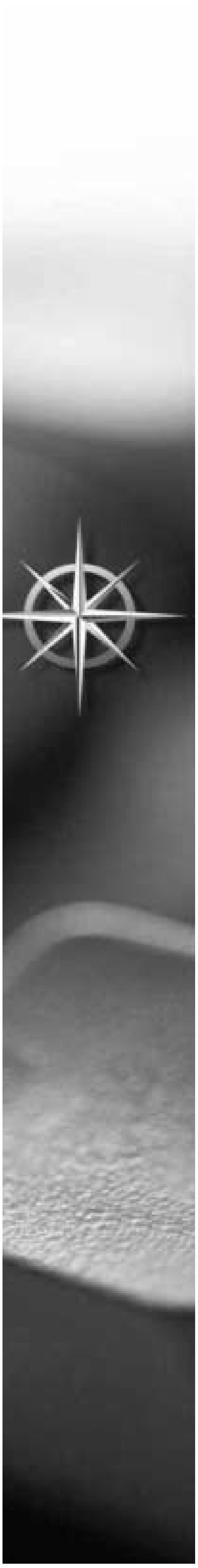


MPCOS-EMV R4

Reference Manual

Version 2.1

GEMPLUS

July 2001

Printed in France.

GEMPLUS, B.P. 100, 13881 GEMENOS CEDEX, FRANCE.
Tel: +33 (0)4.42.36.50.00 Fax: +33 (0)4.42.36.50.90

Document Reference: DPD15141B1

Contents

List of Figures

List of Tables

Preface

This document provides information about the MPCOS-EMV operating system and its range of cards. It provides detailed description of the MPCOS-EMV data structure, security system and command format.

Audience

This document is intended for users who want:

- To understand how the MPCOS-EMV operating system works
- To develop applications for the MPCOS-EMV cards

It assumes you have some basic knowledge on the following subjects:

- Smart card technology
- Cryptography

Conventions

The following conventions are used throughout this document.

- By default, a numeric value is expressed in decimal notation.
- Whenever a value is expressed in binary, it is followed by the b character. For example, the decimal value 13 expressed in binary becomes **1101b.**
- The h character follows a hexadecimal number. For example the decimal value 13 expressed in hexadecimal becomes **0Dh.**
- The value 00h is assigned to each RFU byte.

Bit Numbering

A **byte B** consists of eight bits: $b_7b_6b_5b_4b_3b_2b_1b_0$
b_7 is the most significant bit and b_0 the least significant bit:

One Byte	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

A **string of bytes** consists of n concatenated bytes: $B_nB_{n-1}....B_1B_0$
B_{n-1} is the most significant byte and B_0 the least significant byte:

A string of n bytes	B_{n-1}	B_{n-2}	—	—	B_3	B_2	B_1	B_0

Reserved for Future Use

Reserved for Future Use (RFU). When RFU bytes are included in a command, the value 0 (or 00h) should be used.

3DES_16

This notation is used to indicate that 3DES is being used to diversify a 16-byte 3DES key, instead of merely running its value through the 3DES algorithm. The key diversification process is indicated as follows:
Diversified Key = 3DES_16 [Data]< Key>

A 3DES_16 process gives a 16-byte result, while a 3DES process gives an eight-byte result.

For further details on key diversification, see *“Key Diversification”* on page 36.

For More Information

You should also have access to the following documentation:

- *ISO 7816-3—Identification Cards, Part 3: Electronic Signals and Transmission Protocols*
- *ISO 7816-4— Identification Cards, Part 4: Inter-Industry Commands for Interchange*
- *EMV—IC Card Specifications for Payment Systems, Parts 1, 2, 3*
- *Common Personalization for Visa Smart Debit and Credit (VSDC), version 1.1*
- *Visa Integrated Circuit Card: Card Specifications, version 1.3.2*

Contact Details

We welcome your feedback to help us provide high-quality documentation. Contact us at:

E-mail	TOLDsing@gemplus.com
Postal	Technical & Online Documentation (TOLD) GEMPLUS 12 Ayer Rajah Crescent Singapore 139942 SINGAPORE

Please quote the document reference, your job function and your company.

1 Introduction

Product Introduction

The Multi-application Payment Chip Operating System-EMV[1] (MPCOS-EMV) is an operating system designed for multi-purpose and payment applications.

The MPCOS-EMV features and applicable standards include:

- ISO 7816-1, -2, -3 compliance
  The basic communication protocol is T = 0. MPCOS-EMV also supports protocols T = 1 and T = 14.
- ISO 7816-4 compliance
  The compliant commands, data structures (multi-application) and return codes ensure a wide acceptance of the operating system by application issuers and terminal manufacturers.
- ISO 7816-5 compliance
  The Gemplus Registration IDentifier (RID) can be implemented. Application IDentifier (AID) is used to address an application in the card.
- EMV compliance
  This compliance allows implementation of VSDC Template 1 (magnetic stripe image applications in the smart card according to VISA specifications).
- I/O routines
  Up to 115,200 baud and fast cryptographic routines.
- Dedicated security features
  - Secret code/key management and verification
  - Sensitive system data protection
  - 3DES[2] algorithm for authentication, secure messaging and payment certification
  - Application-level countermeasures against DPA[3]
- Administration command set
  An enhanced administrative command set is available for easy card personalization.
- Payment command set
  A command set for electronic purse structure, payment-dedicated command set (for example, debit, credit and read balance).
- Customization features
  This provides the required flexibility to develop a wide range of applications.

1. Europay, Mastercard, Visa common specifications for chip debit/credit applications.
2. Triple Data Encryption Standard, the de-facto cryptographic standard for symmetric secret algorithms in the banking industry.
3. Differential Power Analysis, a process of forcing the card operating system to repeatedly perform cryptographic computations while collecting power signatures from the power line.

Product Range

"Table 1 - Range of MPCOS-EMV Cards" shows the different MPCOS-EMV cards available.

Product Range	Size
MPCOS-EMV R4 2000	2 kilobytes
MPCOS-EMV R4 8000	8 kilobytes
MPCOS-EMV R4 32000	32 kilobytes

Table 1 - Range of MPCOS-EMV Cards

Note: The available memory is less than the chip memory to accommodate the creation of initial mapping.

Data Structure

The MPCOS-EMV operating system is based on specific file structures. Files are organized into a two-level hierarchy—the global level and local level. The Master File (MF) occupies the top (global) level of the hierarchy. The level formed by the MF with Elementary Files (EF)s directly beneath it is called the *global level.* The level formed by Dedicated Files (DF)s with EFs beneath them is called the *local level.* A DF can also hold several EFs. Each EF contains data.

For more information on how MPCOS-EMV file and data are structured, see *"File and Data Structure"* on page 5.

Data Access Management

Access to MPCOS-EMV files is protected by secret codes. The secret codes are stored in a specific EF that are called Secret Code Elementary File (EF_{sc}). Each EF_{sc} can store up to eight secret codes, numbered from 0 to 7. The MF and each DF can hold one EF_{sc}.

Access conditions define the level of protection for a file. They are two-byte registers stored in DF and EF descriptors. They define the following:

- The number of secret codes (none, one or two) that have to be submitted to access the file for each access type (create, read, write or update).
- The cryptographic key used to generate session keys for all secure messaging[4] with the file.

The presentation of secret codes can be imposed prior to selecting a file. The status of successfully submitted secret codes are stored in an authorization register. The card then determines if access to a file should be granted by comparing the secret codes in the authorization register with that required by the access condition.

Secure messaging is a cryptographic process that secures data transmissions with MPCOS-EMV cards.

For more information on accessing MPCOS-EMV files, see *"Access Conditions"* on page 29.

Security

The MPCOS-EMV operating system includes commands that implement cryptographic functions to form a complete security scheme for the application. This security scheme involves authentication, calculation of the temporary/session key, certificate generation, signatures and secure messaging.

The MPCOS-EMV operating system has also incorporated the latest application-level protection against DPA attacks. For more information on the MPCOS-EMV security features, see *"Cryptography"* on page 35.

Command Set

There are three classes of MPCOS-EMV commands—administrative, payment, and EMV. The administrative commands include functions for creating a file, reading a record and updating a record. The payment commands include functions for crediting a purse, debiting a purse and reading a purse balance.

The MPCOS-EMV commands are listed individually and in detail in *"Chapter 9 - Commands"*.

4. Secure messaging is a cryptographic process that secures data transmission with MPCOS-EMV cards.

Communication

MPCOS-EMV cards send and receive data using the T = 0 communication protocol format that is in accordance with the ISO 7816-3 standard. MPCOS-EMV cards also supports T = 1 and T = 14 communication protocols. Check with a Gemplus technical consultant for more details on using MPCOS-EMV cards with T = 14 protocol.

In addition, the MPCOS-EMV operating system has high-speed I/O routines that can be activated using the **Switch Protocol** command. For more information on this command, see *"Switch Protocol (SwtPrt)"* on page 134.

Custom Configuration

As the MPCOS-EMV card is adapted to multi-purpose and payment applications, it can be customized to meet specific requirements. Contact a Gemplus sales representative for more information about application customization.

2

File and Data Structure

The files in MPCOS-EMV cards are organized in a two-level hierarchy.

The level formed by the Master File (MF) with Elementary Files (EFs) directly beneath it is called the *global level.* The level formed by Dedicated Files (DFs) with EFs beneath them is called the *local level.*

"Figure 1 - Hierarchy of MPCOS-EMV Files" shows the structure of the MPCOS-EMV file hierarchy.

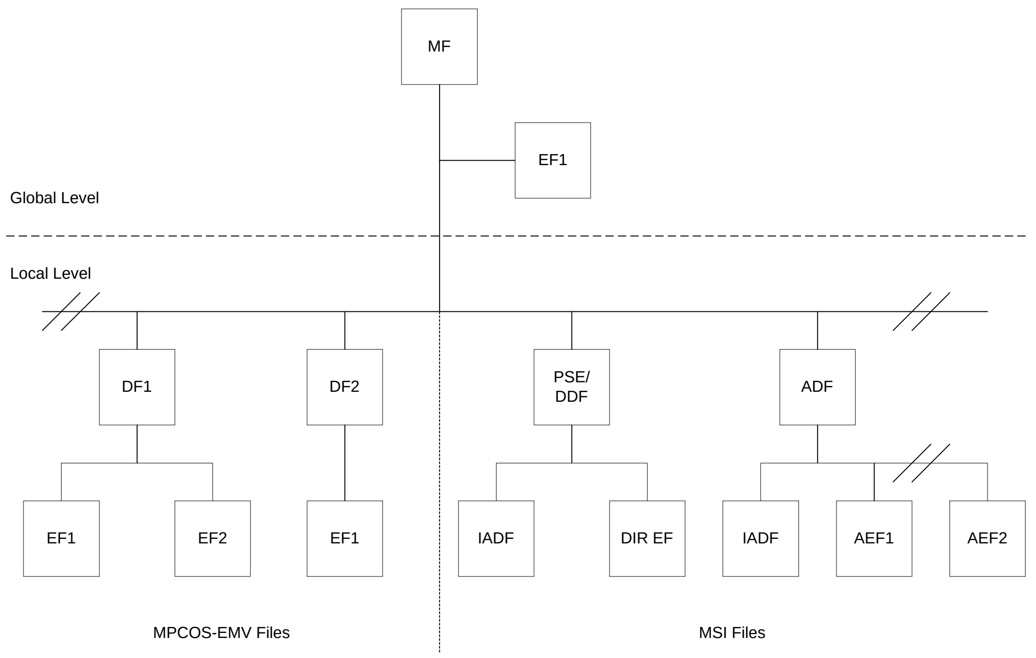


Figure 1 - Hierarchy of MPCOS-EMV Files

Master File

The Master File (MF) is the root of the MPCOS-EMV file structure. It is the equivalent of the root directory in the operating system of a computer. The MF (only one per card) can store up to 63 dedicated files and elementary files.

Dedicated Files

Dedicated Files (DFs) store sets of elementary files. In MPCOS-EMV cards, each DF stores the set of elementary files that form an application. A DF is the equivalent of a directory in Microsoft DOS. Each DF can store up to 63 elementary files, but no nested DFs are allowed.

Each MPCOS-EMV dedicated file consists of a 16-byte file descriptor and a file body. The file descriptor stores the information needed by the MPCOS-EMV operating system to manage the file. The file body holds the DF's name.

File Descriptor

When you create a file, the **Create File** command generates the file descriptor. *"Figure 2 - Structure of Dedicated File Descriptor"* shows how the file descriptor of a DF is structured.

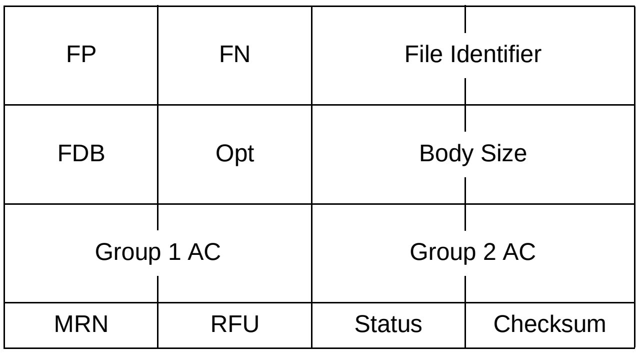


Figure 2 - Structure of Dedicated File Descriptor

The following describes the structure of the file descriptor of a dedicated file:

FP	1 byte	*File Pedigree*. This byte is used by the operating system to locate a file within the hierarchy. It specifies the descriptor type in bit F (1 for DF), the file hierarchy level in bit L level (1 for local level) and its parent's file number (that is, the master file):

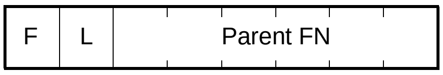


Figure 3 - Structure of FP in Dedicated Files

FN	1 byte	*File Number*. This is the sequential creation number of a file. Coded on six bits, it allows the creation of up to 63 EFs (0h value is not allowed).

Note: The FP and FN combination is unique throughout the memory and is used by the operating system to locate a given file.

File Identifier	2 bytes	This is allocated when the file is created. Each file is identified by a unique 16-bit number within a DF. The file can also be referenced by a Short File Identifier (SFI), which corresponds to the five least significant bits of the file identifier.
FDB	1 byte	*File Descriptor Byte.* When a DF is created, this byte is set to the following (38h):

Bit	7	6	5	4	3	2	1	0
	0	0	1	1	1	0	0	0

Figure 4 - Structure of a File Descriptor Byte

Opt	1 byte	*File Option Byte.* This is only used in a DF and it has the following format:

b7	b6	b5	b4	b3	b2	b1	b0

Figure 5 - Structure of a File Option Byte

"Table 2 - Dedicated File Options Byte Values" shows the various options indicated by each bit.

Bit	Value	Option Indicated
0		RFU
1	1	**Cancel Debit** command disabled.
	0	**Cancel Debit** command enabled.
2	1	Current balance can be used to compute sign certificates provided this option is not reversed using **Set Options** (see *"Set Options (SetOpts)"* on page 126).
	0	Current balance can never be used to compute Sign certificates (see *"Sign"* on page 131).
3–6		RFU
7	1	**Select Purse & Key** and **Select File Key** commands require external authentication.
	0	**Select Purse & Key** and **Select File Key** commands do not need an external authentication.

Table 2 - Dedicated File Options Byte Values

Body Size	2 bytes	This specifies the size of the file body. In a DF, the file body usually contains the DF name (up to 16 bytes). The memory is allocated at the byte boundary.
Group 1/2 AC	2 bytes	*Group 1 and 2 Access Conditions.* This defines the access conditions assigned to the DF. For more information on access conditions, see *"Access Conditions"* on page 29.
MRN	1 byte	*Maximum Replay Number.* The maximum number of times a session key can be used in any session. This byte holds the initial value of the MRN counter. For more information on the MRN counter mechanism, see *"Maximum Replay Number (MRN) Mechanism"* on page 46.
Status	1 byte	Reserved
Checksum	1 byte	This is the sum of the number of bits that are set to zero in the other 15 bytes of the file descriptor.

File Body

A file body stores an optional name in DFs. This name can be up to 16 bytes in length and it can be used by the **Select File** command to select a DF. DFs do not have a structure.

Selection by Partial Name

The MPCOS-EMV operating system allows DF to be selected by partial name. For example, a DF called "Gemplus_EP" will be selected even if only "Gemplus" is specified as the selection criterion in the **Select File** command.

Note: If more than one DF satisfies the selection criterion, the **Select File** command will select the DF that was created first. In the above example, if another DF called "Gemplus" exists, then running the **Select File** command with "Gemplus" as the selection criteria will not select the "Gemplus" file if "Gemplus_EP" was created first.

