODE the access right referenced by the BS object’s AC *UNBLOCK* must be granted. Finally the current error counter of the BS object is set to its maximum.

Prerequisites The command can only be executed, if the right referenced by the BS object’s *AC UNBLOCK* is granted in the *security status* of the *current DF* or in the case of sent Unblock_CHV_Data if the Unblock_CHV_Data are identical to the PIN TEST data (see description above).

Options P3=LC, Tr=ON, AC=UNBLOCK (BS object); SM=UNBLOCK (BS object)

3.32 RESET SECURITY STATUS

Command RESET SECURITY STATUS

Function Reset the security status of the *current DF*

Standard / Use PCSC ICC Service Provider Part 6 / CardOS/M4.00: PCSC_Package, CardOS/M4.01: ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
80h	EAh	00h	MODE

Data

LC	Command Data Field	LE
*	empty	**

* Length of subsequent command data field
** Length of requested response data field

Response ./.

Description RESET SECURITY STATUS resets the security status of the *current DF* (i.e. withdraws all granted rights in the security status of the *current DF*) or the MF.

Table 3.32-1 shows the meaning of the MODE byte P2.

Table 3.32-1 Meaning of MODE byte P2

00h	Security status of the MF will be reset.
80h	Security status of the current DF will be reset.

Note that the deletion of all access rights in the smart card is only guaranteed, if the MF is the *current DF*.

Prerequisites The command is available for CardOS/M4.01 always and for CardOS/M4.00 only, if the PCSC_Package was loaded and activated before.

Options P3=LC, Tr=OFF, AC=none, SM=none

3.33 SELECT FILE

Command SELECT FILE

Function Select a file

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
00h	A4h	MODE	00h / 0Ch

Data

LC	Command Data Field	LE
	Command_Data (depending on MODE)	
*	(var. length)	**

* Length of subsequent command data field
** Length of requested response data field

Response

DATA		
FCI template		
TAG (T)	**Length (L)**	**Value (V)**
6Fh	*n*	*FCI data*

FCI data												
FCI 1			FCI 2			FCI 3 ... FCI 5	FCI 6			FCI 7 / empty		
T	L	V	T	L	V	...	T	L	V	T	L	V
For TLV-data of FCIs see table below												

n Length of subsequent FCI data field

Description The command selects a file using its file ID, its name (only DFs) or a path. The MODE parameter specifies how the file is selected (see table 3.32-1).

Table 3.32-1 MODE Byte P1

00h	Select DF or EF directly below the current DF using the file ID. [Special Case : Using an empty data field or file ID 3F00h then the MF is selected.]	File ID (0 or 2 bytes)
01h	Select DF directly below the current DF using the file ID.	File ID (2 bytes)
02h	Select EF directly below the current DF using the file ID.	File ID (2 bytes)
03h	Select parent-DF of current DF. If the MF is the current DF, then SELECT FILE returns an error code.	empty (0 bytes)
04h	Select a DF using its name (File access is performed using the complete file tree).	DF Name (1...16 bytes)
08h	Select a DF or EF using the path starting from the MF. The path specifies all file IDs from the MF to the file to be selected (file ID of the MF excluded, file ID of file to be selected included)	Path (m * 2 bytes)
09h	Select a DF or EF using the path starting from the current DF. The path specifies all file IDs from the current DF to the file to be selected (file ID of the current DF excluded, file ID of file to be selected included)	Path (m * 2 bytes)

If P2 = 00h, then SELECT FILE returns FCI data. In this case the LE byte can be used to specify the length of the response data according to ISO 7816-4. If P2 = 0Ch, then no response data are sent.

The FCI data returned by SELECT FILE are described in table 3.32-2.

Table 3.32-2 Description of FCIs for the SELECT FILE command

T	L	V (Value)	Description
81h	02h	EF: Net size in bytes DF: Size of largest available memory block	File Size
82h	02h	**BINARY and CODE files:** Byte 1: File Type: 01h: BINARY 21h: CODE Byte 2: Data coding byte (21h), don't care	File Type
	06h	**LINEAR FIXED, LINEAR TLV, CYCLIC FIXED files and DFs:** Byte 1: File Type: 02h: LINEAR FIXED 05h: LINEAR TLV 06h: CYCLIC FIXED 38h: DF Byte 2: Data coding byte (don't care): 21h Byte 3: Maximum record/object length (HI-byte): 00h Byte 4: Maximum record/object length (LO byte): FEh: for LINEAR TLV files and DFs length: for LINEAR FIXED and CYCLIC FIXED files Byte 5,6: Number of stored records/objects (HI-LO byte)	
83h	02h	File ID (16 bits)	File ID
84h	p	DF Name if defined (for DFs only)	DF Name

p: Length of DF name in bytes

Table 3.32-2 Description of FCIs for the SELECT FILE command (continued)