However, to select the next DF with the same partial name, use the **Select File** command with P1 = 04h.

Referencing by AID

Application Definition Files (ADFs) and Directory Definition Files (DDFs) may be referenced by an Application Identifier (AID), coded on 5 to 16 bytes, as follows:

AID	
RID	PIX
5 bytes	up to 11 bytes

Where:

RID	Registered application provider IDentifier. A mandatory field of 5 bytes identifying the payment system.
PIX	Proprietary application Identifier eXtension. An optional field of up to 11 bytes assigned by the application provider to identify the application within a payment system.

Note: The RID must be registered with ISO, per ISO 7816-5 specifications. For ease of implementation, Gemplus RID is offered for use by customers. Please contact the nearest Gemplus sales office for assistance.

DF Types

Payment Systems Environment (PSE)

The Payment System Environment (PSE) is a DF that is mandatory for EMV compliance. It begins with a Directory Definition File (DDF) with the name ’1PAY.SYS.DDF01’. Under this DDF is a directory file known as Payment System Directory.

The PSE is referenced by its name ’1PAY.SYS.DDF01’.

Directory Definition File

The Directory Definition File (DDF) is a DF which defines the applications directory structure beneath it. Its File Control Information (FCI) is contained in an IADF EF.

DDFs are referenced using the **Select File** command. See *“Selection by Partial Name”* on page 8.

Application Definition File

The Application Definition File (ADF) is the entry point to the Application Elementary Files (AEFs). The File Control Information (FCI) or the Answer To Select to this ADF is contained in an IADF EF.

ADFs are referenced using the **Select File** command. See *“Selection by Partial Name”* on page 8.

Elementary Files

Elementary files are the main component of the MPCOS-EMV file structure. They store application data.

Each MPCOS-EMV elementary file consists of a 16-byte file descriptor and a file body. The file descriptor stores the information needed by the MPCOS-EMV operating system to manage the file. The file body holds the data.

File Descriptor

When you create a file, the **Create File** command generates the file descriptor. *"Figure 6 - Structure of Elementary File Descriptor"* shows how an EF descriptor is structured.

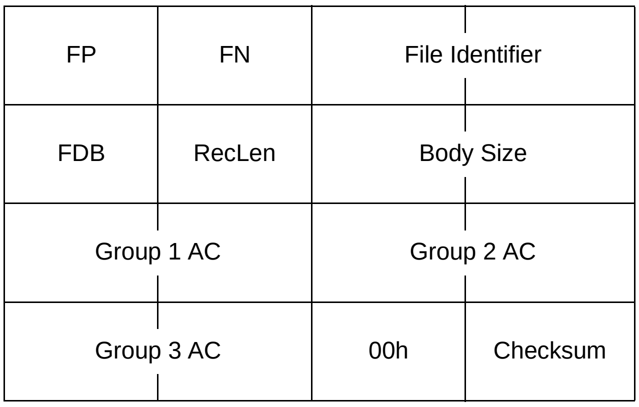


Figure 6 - Structure of Elementary File Descriptor

The following describes the structure of the file descriptor for an elementary file:

FP	1 byte	*File Pedigree.* This byte is used by the operating system to locate a file within the hierarchy. It specifies the descriptor type in bit F (0 = EF), the file hierarchy level in bit L level (0 for global level and 1 for local level) and its parent's file number:

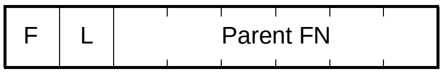


Figure 7 - Structure of FP in Dedicated Files

FN	1 byte	*File Number.* This is the sequential creation number of a file. Coded on six bits, it allows the creation of up to 63 EFs (0h value is not allowed).

Note: The FP and FN combination is unique throughout the memory and is used by the operating system to locate a given file.

File Identifier	2 bytes	This is allocated when the file is created. The Short File Identifier (SFI) corresponds to the five least significant bits of the file identifier.
FDB	1 byte	*File Descriptor Byte.* This contains information about the EF type and structure. This byte takes the format shown in *"Table 3 - Elementary File Descriptor Byte Format"*:

b7	b6	b5	b4	b3	b2	b1	b0	Description
0	0							File not shareable.
		x	x	x				**Type of EF**
		1	0	1				Key EF
		1	0	0				Secret Code EF
		0	1	1				Purse EF
		0	1	0				Transaction Manager EF
		0	0	1				Internal Application Data File (IADF)
		0	0	0				EF containing data not interpreted by the card (data file)
					x	x	x	**Structure of EF**
					0	0	0	No information recovered.
					0	0	1	Transparent
					0	1	0	Linear fixed, no further information.
					0	1	1	Linear fixed, simple Tag Length Value (TLV).
					1	0	0	Linear variable, no further information.
					1	0	1	Linear variable, simple TLV.
					1	1	0	Cyclic, no further information.
					1	1	1	Cyclic, simple TLV.

Table 3 - Elementary File Descriptor Byte Format

The different types of EFs listed in *"Table 3 - Elementary File Descriptor Byte Format"* are described later in this chapter.

b7 is normally set to 0 (the operating system does not allow the creation of a file if this bit is set to 1).

When creating an EF, the operating system does not check the contents of the FDB (except for the two most significant bits), so more than one of each type of file can be created. However, some EFs are assumed to be unique in a given DF (for example, secret code files and IADF). These files are accessed internally by MPCOS-EMV using their FDB, so if more than one of these files is created, only the first one will be recognized.

RecLen	1 byte	This field contains the record length for linear files with a fixed record length (for example, linear fixed files and cyclic files). It has no value for other file types.

Body Size	2 bytes	This specifies the size of the file body. In an EF, the file body contains the data. The memory allocation always corresponds to a word boundary. If the body size is not a multiple of four, the file will be padded with additional (and inaccessible) bytes.
Group 1/2/3 AC	2 bytes	*Group 1, 2 and 3 Access Conditions.* This defines the access conditions assigned to the EF. For more information on access conditions, see *"Access Conditions"* on pag e29.
Checksum	1 byte	This is the checksum of the number of bits that are set to zero in the other 15 bytes of the file descriptor. The checksum is checked during any file selection in order to verify the integrity of the descriptor in case of a memory failure.

File Body

The file body of an EF stores data. There are six types of EFs in MPCOS-EMV cards, as follows:

- Purse files
- Enhanced purse files
- Key files
- Transaction manager files
- Secret code files
- Internal Application Data Files (IADF)

EF Types

Each file type has a pre-defined body structure.

Purse Files

FDB value: 00011001b or 19h.

MPCOS-EMV purse files contain one purse only. Each dedicated file can hold up to 32 purse files. These files must all be among the first 32 files created in a dedicated file.

Purse files are specified by their short file identifier (see *“File Body”* on page 12). Purses are structured as follows:

				Offset (in words)
Maximum Balance			Credit Key (Kc) File	0
Maximum Free Debit Value			Access: Cd / Cb	1
Current (Active) Balance			Cks 1	2
Backup Balance			Cks 2	3
Terminal Transaction Counter (TTC)		RFU		4

Figure 8 - Structure of a Purse

Where:

Maximum Balance	3 bytes	The maximum balance that the purse can hold.
Credit Key File	5 bits	Specifies the short file identifier of the file holding the purse credit key. This file must be stored in the same dedicated file as the purse file itself.
Maximum Free Debit	3 bytes	The maximum value that can be debited from the purse when the debit access condition has not been fulfilled. If this value is set to 0h, the debit access condition must be fulfilled for all debits.
Cd	1 nibble	The access condition for a debit operation. Values of 1–7 specify the secret code number that must be submitted before the purse can be accessed. Values greater than 7 indicates the purse cannot be accessed for that function. A zero value indicates unrestricted access to the purse for that function.
Cb	1 nibble	The access condition to read the purse balance. Values of 1–7 specify the secret code number that must be submitted before the purse can be accessed. Values greater than 7 indicates the purse cannot be accessed for that function. A zero value indicates unrestricted access to the purse for that function.
Current Balance	3 bytes	The current balance value of the purse.
Backup Balance	3 bytes	The previous balance of the purse (that is, before the last transaction was carried out). MPCOS-EMV can use this value to restore the purse balance after any incorrect purse updates.
Cks1	1 byte	Checksum of the current register, obtained by performing an XOR operation on the first three bytes of each word and inverting the final result.

Cks2	1 byte	Checksum of the backup register, obtained by performing an XOR operation on the first three bytes of each word and inverting the final result.
Terminal Transaction Counter (TTC)	2 bytes	Contains the TTC’s two most significant bytes while the debit operation is being processed. This field is used to identify which terminal performed the last debit operation and it is checked by MPCOS-EMV before any **Cancel Debit** operation.

Enhanced Purse Files

FDB value: 00011001b or 19h.

MPCOS-EMV purses can be enhanced to include additional functions. Enhanced purses include an extra word at the Offset 5 position. The extra word can be used to protect the **Credit** operation with a secret code. It can also be used to specify the hierarchical level of the access conditions for the **Read Balance, Debit** and **Credit** operations. That is, whether the secret codes are to be read from the global EF_{sc} or the local EF_{sc} (see *L* below).

The extra word has the following format:

RFU	RFU	RFU	000L	CAC

Figure 9 - Extra Word Format in Enhanced Purse File

Where:

RFU	Reserved for Future Use.	
L	Coded on a single bit. Defines the hierarchical level of the EF_{sc} file for the **Read Balance, Debit** and **Credit** access conditions. It is set to 0 or 1:	
	0	*Global.* The secret codes are contained in the EF_{sc} of the master file.
	1	*Local.* The secret codes are contained in the EF_{sc} of the currently selected dedicated file.
CAC	Credit Access Condition (1 nibble) defined as follows	
	0000b	No secret code protection for **Credit** operations.
	0xxxb	**Credit** operations are protected by secret code number xxx from either the current EF_{sc} of the selected dedicated file or the EF_{sc} of the master file (see ***L*** above).
	1xxxb	This purse cannot be credited.

Key Files

FDB value: 00101001b or 29h.

Key files store the cryptographic keys used in all MPCOS-EMV cryptographic functions.

The master file and each dedicated file can store one or more key files. Each key file can store up to four 3DES_16 keys.

MPCOS-EMV uses different types of cryptographic keys as follows:

Administration Keys	Used for the computation of temporary administration keys and secure messaging.
Payment Keys	Used for payment commands such as transaction certificate generation and the computation of temporary certification keys. It is recommended that different payment keys be assigned to the main payment functions: for example, the key used for **Credit** operations is different from that used for **Debit** operations.
Log Keys	Also known as Multipurpose Keys. Used to initiate a payment session (see *“Select Purse & Key (SelPK)”* on page 122) but not an administration session (see *“Select File Key (SelFk)”* on page 119). However, a temporary key derived from a log key can be used for administration actions such as secure messaging.
Signature Keys	Payment keys dedicated to the computation of signatures. These keys cannot be used to compute temporary certification keys and they cannot be used by the **Select Purse & Key** command.
Authentication Keys	Used for authentication commands. These keys cannot be used by the **Select Purse & Key** and **Select File Key** commands.

The commands in *“Table 4 - Commands that Require Cryptographic Key”* require key files, as follows:

	SelP&K	SelFk	Credit /Debit	Sign	External/ Internal Authentication	Secure Messaging
Administration		✔				✔
Payment	✔		✔	✔		
Log	✔		✔	✔		✔
Authentication					✔	
Signature			✔	✔		

Table 4 - Commands that Require Cryptographic Key

Different keys in a given file must be updated separately.

3DES_16 cryptographic keys are stored as two consecutive standard DES keys in a single key file. That is, they take up 16 bytes plus an additional eight bytes for the two key headers.

3DES_16 keys are stored in key files as follows:

Key Type Part 1	Current ONC	Kv	Cks
K15	K14	K13	K12
K11	K10	K9	K8
Key Type Part 2	Backup ONC	Kv	Cks
K7	K6	K5	K4
K3	K2	K1	K0

Figure 10 - 3DES_16 Key Record Structure

Where:

Key Type Part 1	1 byte	Key type for the first part of the 3DES_16 key, coded on one byte. Both parts of the 3DES_16 key must be of the same key type. The possible key types are as follows:

b7	b6	b5	b4	b3	b2	b1	b0	Key Types
0								ONC disabled
1								ONC enabled
	x	x	0	x	x	0	0	Administration Key
	x	x	0	x	x	0	1	Payment Key
	x	x	1	x	x	0	1	Signature Key
	x	x	0	x	x	1	0	Log Key
	x	x	0	x	x	1	1	Authentication Key

Current ONC	1 byte	The current Off-Nominal Counter (ONC) that tracks the use of commands which require permanent keys. The ONC associated with the permanent key will decrement when an incoming cryptogram verification fails or when an outgoing cryptogram is generated. When the counter decrements to zero, the key will be disabled. For more information on the ONC, see *“Off-Nominal Counter (ONC) Mechanism”* on page 45.
Kv	1 byte	Key version number. If a terminal updates a key, it can write a key version number to this field. MPCOS-EMV does not manage this field. The default value is 0h.
Cks	1 byte	Checksum obtained by performing an XOR operation on the 10 other bytes of the key part and inverting the final result.

K15 to K8	1 byte each	Stores the key values for the first part of the 3DES_16 key.
Key Type Part 2	1 byte	Key type for the second part of the 3DES_16 key, coded on one byte. The key must be of the same type as the first part of the 3DES_16 key.
Backup ONC	1 byte	The ONC that was current before the previous command was executed.
K7 to K0	1 byte each	Stores the key values for the second part of the 3DES_16 key.

Warning: For all applications, it is essential that you lock read access to key files.

3DES_16 keys are stored as two consecutive DES keys, the actual numbers for each of the above keys are given below:

Key	Key Number
First	0
Second	2
Third	4
Fourth	6

Table 5 - Key Number

Because each key is double-mapped, the key numbers must be an even number.

Warning: The selection of odd key numbers will be rejected by the operating system and an error status will be returned.

Transaction Manager Files

FDB value: 00010001b or 11h.

Each dedicated file holding purse files must also hold a transaction manager file in order to recognize payment commands. The transaction manager file is a transparent EF and is eight bytes in length. The master file and each dedicated file can hold only one transaction manager file.

Transaction manager files have the following structure:

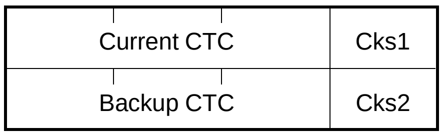


Figure 11 - Data Structure of a Transaction Manager File

The transaction manager file contains the following:

Current CTC	The Card Transaction Counter (CTC) is a three-byte counter that is incremented every time a payment transaction session is established (for example, each time the **Select Purse & Key** command is used). The CTC is used as a variable element for payment-oriented cryptographic processing. The current CTC is stores the card transaction counter value for the parent DF after the last transaction was completed.
Cks	Checksum obtained by performing an XOR operation on the first three bytes of each word of the current CTC and inverting the final result.
Backup CTC	Stores the value of the card transaction counter that was current before the last transaction was executed.
Cks2	Checksum obtained by performing an XOR operation on the first three bytes of each word of the backup CTC and inverting the final result.

Warning: The access conditions for updating and writing to transaction manager files must be locked.

Secret Code Files

FDB value: 00100001b or 21h.

A secret code file is a transparent EF. The master file and each dedicated file can only have one Secret Code Elementary File (EF_{SC}). Only the first secret code file created in the DF (or MF) can be interpreted by the operating system. Each secret code file can store up to eight secret codes.

Secret codes are stored on eight bytes. The first four bytes correspond to the header, followed by a four-byte code value. The header values, excluding the checksum and the MPN, can be read using the **Get Card Information** command. The values of the codes themselves are inaccessible to the terminal. Secret codes are held at addresses 0 to 7. The number defines the order in which a secret code appears in the file. For example, the first secret code in a file corresponds to address 0, the second to address 1 and so on.

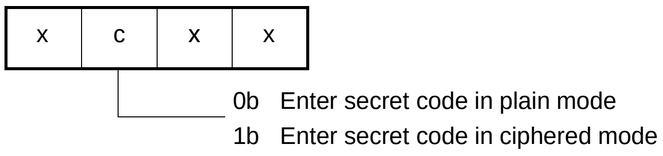
Secret codes have the following structure:

Mode	MPN	SCR	UCR	Cks
Secret Code (4 bytes)				

Figure 12 - Structure of Secret Code

Where:

Mode	Defines how the secret code is to be entered—in plain or ciphered text—on one nibble. It has the following format:


x
c
x
x
0b Enter secret code in plain mode
1b Enter secret code in ciphered mode

MPN (Maximum Presentation Number)	Defines the maximum presentation number on one nibble. This specifies the number of times that the secret code can be incorrectly entered consecutively before MPCOS-EMV locks it. This number from 1 to 7 is coded on bits 0 to 2.
SCR (Secret Code Ratification)	Secret code ratification counter, on one byte. When a secret code is initialized this value must be set to 0h. The counter records the number of consecutive times that the secret code has been presented incorrectly. When the counter value reaches the MPN value, the card locks the secret code. Each time the secret code is correctly entered, the card sets this value to 00h. However, once the code has been locked, a successful presentation will not reset the counter to 00h (and will not unlock the secret code). To unlock a secret code use the **Set Secret Code** command to present the code specified in the UCR byte (see below).
UCR (Unlock Code Reference)	Specifies the secret code that must be entered to unlock the secret code for this header (see *"Set Secret Code (SetCod)"* on page 128). This byte has the following structure:

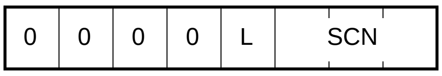


Figure 13 - Structure of an Unlock Code Reference

Where:

L (Level)	Defines the level of the EF_{sc} described above. The value 0 specifies the global EF_{sc}. The value 1 specifies the local EF_{sc}.
SCN (Secret Code Number)	In either the current dedicated file EF_{sc} or the master file EF_{sc} that has to be successfully submitted to unlock the secret code for this header.

Cks (Checksum) The checksum is derived from the following:

1. Perform an XOR operation of the six bytes (all bytes excluding the Checksum and SCR bytes).
2. Invert the resulting XOR operation.

Secret Code The four-byte secret code value. Terminals submit eight-byte secret codes. When the card receives the secret code, MPCOS-EMV extracts the least significant nibble of each character. An eight-byte secret code is therefore stored on four bytes in the secret code file. This is shown in *"Figure 14 - Example of How a Secret Code Is Stored"*.

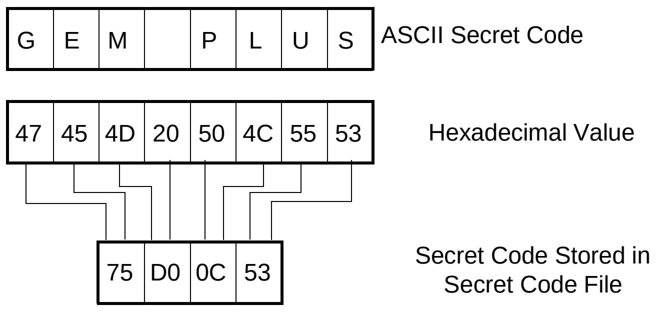


Figure 14 - Example of How a Secret Code Is Stored

Warning: For all applications, it is recommended that read access to secret code files be locked.

Internal Application Data File (IADF)

FDB value: 00001001b or 09h.

The IADF (for Internal Application Data File) is a specific file that can be interpreted by MPCOS-EMV in order to return information after the selection of a DF.

The IADF allows the implementation of the File Control Information (FCI) to be returned after the selection of a DF, as required by the EMV specifications. It also contains the internal data of the MSI application.

The IADF is a transparent file with a specific FDB value and it is updated by using the **Update Binary** and **Write Binary** commands.

Note: If no IADF is attached to a given DF, standard system data is returned (see *"Select File (SelFil)"* on page 116).

Any number of IADFs can be created in a DF, but only the first one can be interpreted by the operating system.

The IADF structure is shown in *"Figure 15 - Structure of an Internal Application Data File"*:

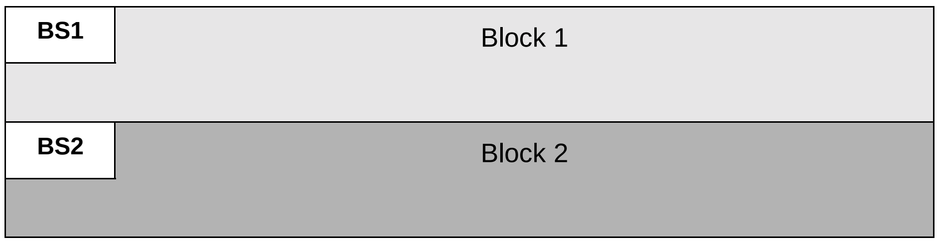


Figure 15 - Structure of an Internal Application Data File

Where:

BS1	1 byte	Defines the block size (in bytes) of Block 1, (that is, the size of the next field). If the indicated size is 0h (no Block 1), no FCI will be returned by the card.

Block 1	Codes the Answer to Select FCI. This field is directly interpreted by MPCOS-EMV to build the response message when selecting the DF. The global structure of Block 1 is as follows:

TLg	T1	L1	V1	T2	L2	V2	...

TLg	Total length of the response in bytes.
T_n, L_n, V_n	Represent a proprietary TLV format and are interpreted by MPCOS-EMV.
T_n	Represents a proprietary tag. MPCOS-EMV recognizes two tag values:

- T_n= 55h (direct addressing): V_n holds the data to be sent and L_n holds its length.
- T_n = AAh (logical addressing): V_n holds logical information used by the card to access the data, using file access mechanisms. L_n holds the data length (and may be different from V_n length).

  In this case, V_n is 3 bytes long and is coded as follows:

T	L	Short ID	Offset/Rec.nb.	Offset

Where:

T	Type (0: EF, 1: DF)
L	Level (0: Global, 1: Local)
ShortID	Short File Identifier (SFI) of the file.
Offset/Rec.nb.	Most Significant Bytes of offset in the case of a transparent file. Rec.nb. in the case of a structured file.
Offset	Offset in bytes.

- In the case of logical addressing in a DF, the data forms part of the DF name.
- In the case of logical addressing and when accessing an EF, the read access conditions should be unrestricted (the access conditions cannot actually be fulfilled at the time of the DF selection).
- The sum L1+L2+...+ L_n must be equal to TLg.

BS2	1 byte	Defines the block size (in bytes) of Block 2, (that is, the size of the next field).
Block 2	Coded as follows (applicable to MSI ADF only):	

Byte No.	Length (in bytes)	Specified Values	Description
1	1	00h	Application FLAG
2–4	3	Initialized	ATC(2) + Checksum (1)
5–7	3	Initialized	ATC_{backup} (2) + Checksum (1)
8–9	2	Initialized	ATC_{max} (2)
10	1	00h	DKI
11	1	0Ch	CVN
12	1	80h	Tag (for GetProcessingOpt Response)
13	1	Var.	Length (AIP(2) + AFL length (var))
14–15	2	1800h	AIP
16–XX	0 to 240 (multiple of 4)	See *“Table 7 - Application File Locator Bytes”.*	AFL

Where:

Application FLAG	Used by the operating system as indicators for card risk management (1 byte).
ATC	Application Transaction Counter (2 bytes). Incremented every time a transaction is initiated. It is protected by a checksum byte which is computed as the XOR of the two-byte ATC.
ATC_{backup}	Backup of the ATC. Protected by a checksum byte computed the same way as the ATC checksum.
ATC_{max}	The maximum value that the ATC can reach, beyond which the application can no longer be executed.
DKI	Derivation Key Index. Part of the data (cryptogram) returned in a **Generate AC** command.
CVN	Cryptogram Version Number. Part of the data (cryptogram) returned in a **Generate AC** command.
AIP	Application Interchange Profile. These bytes determine the functions supported by the card, coded as follows:

Byte No.	Bit	Value*	Definition
14	8	00b	Reserved for Future Use.
14	7	00b	Offline Static Data Authentication: not supported.
14	6	00b	Offline Dynamic Data Authentication: not supported.
14	5	01b	Cardholder Verification: supported.
14	4	01b	Terminal risk management performed.
14	3	00b	Issuer Authentication: not supported.
14	2	00b	Combined DDA-Generate AC: not supported
14	1	00b	Reserved for Future Use.
15	8–1	0000 0000b	Reserved for Future Use.

* Other values are not supported.

Table 6 - Application Interchange Profile Bytes

AFL	Application File Locator. Lists the files and records that are used by the application in the transaction. Each AFL entry consists of 4 bytes and is coded as follows:

Byte No.	Definition
1	Short File Identifier (SFI)
2	Record number of the first record to read
3	Record number of the last record to read
4	Number of records involved in SDA: not applicable

Table 7 - Application File Locator Bytes

Note: The main way to identify an ADF is through this IADF with Block 2 (that has been properly initialized). Otherwise, it will be regarded as normal MPCOS-EMV DF.

Directory File

The Directory File (Directory EF) is a record EF listing DDFs and Application Definition Files contained within the directory and it must be accessible by the **Read Record** command.

Application Elementary File

The Application Elementary Files (AEFs) are EFs that contain the data elements used by the application in its processing. Specifically, these EFs are record files and the data in TLV format can be retrieved by using the **Read Record** command.

EF Structures

Elementary files take the form of one of the following data structures:

- Transparent file
- Structured file

Transparent Files

FDB value: 00000001b or 01h.

A transparent file consists of an unstructured sequence of bytes.

The offset size is given in data units which can be 1 byte or 4 bytes. The size of a data unit is written to the lock byte, using the **Set Card Status** command (see *“Set Card Status”* on page 125), during the personalization stage and is displayed in the Answer To Reset (see *“Default Answer To Reset”* on page 149). The contents of the lock byte can be viewed by invoking the **Get Card Information** command. The first byte of a transparent EF has the relative address 00h.

Note: LOCK2 is the complement of LOCK1. The response to the **Get Card Information** command will indicate that LOCK2 is the same as LOCK1.

Data in a transparent file can be addressed using the offset, as shown in *“Figure 16 - Data Referencing in a Transparent File”*.

Data Unit	4 bytes	1 byte	
	Offset	Offset	Byte to Be Read
	0h	0h	B1
		1h	B2
		2h	B3
		3h	B4
	1h	4h	B5
		5h	B6

Figure 16 - Data Referencing in a Transparent File

Note: Data is no longer allocated on a word boundary. The operating system will not pad additional bytes.

Structured Files

Structured files consist of the following types:

- Linear fixed files
- Linear variable files
- Cyclic files

Each type is described separately in the following sections. Structured files contain records and each record reserves one byte for system use.

Note: The space will be allocated exactly the number of bytes, as specified. The operating system does not pad additional bytes.

Linear Fixed Files

FDB value: 00000010b or 02h. - No further information.

Or FDB value: 00000011b or 03h. - Simple TLV (Tag, Length, Value)

A linear fixed file consists of a sequence of individually identifiable records of the same size. The size is determined during file creation and is stored in the file descriptor (see *"File Descriptor"* on page 10).

Records are referenced #1, #2, #3,... in the order of their creation.

Note:
- Updating a record does not modify the record number.
- The maximum number of records in a linear variable file is 255
- The maximum length of each record is 255 bytes.

Data in a linear file with records of fixed size can be referenced as shown in *"Figure 17 - Data Referencing in a Linear File with Records of Fixed Size"*:

SB	Record 1
SB	Record 2
SB	Record...
SB	Record n-1
SB	Record n

SB is a byte reserved by the operating system.

Figure 17 - Data Referencing in a Linear File with Records of Fixed Size

Linear Variable Files

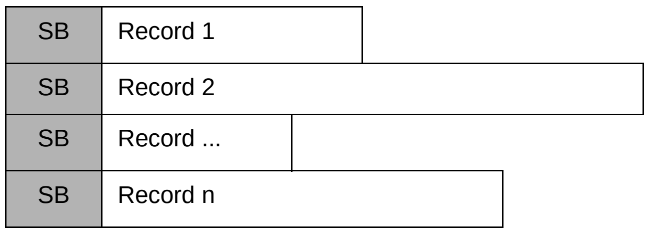
FDB value: 00000100b or 04h. - No further information.

Or FDB value: 00000101b or 05h. - Simple TLV (variable length).

The record selection is the same as for linear files with records of fixed size.

The file is handled by the interface as a sequence of independent records.

*"Figure 18 - Data Referencing in a Linear File with Records of Variable Size"*shows the global file structure:



SB is a byte reserved by the operating system.

Figure 18 - Data Referencing in a Linear File with Records of Variable Size

Note: • The maximum number of records in a linear variable file is 255.
• The maximum length of each record is 255 bytes.

Cyclic Elementary Files

FDB value: 00000110b or 06h. - No further info.

Or FDB value: 00000111b or 07h. - Simple TLV.

A cyclic EF consists of a sequence of records of the same length, used to store information in chronological order. When all the records have been used for storage, the data update overwrites the oldest record. The maximum number of records is 254.

The file is handled by the interface as a sequence of independent records.

The record length is indicated during the file creation and stored inside the file descriptor.

The records are referenced as #1, #2, #3, in the inverse order of their creation. This means that the last written record (appended or updated) is always referenced as record #1. The current record is always record #1. On selection of a cyclic EF, the current record reference is #1.

Note: • The maximum number of records in a cyclic file is 254.
• The maximum length of each record is 255 bytes.

The data is referenced as shown in *“Figure 19 - Data Referencing in a Cyclic File with Records of Fixed Size”*

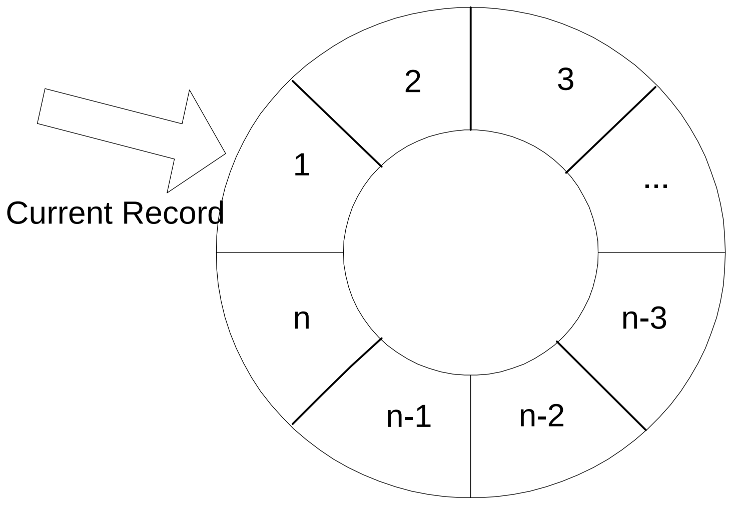


Figure 19 - Data Referencing in a Cyclic File with Records of Fixed Size

Each record reserves one byte for system use. This byte must be taken into consideration when calculating the body size of a cyclic file. In addition, each cyclic file needs another four bytes that are reserved for system use.

Note: Each record does not need to be a multiple of four bytes since the memory allocation is on a word boundary; no padding will be done.

The **Append Record** command does not cycle; that is, after appending the last record of the file, no more **Append Record** commands are allowed.

Reading the current record will always return the contents of the last appended or the last updated record.

Before updating, the current record pointer is incremented, so that the next record is updated and becomes the current record. The **Update Record** command cycles at the end of the file.

3

Access Conditions

General

MPCOS-EMV cards allow file access to be protected using secret codes or keys. Secret codes are stored in elementary files called Secret Code Elementary Files (EF_{sc}). The master file and each dedicated file can hold one EF_{sc} (see *"Secret Code Files"* on page 18).

Access to MPCOS-EMV files is protected by secret codes. The secret codes are stored in a specific EF that is called Secret Code Elementary File (EF_{sc}). Each EF_{sc} can store up to eight secret codes, numbered from 0 to 7. The MF and each DF can hold only one EF_{sc}.

Access conditions define the level of protection for a file. Access conditions are two-byte registers stored in Dedicated File (DF) and Elementary File (EF) descriptors. They define the following points:

- Up to two secret code numbers that must be submitted to access the file for each access type (create, read, write or update),
- The secret key used for all secure messaging performed on the file for each access type (create, read, write or update).

Note: Keys can only be used to generate Computation of the Cryptographic Checksum (CRYCKS), they cannot be used to encrypt data.

Data protection conditions can be defined by the application issuer for each EF or DF.

For example, an application issuer creates a data EF requiring the terminal to perform the following steps before data can be accessed:

1. Select the parent DF.
2. Ask the cardholder to type in his/her PIN code (read access to this EF is protected by PIN code presentation).
3. Submit this code to the card.

When the terminal submits a secret code, it specifies a secret code number from 0 to 7. The card compares the secret code submitted with the secret code specified by the EF_{sc}.