T	L	V (Value)	Description
85h	01h	**File status for EF:** Byte 1: 2^7 Deactivation flag: 1/0: invalid/valid 2^6 Deactivation Mode: 1: READ can be used for deactivated file 0: READ cannot be used for deactivated file (default) 2^5 Termination Flag: must be set to ‘0’ 1/0: terminated/usable 2^0 Checksum: 1: using header only 0: using header + body (default)	File Status
	03h	**File status for MF / DF:** Byte 1: 2^7 Deactivation flag: 1/0: invalid/valid 2^5 Termination Flag: 1/0: terminated/usable $2^{4(*)}$ DeathEnableBit: 1/0 permits the switch to life cycle phase DEATH 2^0 Checksum: don’t care bit Checksum is built over the header always Byte 2: HI byte of DF body size Byte 3: LO byte of DF body size	
86h	q	Definitions for Access Conditions	AC Definitions (see table 3.32-3)
8Bh	r	Definitions for Secure Messaging	SM Definitions (see table 3.32-4)

q:	Length of AC definitions in bytes
r:	Length of SM definitions in bytes
(*):	This behavior applies to CardOS/M4.01 always and to CardOS/M4.00, if the so-called UseCounter_Package was loaded and activated before.

The **DeathEnableBit** permits the switch to lifecycle phase DEATH if an object located at this or deeper level tries to perform it due the condition

- USECOUNT reaches zero
- the DeathBit of the object is set

Since this affects the whole card, every DF in the hierarchy and the MF can veto this option by letting this bit cleared. This behavior applies to CardOS/M4.01 always and to CardOS/M4.00, if the so-called UseCounter_Package was loaded and activated before.

Table 3.32-3 and 3.32-4 show the possible AC and SM definitions.

Table 3.32-3 AC Definitions in FCI 86h

Byte No.	AC definitions (possible values)	Meaning for MF	Meaning for DF	Meaning for EF
1	00h ... 7Fh, FFh	AC LCYCLE	AC LCYCLE	AC READ (Data)
2	00h ... 7Fh, FFh	AC UPDATE (Objects)	AC UPDATE (Objects)	AC UPDATE (Data)
3	00h ... 7Fh, FFh	AC APPEND (Objects)	AC APPEND (Objects)	AC APPEND (Record) / rfu
4	00h ... 7Fh, FFh	AC DEACTIVATE (MF)	AC DEACTIVATE (DF)	AC DEACTIVATE (EF)
5	00h ... 7Fh, FFh	AC ACTIVATE (MF)	AC ACTIVATE (DF)	AC ACTIVATE (EF)
6	00h ... 7Fh, FFh	AC ERASE (MF)	AC DELETE (DF)	AC DELETE (EF)
7	00h ... 7Fh, FFh	AC ADMIN (MF)	AC ADMIN (DF)	AC ADMIN (EF)
8	00h ... 7Fh, FFh	AC CREATE (Files)	AC CREATE (Files)	AC INCREASE / rfu
9	00h ... 7Fh, FFh	AC EXECUTE	rfu	AC DECREASE / rfu

Meaning of possible values:

00h:	Access action referenced by AC can **always** be performed (i.e. right is granted).
01h ... 7Fh	Access action referenced by AC can be performed, if right 01h ... 7Fh is granted.
FFh:	Access action referenced by AC can **never** be performed.