If the secret codes do not match, MPCOS-EMV returns an error status byte. If the secret codes do match, MPCOS-EMV sets the bit number corresponding to the number of the successfully submitted secret code to 1 in an authorization register (see *"Authorization Registers"* on page 32). MPCOS-EMV cards have two authorization registers, as follows:

- *MF authorization register.* Records the secret codes that a terminal successfully submits while the MF is selected. This is also called the global level register (the EF_{sc} at the global level).
- *DF authorization register.* Records the secret codes that a terminal successfully submits while a DF is selected. This is also called the local level register (the EF_{sc} at the local level).

The following paragraphs describe access conditions and authorization register and how MPCOS-EMV manages them.

Access Conditions

Access conditions specify the following:

- The secret codes that must be submitted before access can be granted to a file.
- The key file containing the key used for secure messaging transactions with the file (see *"Secure Messaging"* on pag e41).

Access conditions are initialized by the **Create File** command when the file is created. They can also be *locked* or *localized* using the **Freeze Access Conditions** command at any stage in the card's life-cycle.

- Locking an access condition means setting it to 1 so that access is permanently refused.
- Localizing an access condition means moving the access condition key or secret code references from the MF EF_{key} and EF_{sc} files to the parent DF EF_{key} and EF_{sc} files.

There are three groups of access conditions for each MPCOS-EMV file. These are shown in *"Table 8 - DF and EF Access Conditions"*:

Group #	In a Dedicated File	In an Elementary File
1	**Create** sensitive files	**Update** binary
	Freeze AC in sensitive files	**Update** record
2	**Create** common files	**Write** binary
	Freeze AC in common files	**Append** record
3	Not used	**Read** binary
		Read record

Table 8 - DF and EF Access Conditions

Data files are EFs that contain data which is not interpreted by the card. MPCOS-EMV recognizes these files by their File Descriptor Byte (FDB). Bits 5–3 are set to 000b. For example, a data file may be one that contains the card holder's address.

Sensitive files contain data that is interpreted by the card. The following files are sensitive files (bits 5–3 of the FDB are shown between parentheses):

- All DFs (b5b4b3=111b)
- Key EFs (b5b4b3=101b)
- Secret code EFs (b5b4b3=100b)
- Purse EFs (b5b4b3=011b)
- Transaction manager EFs (b5b4b3=010b)
- Internal Application Data File, IADF (b5b4b3=001b)

Access conditions can assign the following levels of security to a file:

- No protection
- Secure messaging
- Up to two secret codes
- Access never allowed

The secure messaging level is described in *"Secure Messaging"* on page 41.

The secret codes protect access to the files for each access condition group.

The following diagram shows how an access condition is coded:

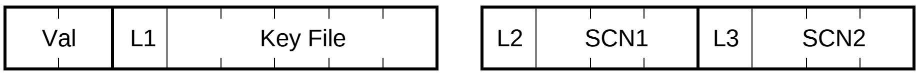


Figure 20 - Coding of Access Conditions

Where:

Val Validity field that defines the group protection level as follows:

Value	**Protection Level**
00b	No secret code protection.
01b	File protected by the secret code specified in SCN1 (and possibly secure messaging, see *"Key File"* on this page).
10b	File protected by the secret codes specified in SCN1 and SCN2 (and possibly secure messaging, see *"Key File"* on this page).
11b	Access never allowed.

L1 Defines the level of the EF_{key} file. It can be either at the global level or at the local level. The value 0 specifies the global level and the value 1 specifies the local level.

Key File	Short identifier of the key file (5 bits), at the level specified in L1, that contains the key that the **Select File Key** command used to generate the temporary key (see *"Select File Key (SelFk)"* on page 119) for the current administration transaction session. If the key used to generate the temporary key is not in the file specified here, access is denied to the access condition group for secure messaging. The value 00h means that the secure messaging is not needed to transact with the access condition group.
L2	Defines the level of the EF_{sc} containing SCN1. It can be either the EFsc at the global level or an EF_{sc} at the local level. The value 0 specifies the global level and the value 1 specifies the local level.
SCN1	Defines the first secret code number (0 to 7) in the EF_{sc} for the level specified in L2 that has to be entered to gain access to this group.
L3	Defines the level of the EF_{sc} containing SCN2. It can be either an EF_{sc} at the global level or an EF_{sc} at the local level. The value 0 specifies the global level and the value 1 specifies the local level.
SCN2	Defines the second secret code number (0 to 7) in the EF_{sc} for the level specified in L3 that has to be entered to gain access to this group.

Warning: The access condition for **Read** command is handled differently from that for **Update** and **Write** commands. The key file number indicates the key file to use during secure messaging. Even if it is specified, it is still possible to read the file without using secure messaging when the CLA byte is set to 00h.

Authorization Registers

Authorization registers are eight-bit RAM registers that keep a record of the successfully entered secret codes. MPCOS-EMV manages two authorization registers:

- *MF Authorization Register.* Stores security status at the global level.
- *Local DF Authorization Register.* Stores security status at the local level.

Each time a secret code is successfully entered using the **Verify** command, the card writes a 1 to the corresponding bit in the authorization register for the level at which the secret code was presented. For example, if secret code 2 is successfully submitted at the local level, bit number 2 in the DF authorization register is set to 1.

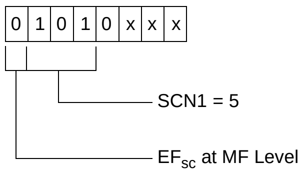
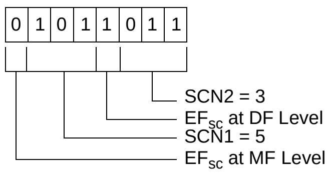
Warning: For payment-dedicated commands (**Debit, Credit, Read Balance**) Secret Code 0 means unrestricted access (not protected by Manufacturer Secret Code).

MPCOS-EMV cards manage authorization registers according to the following rules:

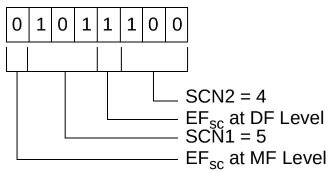
- The card sets all authorization registers to zero when it is reset.
- The card sets the DF authorization register to zero each time the terminal selects a new DF (even if the new DF is the same file as the previously selected DF).
- The card never sets the MF authorization register to zero during a session.

Example

The following example shows how the MPCOS-EMV data protection system works.

Terminal	**MPCOS-EMV R4**	**Authorization Register Status at the Completion of the Command**	
		MF Level Global	**DF Level Local**
Reset Card	Set authorization registers to zero	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
Verify Secret Code XYZ (No. 5)	Compare value XYZ with Secret Code No. 5 in Master File EF_{sc}. If the secret code is correct, update the MF authorization register.	5 0 0 1 0 0 0 0 0	0 0 0 0 0 0 0 0
Create Dedicated File	Check MF Access Conditions for Group 1, byte 2  SCN = 5, bit 5 in MF Authorization Register = 1 Allow Access		
Select Dedicated File 51	Select Dedicated File 51.	5 0 0 1 0 0 0 0 0	0 0 0 0 0 0 0 0
Verify Secret Code ABC (No. 3)	Compare value ABC with Secret Code No.3 in DF51 EF_{sc}. If the secret code is correct, update the DF.	5 0 0 1 0 0 0 0 0	3 0 0 0 0 1 0 0 0
Select Elementary File 14	Select Elementary File 14.		
Write to Elementary File	Check EF14 Access Conditions, group 2, byte 2:  0 1 0 1 1 0 1 1 SCN2 = 3 EFsc at DF Level SCN1 = 5 EFsc at MF Level SCN 1 = 5, SCN 2 = 3 MF Authorization Register bit 5 = 1 DF Authorization Register bit 3 = 1 Allow access		
Select Dedicated File 40	Select Dedicated File 40.	5 0 0 1 0 0 0 0 0	0 0 0 0 0 0 0 0

Terminal	**MPCOS-EMV R4**	**Authorization Register Status at the Completion of the Command**	
		MF Level Global	**DF Level Local**
Verify Secret Code DEF (No. 2)	Compare value DEF with Secret Code No. 2 in DF40 EF_{sc}. If the secret code is correct, update the DF authorization register.	5 0 0 1 0 0 0 0 0	2 0 0 0 0 0 1 0 0
Select Elementary File 20	Select Elementary File 20.	5 0 0 1 0 0 0 0 0	2 0 0 0 0 0 1 0 0
Read from Elementary File	Check EF20 access conditions group 3, byte 2:		



SCN 1 = 5, SCN 2 = 4
MF Authorization Register bit 5 = 1
DF Authorization Register bit 4 = 0
Access not allowed

4

Cryptography

3DES Algorithm

3DES can be implemented to achieve various security scheme requirements.

This cryptographic algorithm is the secret key symmetric algorithm: "3DES" in EDE mode (cf. ANSI X9.17 standards).

3DES requires the use of keys which are 16 bytes in length (also called double-length keys). This 3DES key thus consists of a key K (Kl; Kr) where Kl is the left part of the key and Kr is the right part of the key.

The following diagram shows how 3DES is implemented in MPCOS-EMV:

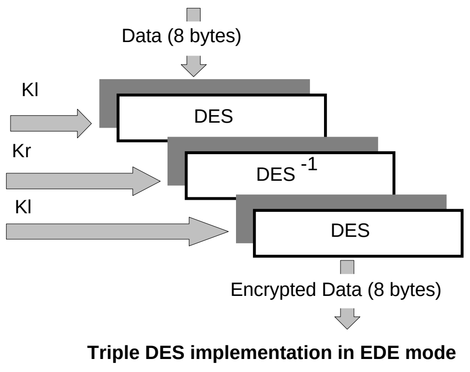
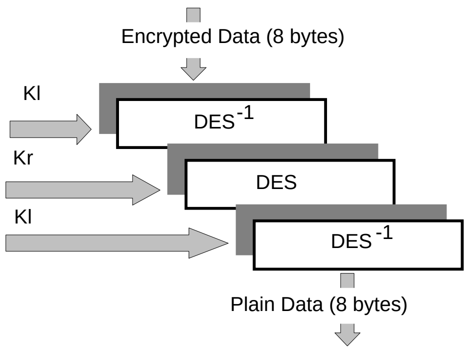


Figure 21 - Triple DES Implementation in EDE Mode

The inverse function 3DES^{-1} is as shown below:

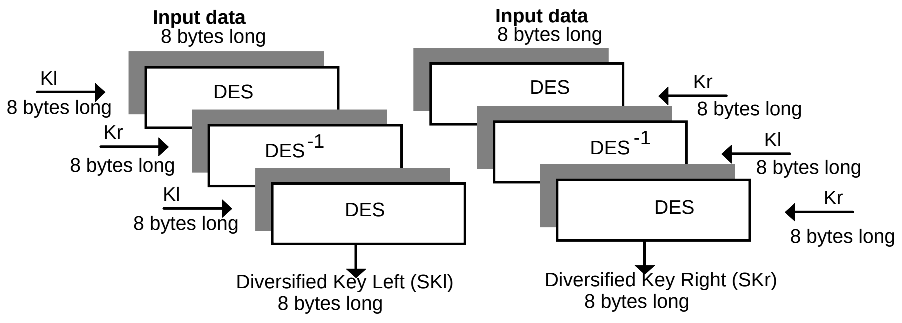


Inverse Triple DES implementation

Figure 22 - Inverse Triple DES Implementation

Key Diversification

The card computes a temporary diversified 3DES key SK (SKl; SKr) from a card secret 3DES key K (Kl; Kr) as follows:



Temporary diversified 3DES key : SK (SKl; SKr)

Figure 23 - Temporary Diversified 3DES Key

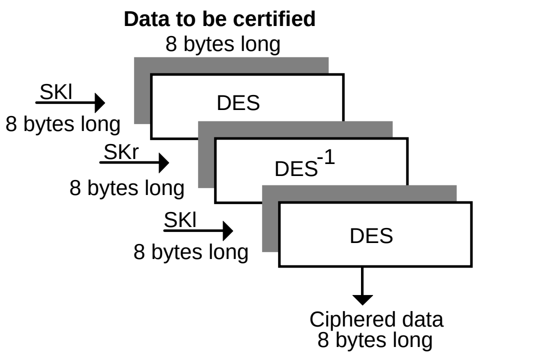
The card thus has a temporary diversified 3DES key SK (SKl; SKr). This temporary key can be used to compute certificates during transactions.

Notation for 3DES key diversification: SK = 3DES_16[data]<K>

Note: 3DES_16 gives a result of 16 bytes, while 3DES gives a result of 8 bytes. For key diversification, 3DES is used in ECB (Electronic Code Book) mode.

Authentication/Computation of Certificates

Certificates or authentication cryptograms are computed using 3DES as follows:



Certificate = part of the ciphered data (e.g., 3 bytes)

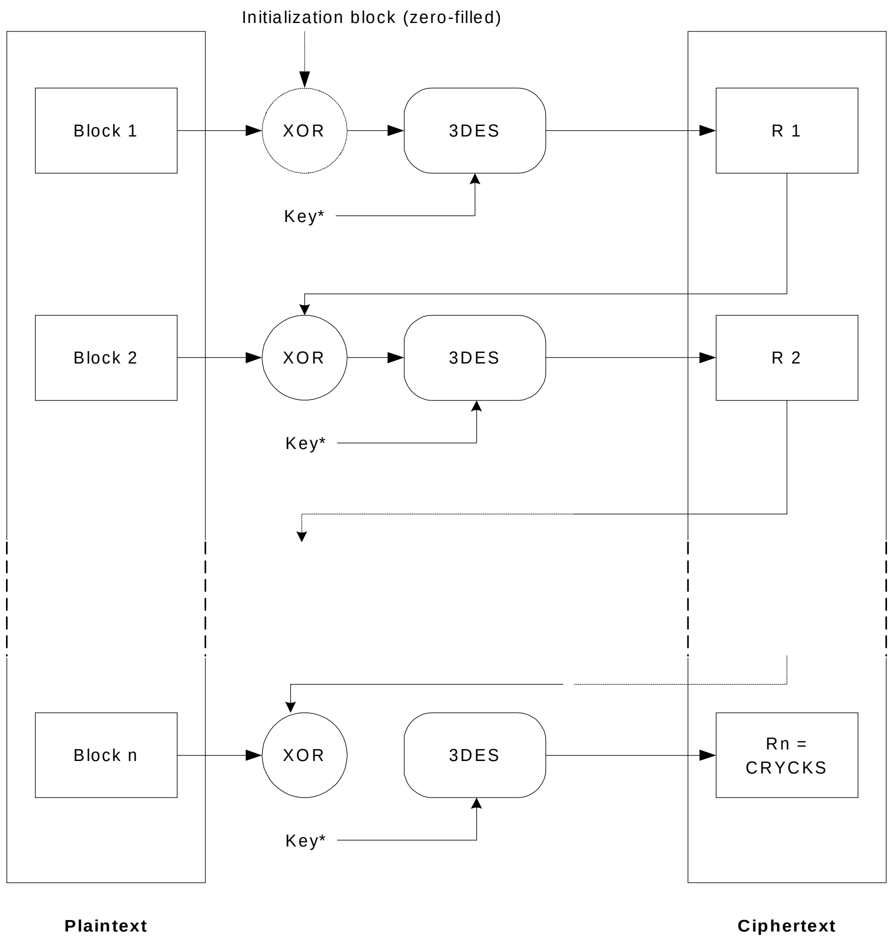
Figure 24 - Certificate Computation

Computation of the Cryptographic Checksum

Cryptographic checksums (CRYCKS) are used in the secure messaging process in MPCOS-EMV, in particular they are used by the data management and other commands (for example, **Update Binary, Write Binary** and **Create File** commands).

The CRYCKS is calculated using the whole of the data to be exchanged between the card and the terminal (including the command header at the beginning). The CRYCKS is computed using the Cipher Block Chaining mode.

The CBC mechanism consists of dividing the command header plus the data into eight-byte blocks. If the data is not a multiple of eight bytes, the last block will be padded with zeroes as described in the next paragraph. The result of each encrypted block is fed back into the encryption of the next block by means of an XOR operation as shown in the following diagram.



* Temporary session key

Figure 25 - Computing a Cryptographic Checksum

The following example shows how to compute the CRYCKS as required to update data in the card.

CLA	INS	P1	P2	Lc	Data (3 bytes)	Data (8 bytes)	...	Data (6 bytes)	00	00
Block 1						Block 2	...	Block n		

The data is arranged in blocks of eight bytes. The first data block, Block 1 comprises the five bytes of the header (CLA, INS, P1, P2, Lc) plus the first three bytes of data. Subsequent blocks, Block 2, Block 3 etc. contain the rest of the data in blocks of eight bytes. In this example, the final block has six bytes of data, so the final two bytes are zero-padded.

The cryptographic checksum is calculated as follows:

Kts is the temporary session key.