Table 3.32-4 SM Definitions in FCI 8Bh

Byte No.	SM definitions (possible values)	Meaning for MF	Meaning for DF	Meaning for EF
1	00h ... FFh	rfu	rfu	ENC READ-OUT (Data)
2	00h ... FFh	rfu	rfu	SIG READ-OUT (Data)
3	00h ... FFh	ENC UPDATE (0) / ENC UPDATE/APPEND (1)	ENC UPDATE (0) / ENC UPDATE/APPEND (1)	ENC UPDATE (Data)
4	00h ... FFh	SIG UPDATE (0) / SIG UPDATE/APPEND (1)	SIG UPDATE (0) / SIG UPDATE/APPEND (1)	SIG UPDATE (Data)
5	00h ... FFh	ENC APPEND (0) / rfu (1)	ENC APPEND (0) / rfu (1)	ENC APPEND (Record) / rfu
6	00h ... FFh	SIG APPEND (0) / rfu (1)	SIG APPEND (0) / rfu (1)	SIG APPEND (Record) / rfu
3	00h ... FFh	ENC UPDATE (Objects)	ENC UPDATE (Objects)	ENC UPDATE (Data)
4	00h ... FFh	SIG UPDATE (Objects)	SIG UPDATE (Objects)	SIG UPDATE (Data)
5	00h ... FFh	ENC APPEND (Objects)	ENC APPEND (Objects)	ENC APPEND (Record) / rfu
6	00h ... FFh	SIG APPEND (Objects)	SIG APPEND (Objects)	SIG APPEND (Record) / rfu
7	00h ... FFh	rfu	rfu	rfu
8	00h ... FFh	rfu	rfu	rfu
9	00h ... FFh	rfu	rfu	rfu
10	00h ... FFh	rfu	rfu	rfu
11	00h ... FFh	rfu	ENC DELETE (DF)	ENC DELETE (EF)
12	00h ... FFh	rfu	SIG DELETE (DF)	SIG DELETE (EF)
13	00h ... FFh	ENC ADMIN (MF)	ENC ADMIN (DF)	ENC ADMIN (EF)
14	00h ... FFh	SIG ADMIN (MF)	SIG ADMIN (DF)	SIG ADMIN (EF)
15	00h ... FFh	ENC CREATE (Files)	ENC CREATE (files)	ENC INCREASE-IN (Data) / rfu
16	00h ... FFh	SIG CREATE (Files)	SIG CREATE (files)	SIG INCREASE-IN (Data) / rfu
17	00h ... FFh	rfu	rfu	ENC DECREASE-IN (Data) / rfu
18	00h ... FFh	rfu	rfu	SIG DECREASE-IN (Data) / rfu
19	00h ... FFh	ENC GENKEY	ENC GENKEY	ENC INCREASE-OUT (Data) / rfu
20	00h ... FFh	SIG GENKEY	SIG GENKEY	SIG INCREASE-OUT (Data) / rfu
21	00h ... FFh	rfu	rfu	ENC DECREASE-OUT(Data)/ rfu
22	00h ... FFh	rfu	rfu	SIG DECREASE-OUT (Data) / rfu
23	00h ... FFh	rfu	rfu	ENC READ-IN (DATA) / rfu
24	00h ... FFh	rfu	rfu	SIG READ-IN (Data) / rfu

Meaning of possible values:

00h ... FEh	ID of SM-key to be used for referenced access action, ENC/SIG and communication direction.
FFh:	**No** SM.
(0):	CardOS/M4.00 (Objects)
(1):	CardOS/M4.01 (Objects)

Additionally to the selection of a file SELECT FILE searches for a Default SE object via backtracking, if the DF context is to be changed. If a Default SE object is found in the newly selected DF, then the data of the Default SE object are automatically copied to the CSE (current security Environment) (equivalent to the MSE (RESTORE) command with SE object ID = 01h).

Prerequisites ./.

Options P3=LC, Tr=ON, AC=none, SM=none

3.34 UNINSTALL PACKAGE

Command UNINSTALL PACKAGE

Function Uninstall a package of the smart card

Standard / Use Proprietary / CardOS/M4.00: Uninstall_Package, CardOS/M4.01: ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
80h / 84h	E4h	00h	00h

Data

LC	Command Data Field	LE
	Package_ID (PID)	
*	var. Length	./.

* Length of subsequent command data field

Response ./.

Description UNINSTALL PACKAGE removes a package referenced by its Package_ID from the smart card.

Table 3.34-1 and 3.34-2 show possible PIDs and packages to be uninstalled for CardOS/M4.00 and CardOS/M4.01.

Table 3.34-1 Possible PIDs for CardOS/M4.00

Package	Package_ID (PID)					FID
	T	L	V			
			Package-No	Version /	CardOS Version	
Delete_File_Package	01h	04h	01h	**01**h	C8h 02h	CF01h
Uninstall_Package	01h	04h	02h	**03**h	C8h 02h	CF02h
PCSC_Package	01h	04h	03h	**01**h	C8h 02h	CF03h
DSA_Package(0)	01h	04h	04h	**01**h	C8h 02h	CF04h
Generate_Key_Pair_Package	01h	04h	07h	**07**h	C8h 02h	CF07h
Manage_Channel_Package	01h	04h	08h	**02**h	C8h 02h	CF08h
LeadingZero_Package	01h	04h	0Bh	**01**h	C8h 02h	CF0Bh
UseCounter_Package	01h	04h	0Ch	**02**h	C8h 02h	CF0Ch
DSI_Extension_Package	01h	04h	0Dh	**01**h	C8h 02h	CF0Dh
Sign_By_Decryption_Key_Package	01h	04h	0Eh	**01**h	C8h 02h	CF0Eh
RSA_Extension_Package	00h	04h	10h	**01**h	C8h 02h	CF10h
Service_Package_01	01h	04h	11h	**01**h	C8h 02h	CF11h

(0): on request

Table 3.34-2 Possible PIDs for CardOS/M4.01

Package	Package_ID (PID)					FID
	T	L	V			
			Package-No.	Version	CardOS Version	
DSA_Package(1)	01h	04h	04h	**01**h	C8h 03h	CF04h
Sign_By_Decryption_Key_Package(1)	01h	04h	0Eh	**01**h	C8h 03h	CF0Eh
Service_Package_02	01h	04h	13h	**01**h	C8h 03h	CF13h

(1): on request

Note

A PID for a certain package changes each time a new release of this package is issued. It is strongly recommended to use the newest releases of the packages always.

For the newest information about packages refer to the newest releases of the *CardOS/M4.00 Packages & Release Notes Manual* respectively *CardOS/M4.01 Packages & Release Notes Manual*.

In combination with the MANAGE CHANNEL command and during execution the UNINSTALL PACKAGE command
- **must be performed in logical channel 00h** and
- **other logical channels must not be opened**.

This usage of the UNINSTALL PACKAGE command in combination with the MANAGE CHANNEL command
- is mandatory for CardOS/M4.01 and
- may otherwise lead to inconsistent data in a CardOS/M4.00 smart card. This problem for CardOS/M4.00 can be solved, if other security mechanisms of CardOS/M4 are used to prohibit other usages of UNINSTALL PACKAGE in combination with MANAGE CHANNEL as described above.

Prerequisites

The command is available for CardOS/M4.01 always and for CardOS/M4.00 only, if the Uninstall_Package was loaded and activated before. The command can only be executed, if the right referenced by the CODE file's *AC DELETE* is granted in the *security status* of the *current DF*.

Options

For application packages:
P3=LC, Tr=OFF, AC=DELETE (CODE file), SM=none

For system packages:
P3=LC, Tr=OFF, AC=none, SM=SM mode x4h (*StartKey*)

3.35 UPDATE BINARY

Command UPDATE BINARY

Function Update a *BINARY* file

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh	D6h	HI	LO

Data

LC	Command Data Field	LE
	Binary_Data	
*	(var. length)	./.

* Length of subsequent command data field

Response ./.

Description This command writes the Binary_Data into a *BINARY* file. The start offset at which the write access is to start is specified in HI-LO.

If bit 2^7 is set in HI, HI is considered as SFI for selection of the desired file (bits 2^0 ... 2^4). 00h is then assumed to be the high byte of the start offset (see table 3.34-1).

Table 3.34-1 HI Byte P1

7	6	5	4	3	2	1	0	¬ Bit Number / Meaning ¯
0	*x*	*x*	*x*	*x*	*x*	*x*	*x*	Use current EF and HI-value=*'0xxx xxxx'*
1	*y*	*y*	*p*	*p*	*p*	*p*	*p*	Use SFI '*p pppp*' *('p pppp' = '1 1111' not permitted)* and
								HI-value=00h; yy-Bits are don't care (shall be set to '00')

Using a SFI the FID to be used is built adding the constant value **FE00h** to the specified SFI.

Prerequisites The command can only be executed, if the right referenced by the file's *AC UPDATE* is granted in the *security status* of the *current DF*.
UPDATE BINARY can only be used on *BINARY* or *CODE* files

Options P3=LC, Tr=ON, AC=UPDATE (file), SM=UPDATE (file)

3.36 UPDATE RECORD

Command UPDATE RECORD

Function Overwrite an existing record.