1. R1 is calculated as 3DES [Block 1]<Kts>.
2. R2 is calculated as 3DES [Block 2 XOR R1]<Kts>.
3. This is repeated for each block of eight bytes, with the result of one computation (R_i) being used in the next one (R_{i+1}).
4. Rn is the final computation, using the final data block, Block n and is calculated as 3DES [Block n XOR R_{n-1}]<Kts>.
5. CRYCKS is the result of this final computation (that is, Rn).

Only the three least significant bytes of the CRYCKS will be used for the **Update Binary** command in secure messaging.

Zero Padding

As all the cryptography features used in the MPCOS-EMV card are based on the 3DES algorithm, the length of the data to be enciphered, deciphered or signed must be multiple of eight bytes. If this condition is not satisfied, zero padding will be performed on the last block of eight bytes before, as shown below:

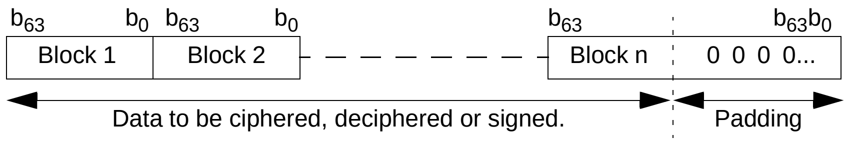


Figure 26 - Zero Padding Method

This padding corresponds to the method 1 described in the ISO/IEC 9797 standard (ref. [5]).

MPCOS-EMV Keys

MPCOS-EMV keys can be used for the following processes:

- Encrypting/decrypting secret codes/keys
- Computing certificates and signatures
- Secure messaging

Keys are supplied by any of the following two sources:

- Key files which are initialized during the card personalization
- MPCOS-EMV (temporary keys)

Key Types Loadable in Cards

Keys initialized during card personalization are loaded in key files. The different types of keys are stored in separate key files. The type of key is defined in the key file header.

Different types of keys can be used in MPCOS-EMV. For further information on these key types, see *“Key Files”* on page 15.

3DES cryptographic keys (16 bytes in length) will actually be stored as 24 bytes since these are stored as two consecutive DES keys.

In a given DF, several key files can be created. When designing an application, it is recommended that a key file be created for each function. For instance a key file should be created to store keys dedicated to authentication.

Keys can be used only in pairs. Key numbers should always be even numbers. Key numbers that are odd numbers will be rejected.

Temporary Keys Generated by MPCOS-EMV

Other types of keys can also be generated by MPCOS-EMV on the basis of keys stored in the card:

- A temporary administration key is generated when the **Select File Key** command is executed, starting a new administration session. This key is used for the secure messaging process during a session.
- A temporary payment key is generated when the **Select Purse & Key** command is executed, starting a new transaction session. This key is used in the generation of certificates and during most encryption procedures.
- A temporary signature key is used by MPCOS-EMV to generate the signature certificate. See *“Sign”* on page 131 for further details.
- A temporary credit key is generated by the host or by a terminal when the **Credit** command is executed and it is used to encrypt and decrypt transmissions between the host, terminal and the card. For further details, see *“Credit (Cdt)”* on page 80.

For example, MPCOS-EMV generates temporary certification keys using the following formula:

Key = 3DES_16 [CTC]<Ks>

Where:

CTC	Card Transaction Counter (see *“Transaction Manager Files”* on page 17).
Ks	Key selected from a key file.

When Temporary Keys Are Lost

A temporary certification or administration key will be lost under any of the following circumstances:

- Card reset
- Selection of a new DF, unless selected from the MF
- Execution of the **Sign** command in Terminate mode (see *“Set Options (SetOpts)”* on page 126) (temporary signature keys)
- Execution of the **Credit** command (temporary credit keys)
- Security error due to secure messaging
- Incorrect execution of the **Read Balance** command
- Incorrect execution of the **Select File Key** or **Select Purse & Key** commands

- Change of session (administrative/payment); for example, invoking a new **Select File Key** or **Select Purse & Key** command.
- Any integrity error (for example, file integrity, key checksum, secret code checksum, counter integrity, balance integrity and ONC integrity)
- The number of times the key is used exceeds the MRN.

Cryptographic Security Implementation

MPCOS-EMV can use the following cryptographic security features:

- Card/terminal authentication
- Secure administration command transmission using secure messaging
- Generation of payment certificates/signatures
- Payment command transmission protection using cryptograms

Card/Terminal Authentication

MPCOS cards support two authentication processes, as follows:

- *Internal Authentication.* The card proves it is genuine to the terminal. For further details, see *"Internal Authenticate (IntAut)"* on page 107.
- *External Authentication.* The terminal proves it is genuine to the card. For further details, see *"External Authenticate (ExtAut)"* on page 86.

Secure Messaging

MPCOS-EMV protects administration command transmissions between cards and terminals using secure messaging. Secure messaging will be implemented in the following processes:

- Encrypting data sent to the card. This is used with the **Write Binary** command when writing to an EF_{sc} or an EF_{key}, the **Update Binary** command when updating an EF_{sc} or an EF_{key} and the **Verify** command. Each command uses a different process to encrypt transmissions; see *"Chapter 9 - Commands"* for more details.
- Decrementing the Maximum Reply Number (MRN) counter.
- Sending three bytes of a cryptographic checksum with the command so that the receiver can verify the integrity of the transmission. The following paragraphs describe this process.

The following conditions must be met before secure messaging can be used:

- The **Select File Key** command must have been executed to establish an administration session (see *"Select File Key (SelFk)"* on page 119) or the **Select Purse & Key** command must have been executed to establish a temporary key using a log key (see *"Select Purse & Key (SelPK)"* on page 122).
- The key file specified in the Key File field in the access condition for the required access type must be the key file that holds either the key used by the **Select File Key** to generate the temporary administration transaction key (see *"Chapter 3 - Access Conditions"*) or the log key used by **Select Purse & Key** to generate a temporary key.

The terminal notifies the card that it is using secure messaging by entering **84h** or **04h** in the class field of the APDU command, thus initializing the MRN counter for the session.

The secure messaging process is as follows:

For commands that send data to the card (such as Update or Create)

The terminal will perform the following:

1. Divide the command transmission into eight-byte blocks, as described in *"Computation of the Cryptographic Checksum"* on page 37.
2. Generate the cryptographic checksum based on all the eight-byte blocks using the temporary administration key.
3. Add the three least significant bytes of the cryptographic checksum (CRYCKS 2-0) to the data field.
4. Send the command APDU in the following format:

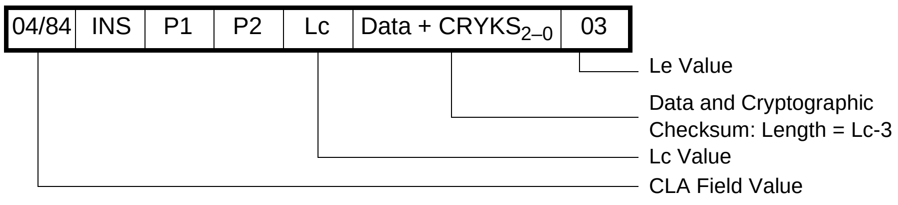
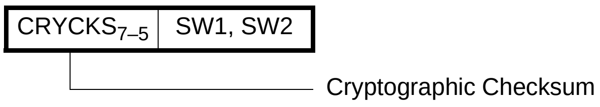


Figure 27 - APDU Format

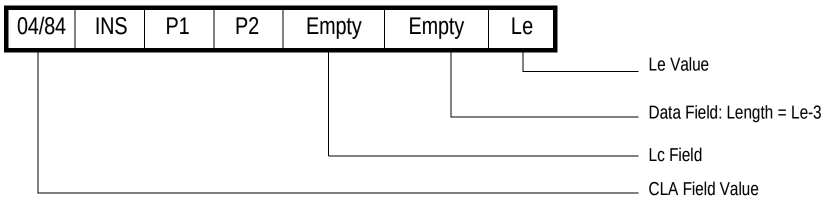
The card will perform the following:

1. Generate a cryptographic checksum for the command it receives using the same process as the terminal. It verifies that the three least significant bytes of the checksum that it generates match CRYCKS 2–0. If the bytes do not match, the card denies access for that command.
2. Execute the command.
3. Return the SW1, SW2 bytes and make the three most significant bytes of the generated cryptographic checksum available:



For commands that only retrieve data from the card (such as the Read command)

The terminal will send the command in the following format:

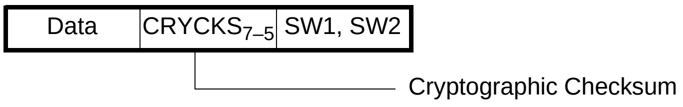


The card will perform the following:

1. Divide the return transmission into eight-byte blocks, as described in *"Computation of the Cryptographic Checksum"* on page 37.

Note: In this case, the first block consists of the command header (CLA, INS, P1, P2, Le) and the first three bytes of the data field.

2. Generate a cryptographic checksum based on all the eight-byte blocks and the temporary administration key.
3. Return the SW1, SW2 bytes and make the response data and the three most significant bytes of the cryptographic checksum available:



Note: You should refer to specific commands for further details on secure messaging (see **Read Binary, Update Binary, Write Binary, Read Record, Append Record, Update Record, Create File, Verify** and **Set Secret Code** commands in *“Chapter 9 - Commands”*).

Payment Certificates

MPCOS-EMV cards generate certificates after performing the following payment actions:

- A debit, which is performed by the **Debit** command
- Reading a purse balance, which is performed by the **Read Balance** command
- Canceling a previous debit, which is performed by the **Cancel Debit** command

The card can also generate a signature when requested. For more details, see *“Sign”* on page 131.

The process that generates the certificate is different for each command and you should refer to the individual command descriptions in *“Chapter 9 - Commands”*.

Certificates can be recorded and then used by the terminal/host system to validate a payment transaction.

For example, after a debit, depending on the personalization options, MPCOS-EMV cards will generate transaction certificates as follows:

Certificate = 3DES [	00h	Pn	Tv2	Tv1	Tv0	TTC2	TTC1	TTC0	]<Kpts>

Where:

Pn	Specifies the purse short file identifier on which the transaction was executed.
Tv2 to Tv0	Transaction value
TTC2 to TTC0	Terminal Transaction Counter (TTC), which is a three-byte value that MPCOS-EMV cards use to keep track of transactions executed with each terminal. Please see *“Terminal Transaction Counters”* on page 44.
Kpts	Temporary payment transaction key

Terminal Transaction Counters

Each terminal stores and maintains a Terminal Transaction Counter (TTC). MPCOS-EMV cards and terminals use TTCs during payment-related cryptographic procedures. They provide a variable element that can be used to identify a terminal and the transactions that it performs.

TTCs are one word in length and are structured as follows:

TTC3	TTC2	TTC1	TTC0

Bytes TTC3 and TTC2 store a terminal number that is defined and maintained by the application. This must be a unique hexadecimal number for each terminal used in the system.

Bytes TTC1 and TTC0 are to be used as a counter that the terminal increments each time it executes a **Debit, Credit** or **Cancel Debit** command. When the TTC reaches the maximimum value (FFFFh), there will be a counter error and the transaction will not be completed.

Payment Command Cryptograms

During payment transaction sessions, MPCOS-EMV cards and terminals generate cryptograms that can be used to verify the integrity of the transmission. Cryptograms are generated during the following operations:

- Crediting a purse. For more details, see *"Credit (Cdt)"* on page 80.
- Starting a transaction session. For more details, see *"Select Purse & Key (SelPK)"* on page 122.
- Canceling a previous debit. For more details, see *"Cancel Debit (CanDeb)"* on page 71.

Each command has a specific means of generating cryptograms. Refer to the command description in *"Chapter 9 - Commands"* for details.

Anti-DPA Implementation

Security in MPCOS-EMV is further enhanced with an Applicative Countermeasure (ACM) against Differential Power Analysis (DPA) attacks. In a DPA attack, power analysis is done when the card performs commands that require secret keys. The DPA attack is limited by restricting the number of times a secret key is wrongly used. This is accomplished by implementing the following security mechanisms:

- Off-Nominal Counter (ONC) mechanism
- Maximum Replay Number (MRN) mechanism

Off-Nominal Counter (ONC) Mechanism

An ONC is assigned to each secret key in the reserved bytes of the 3DES key files:

Key Type Part 1	Current ONC	Kv	Cks
K15	K14	K13	K12
K11	K10	K9	K8
Key Type Part 2	Backup ONC	Kv	Cks
K7	K6	K5	K4
K3	K2	K1	K0

Figure 28 - 3DES Key Record Structure

ONCs track commands that require permanent secret keys (marked with ✔), as follows:

Key Type	SelPK[1]	SelFK[2]	Credit[3]	Sign[4]	ExtAut[5]	IntAut[6]
Administrative		✔				
Payment	✔		✔	✔		
Log	✔		✔	✔		
Authentication					✔	✔
Signature			✔	✔		

1 The **Select Purse & Key (SelPK)** command uses a payment or a log key. The ONC associated with the key decrements each time this command is executed.

2 The **Select File Key (SelFK)** command uses an adminstration key sucessfully. The ONC associated with the key decrements each time this command is executed.

3 The **Credit** command uses a payment, log or signature key and it verifies an incoming cryptogram. The ONC associated with the key decrements if the verification fails.

4 The **Sign** command uses a payment, log or signature key. The ONC associated with the key decrements if the verification fails. However, the ONC does not decrement if there are no transactions to sign since no cryptographic computations is done.

5 The **External Authenticate (ExtAut)** command uses an authentication key and verifies an incoming cryptogram. The ONC associated with the key decrements if the verification fails.

6 The **Internal Authenticate (IntAut)** command uses an authentication key successfully. The ONC associated with the key decrements each time this command is executed.

Table 9 - Commands Tracked by Off-Nominal Counters

Optionally, the ONC associated with a key will decrement when an incoming cryptogram verification fails or when an outgoing cryptogram is generated. The key associated with an ONC will be disabled when the ONC value decrements to zero.

Upon successful execution of **External Authenticate** or ciphered **Verify** commands, all ONCs will be restored back to their original values at the start of the same transaction. Additionally, the ACM will be disabled for the rest of the transaction.

To initialize the ONC, do the following:

1. In the Key Type byte (see *“Key Type Part 1”* on pag e16), set b7 to 1.
2. Compute a checksum by performing an XOR operation on the 10 other bytes of the key part and inverting the final result.
3. Set b7 of the ONC byte to 1 for odd parity. b7 is the odd parity bit for teh ONC byte. Its value is set to make the total number of bits with value ’1’ to an odd number.

Bit	7	6	5	4	3	2	1	0
	Parity Bit	7-bit Counter						

Maximum Replay Number (MRN) Mechanism

Maximum Replay Number (MRN) is a mechanism that optionally puts an upper limit to the usage of temporary keys in a secure session. When the mechanism detects that the upper limit has been reached, the session key will be lost.

The following commands are tracked by the MRN mechanism:

- **Read Binary**
- **Read Record**
- **Read Balance**
- **Debit**
- **Cancel Debit**

The mechanism is optionally activated when **Select File Key** and **Select Purse Key** commands are executed.

5

Initial Status

Initialization Processes

Initialization of the microprocessor cards takes place after the chip is embedded into the card. During the initialization process basic card information such as the card serial number, the system files structure and the system keys are written into the chip.

MPCOS-EMV cards can be initialized by one of the two following initialization processes (which correspond to the different delivery modes):

Sample Shipment	The cards are protected by a common key. There is no batch card.
Secure Shipment	The cards are protected by a diversified key and contain a customer code.

The cards initialized by these two different processes have a similar basic structure. This structure is described in the sections that follow.

Initial File Structure

"Figure 29 - Initial File Structure" shows the initial file structure of MPCOS-EMV.

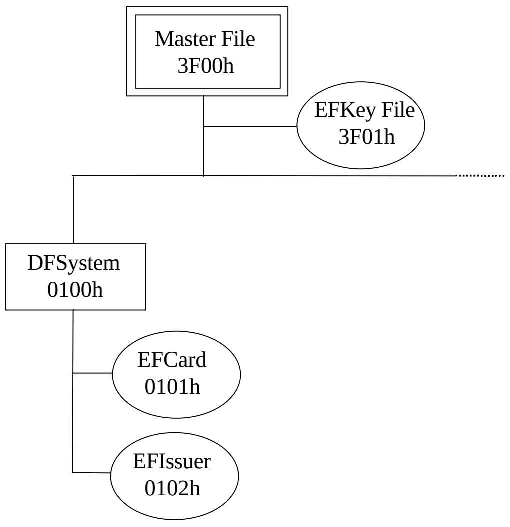


Figure 29 - Initial File Structure

Master File

The Master File (MF) is the root of the MPCOS-EMV file structure.