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh	DCh	REC	EF-ID / MODE

Data

LC	Command Data Field	LE
	Record_Data	
*	(var. length)	./.

* Length of subsequent command data field

Response ./.

Description This command writes the Record_Data into a *LINEAR FIXED*, *CYCLIC FIXED* or *LINEAR TLV* file.

Bits 2^3 to 2^7 of parameter P2 specify the file to be used for the command (i.e. *EF-ID*). Bits 2^0 to 2^2 of parameter P2 (i.e. *MODE*) specify the access mode (see table 3.36-1)

Table 3.36-1 EF-ID / MODE Byte P2

7	6	5	4	3	2	1	0	¬ **Bit Number / Meaning** ¯
0	*0*	*0*	*0*	*0*				Use current EF
p	*p*	*p*	*p*	*p*				Use SFI *'p pppp' ('p pppp' = '1 1111' not permitted)*
					0	0	0	**FIRST** mode: Overwrite the first record REC contains the record identifier
					0	0	1	**LAST** mode: Overwrite the last record REC contains the record identifier
					0	1	0	**NEXT** mode: Overwrite the next record REC contains the record identifier
					0	1	1	**PREV** mode: Overwrite the previous record REC contains the record identifier
					1	0	0	**CURRENT / ABSOLUTE** mode: Overwrite the current record (REC=0) or overwrite via the record number (REC contains the logical number of the record)

Using a SFI the FID to be used is built adding the constant value **FE00h** to the specified SFI.

The modes *FIRST*, *LAST*, *NEXT* and *PREV* affect the internal record pointer which references the last record read. Using *LINEAR FIXED* and *CYCLIC FIXED* files *REC* = 00h is permitted only. Using *LINEAR TLV* files *REC* specifies the record identifier.

In mode *CURRENT*, either the last record referenced with one of the modes *FIRST*, *LAST*, *NEXT* or *PREV* is overwritten again (*REC* = 00h), or the record with the logical number specified in *REC* is overwritten.

With the exception of *CYCLIC FIXED* files, when a file is selected again, *NEXT* overwrites the first record and *PREV* overwrites the last record. *CURRENT* will cause an error. Using *CYCLIC FIXED* files the first record is selected as *current record*.

Using *LINEAR TLV* files UPDATE RECORD has the ability to overwrite records even if the record lengths of the new and the old record are different. In this case the records will be shifted inside the file.

When *LINEAR TLV* files are involved it is assumed that simple-TLV records (i.e. the length field is one byte in length) are used. The TAG byte will not be interpreted by UPDATE RECORD.

Using *CYCLIC FIXED* files *PREV* mode will work like APPEND RECORD, if a new record can be created inside the file. Otherwise, it will work like *LAST* mode. In all cases the new record will become the logically first record and the *current record*.

Prerequisites The command can only be executed, if the right referenced by the file's *AC UPDATE* is granted in the *security status* of the *current DF*.
UPDATE RECORD can only be used on *LINEAR FIXED*, *CYCLIC FIXED* or *LINEAR TLV* files.

Specials As an addition to the ISO 7816-4 standard a record identifier can be specified for the modes *FIRST*, *LAST*, *NEXT* and *PREV*.

Options P3=LC, Tr=ON, AC=UPDATE (file), SM=UPDATE (file)

3.37 VERIFY

Command VERIFY

Function Perform a PIN test (CHV test)

Standard / Use ISO 7816-4 / ROM-masked; *ADMINISTRATION, OPERATIONAL*

Parameters

CLA	INS	P1	P2
0xh	20h	00h	ID

Data

LC	Command Data Field	LE
	PIN_Data	
*	(var. length)	./.

* Length of subsequent command data field

Response ./.

Description The command performs a PIN test using the data given as PIN_Data.

Table 3.37-1 shows the ID Byte P2

Table 3.37-1 ID Byte P2

7	6	5	4	3	2	1	0	¬ **Bit Number / Meaning** ¯
z								z=0: Object search in MF
								z=1: Object search starting in current DF (backtracking)
	v	v	v	v	v	v	v	Object ID ‘*vvv vvvv*’:
								permitted values: ’000 0001’... ’111 1111’

The range of the object IDs is limited because the belonging PIN TEST object (i.e. TEST object using ALGO = PIN (87h)) is always referencing exactly one access right in the *security status* of the belonging DF (see PUT DATA_OCI command also). Each DF supports the access rights 01h ... 7Fh (i.e. ‘0000 0001’ ... ‘0111 1111’).