File Parameters

Identifier:	**3F00h**
Name:	None

The following options are fixed and cannot be changed by the issuer:

- ISO Dedicated File
- **Cancel Debit** command enabled
- Current balance can never be used to compute signature certificates.

Security Access Conditions

Create data files:	Secure messaging with EF_{key} 3F01h
Create sensitive files:	Secure messaging with EF_{key} **3F01h**

Warning: To allow new applications to be added in the future, no application should be loaded directly under the MF.

EF$_{key}$

The EF$_{key}$ contains the system key. This key is needed to carry out most personalization operations by the card application issuer. Its value depends on the card's initial status:

- For MPCOS-EMV cards initialized with the sample process, the value of this key is the same for all the cards: ASCII string TEST KEYTEST KEY.
- For MPCOS-EMV application cards initialized with the batch process, the key is diversified by Gemplus from a mother key that is stored inside a batch card. The batch card is delivered to the card issuer together with the batch of blank cards.

The EF$_{key}$ has a length of 24 bytes.

File Parameters

Identifier:	**3F01h**
Body Size:	24 bytes

Security Access Conditions

Read:	Locked
Write:	Secure messaging with EF$_{key}$
Update:	Secure messaging with EF$_{key}$

DF$_{system}$

The DF$_{system}$ is the Dedicated File (DF) containing the system elementary files.

File Parameters

Identifier:	**0100h**
Name:	**SYSTEM**

The following options are set and cannot be changed by the issuer:

- ISO Dedicated File
- **Cancel Debit** command enabled
- Current balance can never be used to compute signature certificates
- Crypto option: 3DES

Security Access Conditions

Create data files:	Secure messaging with EF$_{key}$ (in master file)
Create sensitive files:	Secure messaging with EF$_{key}$ (in master file)

Note: This DF is dedicated to system data. No application must be loaded directly under the DF system.

EF$_{card}$

The EF$_{card}$ is an elementary file containing the card serial number, issued by the card manufacturer, the issuer reference number and the Card Production Life Cycle (CPLC) data specified and updated by Gemplus.

File Parameters

Identifier:	**0101h**
Body Size:	36 bytes

Value	Length
Card Serial Number	8 bytes
Issuer Reference Number	4 bytes
CPLC Data	24 bytes

Security Access Conditions

Read:	Public
Write:	Locked
Update:	Locked

Card Serial Number

The Card Serial Number (CSN) is unique for each MPCOS-EMV card and it cannot be modified. It is coded on 8bytes (two words of 32 bits), as follows:

```
                                      Bits
 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
 [1       10 bits       ][2                         22 bits                                    ]
 [3       10 bits       ][4    6 bits    ][5       8 bits        ][6       8 bits              ]
```

1:	**Batch Number**	(10 bits)
2:	Serial Number (part)	(22 bits)
3:	Serial Number (part)	(10 bits)
4:	**Week Number**	(06 bits)
5:	Serial Number (part)	(08 bits)
6:	**Year of Manufacture**	(08 bits)

Fields 2, 3 and 5 make up the serial number in a batch.

Figure 30 - Coding of the Card Serial Number

Note: Gemplus reserves the right to modify the structure of the card serial number.

Caution: The bits in the year of manufacture represent the offset, in binary, from the base year of 1900. For example, the year 1999 is denoted by 63h and the year 2000 is denoted by 64h.

Please note that this is different from the case of batch card where the year in byte 4 of the ATR is coded in two BCD digits, representing the last two digits of the year. In the case of BCD digits, the year 1999 is denoted by 99h and the year 2000 is denoted by 00h.

Issuer Reference Number

The issuer reference number is coded on four bytes (three bytes plus a checksum). This number is provided by Gemplus as a unique reference for each customer and cannot be updated. It is recommended that this issuer code is used as a reference in your application.

In sample cards or for customers whose codes have not been assigned, the issuer reference number is set to 00 FF 00h.

CPLC Data

The Card Production Life Cycle (CPLC) data is stored in the EEPROM in EF 0101h and is updated by Gemplus. The data is structured as follows:

Value	Length
IC Fabrication Date	2 bytes
IC Serial Number	4 bytes
IC Batch Identifier	2 bytes
IC Module Fabricator	2 bytes
IC Module Packaging Date	2 bytes
ICC Manufacturer	2 bytes
IC Embedding Date	2 bytes
IC Pre-Personalizer	2 bytes
IC Pre-Personalization Date	2 bytes
Equipment Identifier	4 bytes

Table 10 - Structure of CPLC Data in EF 0101h

Each date is coded in the format “YDDD” (in BCD format).

Where:

Y	The last digit of the year.
DDD	BCD of the day number in the year.

EF$_{issuer}$

This is an elementary file containing the issuer data and the CPLC data. The issuer data is stored and updated by the card application issuer during the personalization phase.

The CPLC data is specified and updated by the application issuer. The default value is zero.

File Parameters

Identifier:	**0102h**
Body Size:	20 bytes

Value	Length
Issuer Data	12 bytes
CPLC Data	8 bytes

Security Access Conditions

Read:	Public
Write:	Secure messaging with EF$_{key}$ (in master file)
Update:	Secure messaging with EF$_{key}$ (in master file)

CPLC Data

The Card Production Life Cycle (CPLC) data is structured as follows:

Value	Length
IC Personalizer	2 bytes
IC Personalization Date	2 bytes
IC Personalization Equipment Identifier	4 bytes

Table 11 - Structure of CPLC Data in EF 0102h

The date is coded in the format “YDDD” (in BCD format).

Where:

Y	The last digit of the year.
DDD	BCD of the day number in the year.

Personalization Flag

In their initial status, the cards’ personalization (Bissuer) flag is not set. That is, the personalization flag is set to 0. The flag is set to 1 once you have completed personalization. Setting this flag shows you have switched to user mode. It is set by the **Set Card Status** command. For more information, please see *“Set Card Status”* on page 125.

IO Buffer Size

The MPCOS-EMV IO buffer size is the same for all the cards (64 bytes). However, MPCOS-EMV manages chaining for normal processing. The maximum data does not depend on the buffer size, but it is specified for each APDU command.

For example, without secure messaging, it is possible to update a record of up to 255 bytes. However, in secure messaging, chaining is not allowed and the buffer size restricts the data size to a maximum of 61 bytes only (buffer size of 64 bytes minus three bytes for the cryptographic checksum).

Note: While it is possible to transfer more than 64bytes by chaining, it is faster to transfer multiple blocks of 64 bytes.

Lock Byte

The lock byte can be read from an MPCOS-EMV card using the **Get Card Information** command. It is updated using the **Set Card Status** command. The lock byte is backed up by the operating system for security purpose. See *"Get Card Information (GetInfo)"* on page 95 and *"Set Card Status"* on page 125 for more information.

"Figure 31 - Detailed Description of a Lock Byte" describes the lock byte:

7	6	5	4	3	2	1	0
RFU by Gemplus	Data Unit	0	RFU (0)	Perso (Bissuer)	1	1	Bconfig.

Figure 31 - Detailed Description of a Lock Byte

Where:

Data Unit	0: Four-byte data unit (compatible with the earlier MPCOS version). 1: One-byte data unit (as required in EMV specifications).
Bissuer (PERSO)	Set to 1 when the personalization has been completed and the card becomes user mode. The **Activate System** command is deactivated.
Bconfig.	Set to 1 when the card is using a customized flow. Kept at 0 when the card is using a standard flow. This bit is only a flag indicating which initialization process has been carried out on the card. It has no effect on the card's behavior.

6

Optional Features

You can order custom configurations for the following MPCOS-EMV card features:

- Communication speed
- Communication protocols
- Answer To Reset
- Custom OS extensions

Warning: These custom configurations are implemented by Gemplus during the card personalization. You must specify all custom configurations when ordering your MPCOS-EMV cards.

Communication Speed

The MPCOS-EMV card communication speed is specified using conversion parameters (clock conversion rate F and bit adjustment rate D) and the clock frequency of the card interface device. The conversion parameters F and D are defined by the ISO 7816-3 standard. The default MPCOS-EMV card communication parameters implement a communication speed of 9600 Baud with the card interface device clock set to 3.68 MHz. These values correspond to F = 372 and D = 1 in relation to ISO 7816-3.

These parameters can be changed using either the double reset mechanism or the **Switch Protocol** command.

Switch Protocol Command

The **Switch Protocol** command enables you to specify the communication speed.

For example, if your card interface device (such as the Gemplus GemPC 410 reader) has an internal clock of 3.68 MHz, the following baud rates can be specified for MPCOS-EMV cards:

- 9,600 Baud
- 19,200 Baud
- 38,400 Baud
- 57,600 Baud
- 115,200 Baud

Double Reset Mechanism

If you need to use two different communication speeds for different phases of your application, you can specify a default speed that will be used after the Answer To Reset sequence and another speed that will be used after a double reset (without powering off the card). The first reset is called a “cold reset.” The second reset is called a “warm reset.”

The Answer To Reset returns the TA1 parameter which specifies the current card communications rate, so that the card interface device can adjust itself accordingly.

If you want to implement a double reset mechanism, please contact a Gemplus technical consultant.

Communication Protocols

MPCOS-EMV cards run under the ISO 7816-3 T = 0 protocol by default.

MPCOS-EMV cards can also be configured to run under the T = 1 protocol described in Clause 9 of the ISO 7816-3 standard (see *“Appendix E - T = 1 Protocol”*).

Additional protocols (T = 1; T = 14) can be activated in one of the following ways:

- Using the **Switch Protocol** command
- Installing a custom default Answer To Reset
- Installing a custom Double Reset mechanism

For information on installing a custom Answer To Reset or a custom Double Reset mechanism, please contact your Gemplus technical consultant.

Answer To Reset

The standard MPCOS-EMV Answer To Reset is described in *“Appendix A - Default Answer To Reset”*. It is an ISO 7816-4 compatible sequence. Additional information can be obtained by sending the **Get Response** command immediately after the Answer To Reset sequence.

If you selected custom communication speed and protocols, Gemplus can change the card interface parameters (see above).

Gemplus can also add customized historical bytes to suit specific issuer requirements.

Custom OS Extensions

Gemplus can develop and add custom OS extensions to satisfy specific issuer requirements. These are loaded as an application filter to the MPCOS-EMV card.

Warning: You must order cards with the customized option if you require any of the following features:

- Double Reset mechanism
- Specific Answer To Reset
- Custom OS Extensions

Standard cards do not offer these features.

7 Electronic Purse Architecture

Gemplus has issued a document called *Electronic Purse Architecture for MPCOS-EMV*. This document describes the minimum requirements to build a secure purse structure.

8

Command Format

MPCOS-EMV cards accept commands and responses in compliance with the Application Protocol Data Unit (APDU) format defined by the ISO 7816-4 standard. This enables MPCOS-EMV commands to be compatible with the Gemplus GCR Interface Driver Library.

However, you should bear in mind that the MPCOS-EMV transmission layer protocol complies with the ISO 7816-3 T = 0 standard, which converts APDUs into Transmission Protocol Data Units (TPDUs). The reader sends TPDU commands to the card and the card returns TPDU responses to the reader.

MPCOS-EMV cards accept commands in any of the following cases:

- **Case 1**
  No command or response data. This is sent as a T = 0 ISO IN TPDU with leng th= 0.
- **Case 2S**
  Short format: no command data, but with response data of 1–255 bytes in size. This is sent as a T = 0 ISO OUT TPDU.
- **Case 3S**
  Short format: command data of 1–255 bytes in size and no response data. This is sent as a T = 0 ISO IN TPDU.
- **Case 4S**
  Short format: command data of 1–255 bytes in size, with response data of 1–256 bytes in size. The command is sent as a T = 0 ISO IN TPDU. It must be followed by a **Get Response** command sent as a T = 0 ISO OUT TPDU. The **Get Response** mechanism is compliant with the ISO 7816-4 standard.

These cases are referred to as 1, 2S, 3S and 4S respectively. MPCOS-EMV does not accept commands in any other format and it responds with the status 67 00h.

Command Format

In the default configuration (T = 0), MPCOS-EMV cards accept commands in the following format:

Header				Body		
CLA	INS	P1	P2	Lc	Parameters/Data	Le

The fields are described as follows:

Header Fields

The header fields, which are mandatory, are as follows:

Field Name	Length	Description
CLA	1 byte	Instruction class. This can be any of the following values:
		00h ISO 7816-4/EMV compatible command
		04h ISO 7816-4/EMV compatible command with secure messaging
		80h Proprietary command
		84h Proprietary command with secure messaging
INS	1 byte	Instruction code. This is given with the command descriptions.
P1	1 byte	Parameter 1
P2	1 byte	Parameter 2

Body Fields

The command body is optional. It includes the following fields:

Field Name	Length	Description
Lc	1 byte	Data length
Data	n bytes	Command parameters or data
Le	1 byte	Expected length of data to be returned

Response Format

MPCOS-EMV cards return responses to commands in the following format:

Body	Trailer
Data	SW1, SW2

The **Body** field is optional and holds the data returned by the card.

The **Trailer** field includes the following two mandatory bytes:

- SW1: Status byte 1 that returns the command processing status
- SW2: Status byte 2 that returns the command processing qualification

See *"Appendix B - Card Contact Interface Return Codes"* for a list of the values that MPCOS-EMV cards can return in the status bytes.

Note: In the following sections (commands description), commands are described in APDU format (T = 0) only, according to the default card protocol.

9

Commands

This section describes the low-level MPCOS-EMV card commands. The commands are divided into the following broad categories:

- Administration commands. In addition to the ISO commands, the administration commands also include a complementary set of commands which perform administration task (for example, file creation) that are not defined by ISO.
- Payment commands
- EMV commands

A comprehensive description of each command is given in the following sections.

Administration Commands

The administration commands constitute the kernel of MPCOS-EMV. The commands include functions for creating a file, reading a record and updating a record, as follows:

Command	Full Name	Description	Page
ApdRec	Append Record	Appends a new record to a structured file.	68
CrtFil	Create File	Creates an elementary or a dedicated file.	75
ExtAut	External Authenticate	Causes the card to check a cryptogram sent from the terminal.	86
Freeze AC	Freeze Access Conditions	Locks or localizes a file access condition.	88
GetChal	Get Challenge	Generates an eight-byte random number.	99
GetInfo	Get Card Information	Retrieves miscellaneous information (for example, card and files).	95

Command	Full Name	Description	Page
GetResp	Get Response	Retrieves and erases the data prepared in RAM by the card in response to the previous command.	104
IntAut	Internal Authenticate	Causes the card to compute a cryptogram for verification by the terminal.	107
RdBin	Read Binary	Reads data from transparent elementary files.	111
RdMem	Read Memory	Reads the card serial number of a card that has been personalized.	113
RdRec	Read Record	Reads data from a structured file.	114
SelFil	Select File	Selects an elementary file or a dedicated file for use in transactions.	116
SelFk	Select File Key	Computes a temporary administration key and an authentication cryptogram.	119
Set Card Status	Set Card Status	Sets the personalization flag to one once card personalization has been completed and sets the size of the data units.	125
SetCod	Set Secret Code	Unblocks or changes a secret code.	128
SwtPrt	Switch Protocol	Switches the card to another speed.	134
UpdBin	Update Binary	Updates data in an elementary file.	137
UpdRec	Update Record	Updates data from a structured file.	141
Verify	Verify	Compares the value submitted to the secret code number (from 0 to 7) in the secret code file pertaining to currently selected dedicated file.	143
WrBin	Write Binary	Writes data to an elementary file by performing a logical OR operation between the current value of the write area and the value to be written.	145

Table 12 - Administration Commands

Payment Commands

The payment commands are used to perform payment tasks. They include functions for crediting a purse, debiting a purse and reading a purse balance, as follows:

Command	Full Name	Description	Page
CanDeb	Cancel Debit	Cancels the previous debit performed by a terminal and replaces it by a new debit.	71
Cdt	Credit	Credits a purse.	80
Dbt	Debit	Debits a purse.	83
RdBal	Read Balance	Reads the specified purse balance value.	109
SelPK	Select Purse & Key	Selects the specified purse and key, then generates a new temporary payment transaction key and an authentication cryptogram.	122
SetOpts	Set Options	Sets the following **Sign** command options: • Use current purse balance in the certificate calculation. • Clear the RAM parameters after executing the **Sign** command.	126
Sign	Sign	Generates a certificate for the previous transaction.	131

Table 13 - Payment Commands

EMV Commands

The EMV commands support VISA Personalization Template 1 - Magnetic Stripe Image.

Command	Full Name	Description	Page
Generate AC	Generate Application Cryptogram	Requests the card to provide a cryptogram that indicates the card's authorization response	91
Get Data	Get Data	Retrieves the Card Production Life Cycle (CPLC) or the Application Transaction Counter (ATC).	100
Get Processing Options	Get Processing Options	Indicates to the card that transaction processing is beginning.	102

Table 14 - EMV Commands

Append Record (ApdRec)

Use this command to format a structured file by appending (and initializing) a new record.

Note: A maximum of 255 records can be appended to linear fixed files and linear variable files; a maximum of 254 records to cyclic files.

Files can be selected directly or implicitly. When selected directly, the file must have been previously selected using the **Select File** command. In an implicit selection, the file is selected in the current Dedicated File (DF), using the Short File Identifier (SFI). If the selection is successful, the selected file becomes the current file and the record pointer will be cleared.

Note: In cyclic files, a new record can be created only if the current record corresponds to the last created record. If a new record is created (in normal processing), this record will become the current record and the record pointer will be set accordingly.

The **Append Record** command does not cycle; that is, after appending the last record of the file, no more records can be appended.

Warning: If an **Update Record** command has been performed since the last **Append Record** command, the current record will no longer be the last created record.

Secure Messaging

This command can use secure messaging, as long as the record length and the certificate length do not exceed the internal buffer size (see *"IO Buffer Size"* on page 53).

See *"Secure Messaging"* on page 41 for further information on the conditions which must be met before secure messaging can be used.

Format

CLA	INS	P1	P2	Lc	Data	Le
00h/04h	E2h	00h	P2	Lc	Data	00h/03h

Where:

CLA	Specifies transmission mode (normal or secure messaging):	
	00h	Transmit normally.
	04h	Transmit with secure messaging.

P2 — Specifies the file to which the record is to be appended. In a direct selection, the file must have been previously selected using the **Select File** command and P2 is set to zero. In an implicit selection, the file is selected in the current DF, using the SFI and the value is sent in the following format:

Bit	7	6	5	4	3	2	1	0
			SFI			0	0	0

Lc — Specifies the length of the record, which is to be appended. If secure messaging is being used, an additional three bytes should be added.

Le		
	00h	No response is expected, if secure messaging is not required during the transmission.
	03h	Three bytes of response message, if secure messaging is required during the transmission.

Response

Response	SW1	SW2

Where:

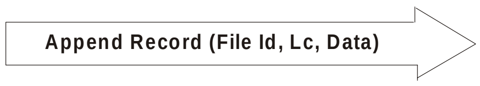
Response — If secure messaging is required, the card generates a cryptographic response (see *“Secure Messaging”* on page 41). The terminal retrieves the three leftmost significant bytes of the eight-byte checksum by executing the **Get Response** command (see *“Get Response (GetResp)”* on page 104).

SW1 and SW2 — Hold the status byte values returned by the card. *“Appendix B - Card Contact Interface Return Codes”* lists the possible contents of SW1 and SW2.

Command Processing

“Figure 32 - Command Processing for Append Record” shows a detailed procedure of the command processing.

Terminal **Card**



1. Appends a record with the contents “DATA” to the specified file.
2. Prepares the response message only for transmission with secure messaging.

CRYCKS 7	CRYCKS 6	CRYCKS 5

CRYCKS$_{7–5}$ are the three leftmost significant bytes of the eight-byte cryptographic checksum.

Terminal		**Card**
3. Issues a **Get Response** command to retrieve the response message. This applies only to transmission with secure messaging.	→	
	←	4. Sends the response message.

Figure 32 - Command Processing for Append Record

Cancel Debit (CanDeb)

Use this command to cancel the previous debit performed on the card. Optionally, it can be used to replace the previous debit by a new value.

Note: This command can be disabled during the personalization step when creating the corresponding Dedicated File (DF). It cannot be used if a **Credit** command has been executed since the last **Debit** transaction.
The MRN mechanism will be activated if the MRN value is non-zero. See *"Maximum Replay Number (MRN) Mechanism"* on page 46.

Before executing the **Cancel Debit** command, the card checks that the following conditions are met:

- The parent dedicated file option byte (OPT) allows the command to be executed. See *"File Descriptor"* on page 10.
- A Temporary Payment Transaction Key (Kpts) was established. See *"Select Purse & Key (SelPK)"* on page 122.
- The debit to be canceled must be performed by the same terminal which is executing the current **Cancel Debit** command. For this purpose, the card checks that the Terminal Transaction Counter (TTC) value transmitted by the terminal to cancel the debit is consistent with the TTC value stored in the card during the previous debit session.
- The debit to be canceled is consistent with the last debit performed with the card.

When a replacement debit is requested, the card cancels the previous debit and then debits the specified replacement value from the purse. Before executing this step, the card checks that the following criteria are met:

- The debit access condition (Cd) has been fulfilled if the debit value is greater than the maximum free debit value. See *"Purse Files"* on page12.
- The debit value is less than or equal to the current balance.

"Table 15 - Debit Access Condition" summarizes the debit access conditions.

	Debit Value ≤ Maximum Free Debit	**Debit Value > Maximum Free Debit**
Cd = 0	Yes	Yes
0 < Cd < 8	Yes	Yes if Cd satisfied
Cd > 7	No	No

Table 15 - Debit Access Condition

The Debit Access Condition (Cd) and the Maximum Free Debit Value are described in *"Purse Files"* on page 12.

Format

CLA	INS	P1	P2	Lc	Data	Le
80h	46h	00h	P2	0Ch	Data	02h

Where:

Data — Specifies the last debit value and the Cancel Debit Cryptogram (SDC). This value has the following format:

00h	Dv2	Dv1	Dv0	SDC7	SDC6	SDC5	SDC4	SDC3	SDC2	SDC1	SDC0

Dv2 to Dv0	Hold the last debit value.
SDC7 to SDC0	Hold the Cancel Debit Cryptogram (SDC).

The terminal generates the SDC as follows for debit cancellation:

$SDC=3DES^{-1}[$

00h	00h	00h	00h	TTC3	TTC2	TTC1	TTC0

$]$ <Kpts>

If a replacement debit is requested, the terminal generates the SDC as follows:

$SDC=3DES^{-1}[$

00h	Rv2	Rv1	Rv0	TTC3	TTC2	TTC1	TTC0

$]$ <Kpts>

Rv2 to Rv0	The replacement debit value (this is set to 00 for debit cancellation).
TTC3 to TTC0	Terminal Transaction Counter value. The two leftmost significant bytes of the TTC contains the terminal serial number; they are compared with the TTC value stored in the purse during the previous debit to verify that the terminal is the one used to perform the original debit.

The two rightmost significant bytes of the TTC are used as the incremental counter. It is incremented for each cryptogram computation that is not checked by the **Cancel Debit** command. TTC is managed by the terminal.

Note: If an error is encountered during the execution of **Cancel Debit** command, the terminal will only cancel the debit, regardless of either debit cancellation or replacement debit is required.

Warning: The **Cancel Debit** command will not cancel a debit, which has already been replaced.

Response

Response	SW1	SW2

Where:

Response	Holds the two leftmost significant bytes of the eight-byte Cancel Debit Certificate (CDC$_{7-6}$). This can be retrieved by the terminal using **Get Response** command.

If a debit cancellation is requested, the card generates the CDC as:

CDC=3DES[	00h	Pf	00h	00h	00h	TTC2	TTC1	TTC0	] <Kpts>

If a replacement debit is requested, the card generates the CDC as:

CDC=3DES[	00h	Pf	Rv2	Rv1	Rv0	TTC2	TTC1	TTC0	] <Kpts>

Pf is the Short File Identifier (SFI) of the purse on which the **Cancel Debit** was executed. The identifier has the following format:

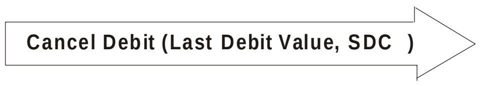
Bit	7	6	5	4	3	2	1	0
	0	0	0	SFI of Purse File				

SW1 and SW2	Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

"Figure 33 - Command Processing for Cancel Debit" shows a detailed procedure of the command processing.

Terminal **Card**



1. Generates the SDC as:
   - For debit cancellation:
     SDC=3DES^{-1} [00h||00h||
     00h||00h||TTC$_{3-0}$] <Kpts>
   - For debit replacement:
     SDC=3DES^{-1} [00h||Rv$_{2-0}$||
     TTC$_{3-0}$] <Kpts>

2. Decrypts SDC to retrieve Rv$_{2-0}$ and TTC$_{3-0}$ as:
   3DES [00h||00h||00h||00h||TTC$_{3-0}$]<Kpts>
   or
   3DES [00h|| Rv$_{2-0}$ ||TTC$_{3-0}$] <Kpts>

Terminal | **Card**

3. Checks the following:
   - Last debit value + current balance = previous balance
   - TTC is compatible
   - Replacement debit value ≤ Previous balance
   - Checks the debit access condition has fulfilled, if the replacement debit value > the maximum free debit
4. Computes the new balance.
5. Computes the Cancel Debit Certificate (CDC) as:
   - For debit cancellation:
     CDC = 3DES [00h||Pf||00h||00h||00h|| TTC$_{2-0}$]<Kpts>
   - For debit replacement:
     CDC = 3DES [00h||Pf||Rv$_{2-0}$||TTC$_{2-0}$] Kpts>
6. Prepares the response message, the two leftmost significant bytes of CDC.

7. Issues a **Get Response** command to retrieve the response message. $\longrightarrow$

$\longleftarrow$ 8. Sends the response message.

9. Computes its own CDC and checks with the CDC retrieve from the card.

Figure 33 - Command Processing for Cancel Debit

Create File (CrtFil)

Use this command to create an Elementary File (EF) or a Dedicated File (DF).

The access condition that governs the file creation, is in the file descriptor of the current DF or the Master File (MF). The Group 1 access condition applies to sensitive files; the Group 2 access condition applies to data files.

The access condition contains a Short File Identifier (SFI) of a key file (see *“Chapter 3 - Access Conditions”*). If this SFI is zero, normal processing will be used; otherwise, secure messaging will be used. In the normal mode, the only data that is sent to the card is the data needed to initialize the descriptor (12 bytes), plus (if creating a DF) a name of not more than 16 additional bytes.

Secure Messaging

See *“Secure Messaging”* on page 41 for further information on the conditions which must be met before secure messaging can be used.

The data sent to the card will be the same as for normal processing in clear text plus an additional cryptographic checksum of three bytes. This checksum is calculated using the data field plus the command header.

When the card receives the command, it verifies that the cryptographic checksum matches the command and data. If they do not match, the card will not access to the command and it will return an appropriate error code. If they do match, the card will create the file, provided that no other errors are detected.

Format

CLA	INS	P1	P2	Lc	Data	Le
80h/84h	E0h	P1	00h	Lc	Data	00h/03h

Where:

CLA	Specifies transmission mode (normal or secure messaging):	
	80h	Transmit normally.
	84h	Transmit with secure messaging.
P1	Specifies the type of file to be created.	
	01h	Dedicated File (DF)
	02h	Elementary File (EF)
Lc	Specifies the data field length.	
	0Ch	For creating EF.
	0Ch to 1Ch	For creating DF.

Note: A value of 03h should be added if secure messaging is in use.

Data

Holds the data field sent to the card. The format and the contents of this data depend on the file type being created—a DF or an EF.

To create a DF, the data field is as follows:

File Identifier	FDB	OPT	Name Length	Group 1 AC

Group 2 AC	MRN	RFU	DF name (1–16 bytes)

File Identifier — Holds a two-byte file identifier. Under ISO 7816-4, the MF has the identifier 3F00h.

FDB — Holds the File Descriptor Byte, which determines the type of file and associated options. This must hold the following value:

Bit	7	6	5	4	3	2	1	0
	0	0	1	1	1	0	0	0

OPT — Holds the file option byte. This is as follows:

7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X

“Table 16 - Dedicated File Option Byte” shows the options indicated by each bit:

Bit	Value	Option Indicated
0		Not used.
1	1	**Cancel Debit** command disabled.
	0	**Cancel Debit** command enabled.
2	1	Current balance can be used to compute Signature certificates, provided that this option is not reversed using **Set Options** (see *“Set Options (SetOpts)”* on page 126).
	0	Current balance can never be used to compute **Signature** certificates (see *“Sign”* on page 131).
3–6		RFU

Table 16 - Dedicated File Option Byte

Bit	Value	Option Indicated
7	1	**Select Purse & Key and Select File Key** commands depend on an external authentication.
	0	**Select Purse & Key** and **Select File Key** commands are free.

Table 16 - Dedicated File Option Byte

Name Length	Specifies the length of DF Name in bytes. This can be any value from 1 to 16.
Group 1, 2 AC	Hold the access conditions for Group 1 and Group 2, respectively. See *"Chapter 3 - Access Conditions"* for details on access conditions.
MRN	Maximum Reply Number. See *"Maximum Replay Number (MRN) Mechanism"* on page 46 for details.
DF Name	Holds the dedicated file name. This is a string of up to 16 bytes. This field is optional.

To create an elementary file, the data field is as follows:

File Identifier	FDB	RecLen	Body Size	Group 1 AC

Group 2 AC	Group 3 AC

Warning: Bits 7 and 6 must be set to 00b for MPCOS-EMV R4. This is not compatible with MPCOS-3DES, which requires it to be set to 11b.

File Identifier — Holds a two-byte file identifier. The SFI corresponds to the five least significant bits.

Bit	7	6	5	4	3	2	1	0
	0	0	X	X	X	X	X	X

Bits 5 to 3: Define the file type, for example, key file.

Bits 2 to 0: Define the structure of the file, for example, cyclic file.

For more details, see *"File Descriptor"* on page 10.

RecLen	Specifies the record length for linear fixed and cyclic files only. It is not valid for other structure of EFs and is normally set to zero.
Body Size	Specifies the file body length in bytes. For EFs, the body corresponds to the data constituting the file.
Group 1, 2, 3 AC	Hold the access conditions for Groups 1, 2 and 3 respectively. See *“Chapter 3 - Access Conditions”* for details about access conditions.

Response

Response	SW1	SW2

Where:

Response	If secure messaging is used, the card generates a cryptographic response (see *“Secure Messaging”* on page 41) The terminal retrieves the three leftmost significant bytes of the eight-byte checksum by executing the **Get Response** command.
SW1 and SW2	Hold the status byte values returned by the card. *“Appendix B - Card Contact Interface Return Codes”* lists the possible contents of SW1 and SW2.

Command Processing

“Figure 34 - Command Processing for Create File” shows a detailed procedure of the command processing.

Terminal **Card**

Create File (File Type, Data)

1. Creates a DF or EF depending on the file type specified in P1. The variables for the file descriptor to be created are contained in Data.
2. Prepares the response message only for transmission with secure messaging.

CRYCKS 7	CRYCKS 6	CRYCKS 5

CRYCKS$_{7-5}$ are the three leftmost significant bytes of the eight-byte cryptographic checksum.

Terminal		**Card**
3. Issues a **Get Response** command to retrieve the response message. This applies only to transmission with secure messaging.	→	
	←	4. Sends the response message.

Figure 34 - Command Processing for Create File

Credit (Cdt)

Use this command to credit a specified value into a purse.

Before executing the **Credit** command, the following conditions must be fulfilled:

- The **Select Purse & Key** command must have been previously executed to select a purse and to establish a Temporary Payment Transaction Key (Kpts).
- The **Debit** has not been executed since the last **Select Purse & Key** command.
- If the purse to be credited is an enhanced purse, the credit secret code (if any), is successfully submitted.
- The credit amount does not cause the purse to exceed the maximum allowable balance.

Note: Upon each execution of this command, the values of the Off-Nominal Counter (ONC) at the session key for all keys referenced will be restored and the ONC mechanism will be disabled.

If the incoming cryptogram is wrong, the ONC value will be decremented (if the ONC has been enabled). See *"Off-Nominal Counter (ONC) Mechanism"* on page 45 for details.

Format

CLA	INS	P1	P2	Lc	Data	Le
80h	36h	P1	00h	0Ch	Data	00h

Where:

P1 Kn Specifies the number of the credit key in the Credit Key (Kc) in the relevant key file to be used to compute the credit cryptogram. The key number is sent in the following format:

Byte	7	6	5	4	3	2	1	0
	Key Number							0

Note: The key number of a 3DES key must be an even number. The Credit Key (Kc) is either a payment key or a signature key.