If P2=00h, then CardOS/M4 “knows” which object shall be used. For this the TEST object ID referenced in the CSE is used. The backtracking mechanism for searching the TEST object ID starts from the *current DF*.

If the PIN_Data are checked successfully by VERIFY, the referenced access right is granted and the CurrentErrorCounter of the PIN TEST object is set to its maximum (i.e. MaxErrorCounter) (if defined so in the TEST object; see PUT DATA_OCI).

If the PIN_Data are **not** checked successfully, the referenced access right will be withdrawn (depending on the p bit in the *OPTIONS* byte of the TEST object; see PUT DATA_OCI); in addition the CurrentErrorCounter of the PIN TEST object is decreased

Depending on the definition in the PIN TEST object the UseCounter in the PIN TEST object is decreased (see PUT DATA_OCI).

The access right will be granted or withdrawn in the DF, where the PIN TEST object is stored, and in all child-DFs.

Prerequisites The command can only be executed, if a PIN TEST object (i.e. the object usage bits *‘uuu’* of the object are *‘000’*) with the object ID specified in P2 exists (according to the backtracking mechanism of CardOS/M4) **and** the PIN TEST object is **not blocked** (see UseCounter and CurrentErrorCounter of objects) **and** the right referenced by the PIN TEST object’s *AC USE* is granted in the *security status* of the *current DF*.

Specials According to ISO 7816-4 the P2 (object ID) bits 2^5 ... 2^6 shall be set to ‘00’. For future compatibility it is recommended to take this restriction into account when defining key IDs.

Even if the CurrentErrorCounter of the TEST object reaches 0 the status code 6300h will be returned. If the CurrentErrorCounter=0 and a VERIFY command is executed the status code 6983h will be returned.

Options P3=LC, Tr=ON, AC=USE (PIN TEST object referenced by object ID), SM=USE-IN (PIN TEST object referenced by object ID)

3.38 Status Codes

Table 3.38-1 shows the SW1/SW2 status codes which are returned by the commands.

Table 3.38-1 SW1/SW2 status codes of CardOS/M4.0

SW1	SW2	Meaning
61h	xxh	*xx* bytes of response data can be received using GET RESPONSE (only T=0 transmission protocol)
62h	83h	File is deactivated (see DEACTIVATE FILE command)
	85h	File is terminated
63h	00h	Authentication failed
65h	81h	EEPROM error; command aborted
67h	00h	LC invalid
68h	81h	Logical channel not supported
	82h	SM mode unsupported
	84h	Chaining Error
69h	81h	Command cannot be used for file structure
	82h	Required access right not granted
	83h	BS object blocked
	84h	BS object has invalid format
	85h	No random number available
	86h	no *current EF* selected
	87h	Key object for SM not found
	88h	Key object used for SM has invalid format
6Ah	80h	Invalid parameters in data field
	81h	Function / mode not supported
	82h	File not found
	83h	Record / object not found
	84h	Not enough memory in file / in file system (e.g. CREATE FILE) available
	85h	LC does not fit the TLV structure of the data field
	86h	P1 / P2 invalid
	87h	LC does not fit P1 /P2
	88h	Object not found (GET DATA)
6Ch	00h	LE does not fit the data to be sent (e.g. SM)
6Dh	00h	INS invalid
6Eh	00h	CLA invalid (Hi nibble)
6Fh	00h	Technical Error: 1 It was tried to create more than 254 records in a file. 2 Package uses SDK version which is not compatible to API version 3 Package contains invalid statements (LOAD EXECUTABLE)
	81h	File is invalidated because of checksum error (prop.)
	82h	Not enough memory available in XRAM
	83h	Transaction error (i.e. command must not be used in a transaction)
	84h	General protection fault (prop.)
	85h	Internal failure of PK-API (e.g. wrong CCMS format)
	86h	Key Object not found
	87h	Internal hardware attack reqiured
	FFh	Internal assertion (invalid internal error) This error is no runtime error but an internal error which can occur because of a programming error only.
90h	00h	Command executed correctly
	01h	Command executed correctly; EEPROM weakness detected (EEPROM written with second trial; the EEPROM area overwritten has a limited life time only.)
98h	50h	Overflow using INCREASE / underflow using DECREASE