Data Specifies the credit value and the credit cryptogram (CC") encrypted by the key Kpts. This data has the following format:

00h	Cv2	Cv1	Cv0	CC"7	CC"6	CC"5	CC"4	CC"3	CC"2	CC"1	CC"0

Cv2–Cv0	Credit value in three bytes.

CC"7–CC"0	Credit cryptogram. Either the host, the terminal or both generate this using the following procedure:

1. Compute a temporary key (Kt) based on the credit key, Kc, (whose number is indicated in the P1 parameter) and the Card Transaction Counter (CTC, which has been retrieved by the terminal during the previous **Select Purse & Key** command) as follows:
   Kt = 3DES_16 [CTC] <Kc>
2. Generate a Credit Cryptogram (CC) using the following data:

CC =								
	00h	Pf	Cv2	Cv1	Cv0	00h	00h	00h

   PF specifies the Short File Identifier (see *"File Descriptor"* on page 10) of the purse file to be credited, in the following format:

Bit	7	6	5	4	3	2	1	0
	0	0	0	SFI of Purse File				

3. Computes the Encrypt the Credit Cryptogram (CC') as follows:
   CC'=3DES^{-1}[CC] <Kt>
4. Encrypt CC' with the temporary payment transaction key (Kpts) as follows:
   CC"=3DES^{-1}[CC] <Kpts>

Note: The temporary credit key (Kt) used by the card to validate the credit cryptogram can be different from the temporary key used for card/terminal authentication. This enables an on-line host to perform the **Credit** operation using a Credit Key (Kc) that is not known by the terminal.

Response

SW1	SW2

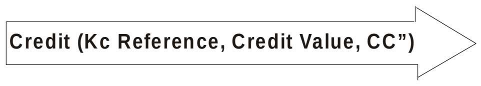
Where:

SW1 and SW2	Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

"Figure 35 - Command Processing for Credit" shows a detailed procedure of the command processing.

Terminal **Card**



1. Computes a Temporary Credit Key (Kt):
   Kt = 3DES_16 [CTC] <Kc>
2. Generates a Credit Cryptogram (CC):

00h	Pf	Cv2	Cv1	Cv0	00h"	00h	00h

3. Encrypts a CC using Kt:
   CC' = $3DES^{-1}$ [CC] <Kt>
4. Encrypts a CC' using Kpts:
   CC" = $3DES^{-1}$ [CC'] <Kpts>
5. In an enhanced purse, checks the credit conditions are fulfilled.
6. Decrypts the CC" using Kpts:
   CC' = 3DES [CC"] <Kpts>
7. Decrypts the CC' using Kt:
   CC = 3DES [CC'] <Kt>
8. Verifies the consistency of CC.
9. Updates the balance.

Figure 35 - Command Processing for Credit

Debit (Dbt)

Use this command to debit a purse.

Before executing the **Debit** command, the following conditions must be fulfilled:

- The **Select Purse & Key** command must have been previously executed to select a purse and to establish a Temporary Payment Transaction Key (Kpts).
- The debit access condition (Cd) must be fulfilled if the debit value is greater than the maximum free debit value (see *"Purse Files"* on page12).
- The debit value is less than or equal to the current balance.

"Table 17 - Debit Access Condition" summarizes the debit access conditions.

	Debit Value ≤ Maximum Free Debit	**Debit Value > Maximum Free Debit**
Cd = 0	Yes	Yes
0 < Cd < 8	Yes	Yes if Cd satisfied
Cd > 7	No	No

Table 17 - Debit Access Condition

The Debit Access Condition (Cd) and the Maximum Free Debit Value are described in *"Purse Files"* on page 12.

Note: *The MRN mechanism will be activated if the MRN value is non-zero. See "Maximum Replay Number (MRN) Mechanism"* on page 46 *for details.*

Format

CLA	INS	P1	P2	Lc	Data	Le
80h	34h	00h	P2	08h	Data	02h/06h

Where:

P2	P2 = 00h	For a response of 02h in length, representing the two leftmost significant bytes of the CDC.
	P2 = 01h	For a response of 06h in length, representing the six leftmost significant bytes of the CDC.
Data	Specifies the last debit value and the Terminal Transaction Counter (TTC). This value has the following format:	

00h	Dv2	Dv1	Dv0	TTC3	TTC2	TTC1	TTC0

Dv2 to Dv0	Debit value in three bytes.
TTC3 to TTC0	Terminal Transaction Counter (TTC) value. The two leftmost significant bytes of the TTC contain the terminal serial number are compared with the TTC value stored in the purse during the **Cancel Debit** command, to verify that the terminal is the one used to perform the original debit. The two rightmost significant bytes of the TTC are used as the incremental counter. It is incremented for each cryptogram computation that is not checked by the **Cancel Debit** command. TTC is managed by the terminal.

Le Response length dependent on the values in P2.

Response

Response	SW1	SW2

Where:

Response Holds the two or six leftmost significant bytes of the eight-byte Card Debit Certificate (CDC_{7-6} or CDC_{7-2}, respectively). This can be retrieved by the terminal using **Get Response** command.

The card generates the CDC as:

CDC=3DES[	00h	Pf	Dv2	Dv1	Dv0	TTC2	TTC1	TTC0	] <Kpts>

Pf is the Short File Identifier (SFI) of the purse on which the **Debit** command was executed. The identifier has the following format:

Bit	7	6	5	4	3	2	1	0
	0	0	0	SFI of Purse File				

Dv2 to Dv0	Debit value
TTC2 to TTC0	Terminal Transaction Counter (TTC) value
Kpts	Temporary payment transaction key

SW1 and SW2 Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

“Figure 36 - Command Processing for Debit” shows a detailed procedure of the command processing.

Terminal **Card**

Debit (Debit Value, TTC) →

1. Checks the following:
   – Checks the debit access condition has fulfilled, if the replacement debit value is greater than the maximum free debit.
   – Replacement debit value is less than or equal to the previous balance.
2. Performs the debit and updates the balance value.
3. Computes a Card Debit Certificate (CDC) as:

CDC=3DES[	00h	Pf	Dv2	Dv1	Dv0	TTC2	TTC1	TTC0	] <Kpts>

4. Prepares the response message, the two or six leftmost significant bytes of CDC.
5. Issues a **Get Response** command to retrieve the response message. →
6. ← Sends the response message.
7. Computes its own CDC and checks with the CDC retrieve from the card.

Figure 36 - Command Processing for Debit

External Authenticate (ExtAut)

Use this command to authenticate the terminal.

The terminal must have retrieved an eight-byte random value from the card using a **Get Challenge** command prior to the external authentication. Any other commands, except **External Authenticate**, is issued after **Get Challenge**, the eight-byte random value will be lost.

The terminal calculates the External Authenticate Cryptogram (EAC), using the random value retrieved from the card and an authentication key. It then sends the string of data that contains the reference of the authentication key and the four leftmost significant bytes of the EAC (that is, EAC_{7-4}) to the card.

The card calculates its own cryptogram and compares it with the $EAC_{7–4}$.

Note: Upon each execution of this command, the values of the Off-Nominal Counter (ONC) at the session key for all keys referenced are restored back and the ONC mechanism is disabled.
If the authentication fails, the ONC value will be decremented (if the ONC has been enabled). See *"Off-Nominal Counter (ONC) Mechanism"* on page 45 for details.

Authentication rights will be lost after any of the following events:

- Card reset
- Authentication failure
- DF selection (unless from the MF)
- Data integrity check

Format

CLA	INS	P1	P2	Lc	Data	Le
00h	82h	00h	P2	06h	Data	00h

Where:

P2 Specifies the level of the relevant file.

Bit	7	6	5	4	3	2	1	0
	L	X	X	X	X	X	X	X

L Specifies the level of the relevant file that is holding the authentication key.
0: Global level
1: Local level

Data String of data contains the reference of the relevant file that stored the authentication key and the four leftmost significant bytes of the eight-byte EAC.

Kn	Kf	EAC7	EAC6	EAC5	EAC4

Kn Specifies the authentication key number and it is coded as:

Bit	7	6	5	4	3	2	1	0
	Key Number							0

Kf Specifies the Short File Identifier (SFI) of the relevant file that is holding the authentication key and it is coded as:

Bit	7	6	5	4	3	2	1	0
	0	0	0	SFI				

EAC_{7-4} The four leftmost significant bytes of the eight-byte EAC computed by the terminal.

Response

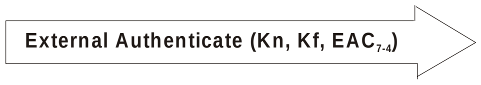
SW1	SW2

SW1 and SW2 Hold the status byte values returned by the card. *“Appendix B - Card Contact Interface Return Codes”* lists the possible contents of SW1 and SW2.

Command Processing

“Figure 37 - Command Processing for External Authenticate” shows a detailed procedure of the command processing.

Terminal **Card**



1. Calculates the $EAC_{Terminal}$ using the authentication key and the random number generated by the card in the **Get Challenge** command.
$EAC_{Terminal}$ = 3DES [CRnd] <Kaut>

2. Verifies that the challenge random value (that is, card random number) is available.
3. Calculates the mirror cryptogram EAC_{Card} using the authentication key and its random number generated in the **Get Challenge** command.
EAC_{Card} = 3DES [CRnd] <Kaut>
4. Verifies that $EAC_{Terminal} = EAC_{Card}$
The authentication is successful if both EAC equal.

Figure 37 - Command Processing for External Authenticate

Freeze Access Conditions (Freeze AC)

Use this command to modify a file access condition. For example, when the original access conditions defined at the time of file creation need to be changed for the application phase.

This command can work in one of the following modes:

Lock mode	It locks an access condition so that it denies access at all times. This mode might be used to make a file read-only after personalization.
Localize mode	It moves key or secret code references from the global EF_{key} and EF_{sc} to the local EF_{key} and EF_{sc}. Once this has been done, you cannot perform another **Freeze Access Conditions** command on the same file except in lock mode. This mode is useful for initializing an application.

Note: To localize the access conditions of EF_{key} and EF_{sc}, the files must exist locally and contain the required keys and secret codes. Otherwise, the localized access condition may become locked.
See *"Chapter 3 - Access Conditions"* for further information.

Secure Messaging

If the key file indicator in the DF descriptor, each of Group 1 and Group 2 access conditions for "*Freeze AC*" contains a value other than zero, the command must be sent with secure messaging.

See *"Secure Messaging"* on page 41 for further information on the conditions which must be met before secure messaging can be used.

In secure messaging mode, the card receives the data in the same way as for normal processing but an additional three-byte cryptographic checksum is sent.

When the card receives the command, it verifies that the cryptographic checksum matches the command and data. If they do not match, the command will be rejected and the card will returns an appropriate error code.

Format

CLA	INS	P1	P2	Lc	Data	Le
CLA	16h	P1	00h	Lc	Data	Le

Where:

CLA	Specifies transmission mode (normal or secure messaging):	
	80h	Transmit without secure messaging
	84h	Transmit with secure messaging.
P1	Mode	Specifies the type of file to be created.
	01h	MF or DF
	02h	EF

Lc	Specifies the data field length.	
	05h	Transmit without secure messaging
	08h	Transmit with secure messaging.

Data Specifies the file identifier and access condition modifications to be made. It has the following format:

Identifier	G1	G2	G3

Identifier	Specifies the identifier of the file to be frozen.
G1, G2, G3	Control bytes for the Group 1, Group 2 and Group 3 access conditions. They describe how each access condition can be modified.

Each control byte has the following format:

Bit	7	6	5	4	3	2	1	0
	L		K		SC1		SC2	

"Table 18 - Control Byte Structure" shows how the control byte contents modify their related access conditions (the values below are in binary):

L	K	SC1	SC2	Modification
01	xx	xx	xx	Locks access condition.
00	00	xx	xx	No change to secure messaging key.
00	01	xx	xx	Localize secure messaging key.
00	xx	00	xx	No change to secret code 1.
00	xx	01	xx	Localize secret code 1.
00	xx	xx	00	No change to secret code 2.
00	xx	xx	01	Localize secret code 2.

Table 18 - Control Byte Structure

Le	00h	No response is expected, if transmission without secure messaging.
	03h	Three bytes of response message, if transmission with secure messaging.

Response

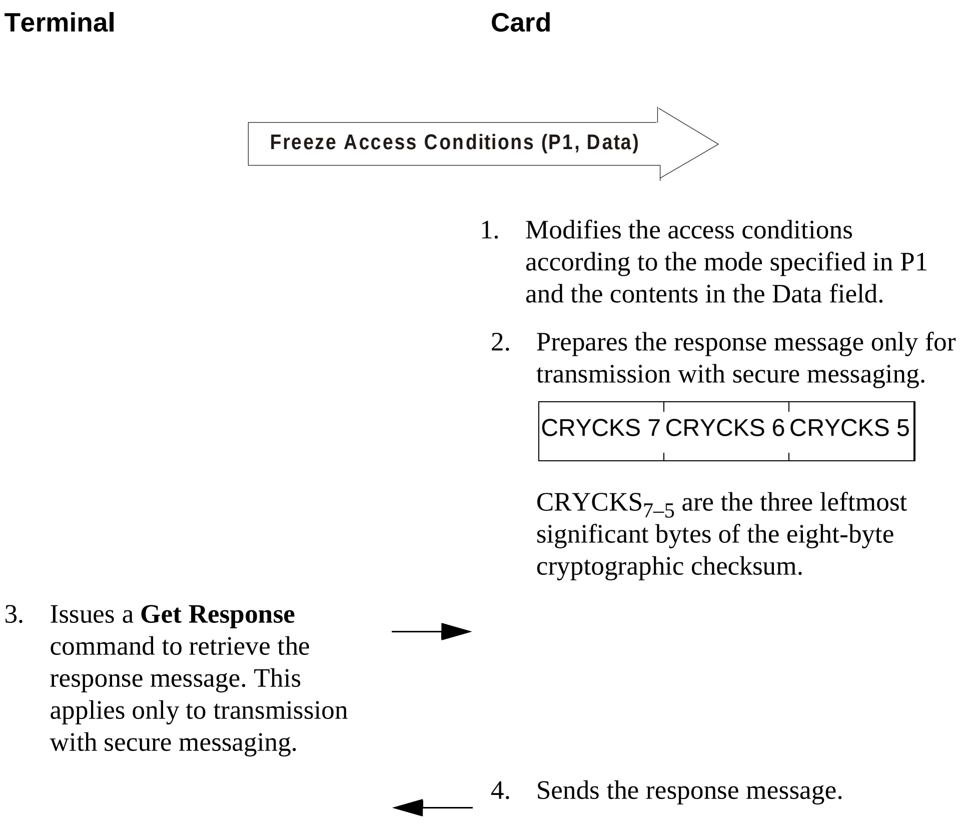
Response	SW1	SW2

Where:

Response	If secure messaging is used, the card generates a cryptographic response (see *"Secure Messaging"* on page 41). The terminal retrieves the three leftmost significant bytes of the eight-byte checksum by executing the **Get Response** command.
SW1 and SW2	Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

"Figure 38 - Command Processing for Freeze Access Conditions" shows a detailed procedure of the command processing.



1. Modifies the access conditions according to the mode specified in P1 and the contents in the Data field.
2. Prepares the response message only for transmission with secure messaging.

CRYCKS 7	CRYCKS 6	CRYCKS 5

$CRYCKS_{7-5}$ are the three leftmost significant bytes of the eight-byte cryptographic checksum.

3. Issues a **Get Response** command to retrieve the response message. This applies only to transmission with secure messaging.
4. Sends the response message.

Figure 38 - Command Processing for Freeze Access Conditions

Generate Application Cryptogram (Generate AC)

Use this command to request the card to provide a cryptogram indicating the card's authorization response. This authorization response is either completing or rejecting the transaction offline, or to request the transaction to go online. This command can be issued at most twice for each transaction.

Note: The **Generate Application Cryptogram** command can only be invoked under ADFs that are properly initialized.

The command can only be used if the **Get Processing Options** command has been successfully performed within an application.

The second **Generate Application Cryptogram** command can only be invoked after the first **Generate AC** command returns ARQC cryptogram.

Format

CLA	INS	P1	P2	Lc	Data	Le
80h	AEh	P1	00h	Lc	Data	00h

Where:

P1 Reference Control Parameter, coded as follows:

Bits								Cryptogram
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
0	0	X	X	X	X	X	X	AAC
0	1	X	X	X	X	X	X	TC
1	0	X	X	X	X	X	X	ARQC
1	1	X	X	X	X	X	X	Not supported

X = Reserved for Future Use

AAC	Application Authentication Cryptogram (AAC): transaction declined
TC	Transaction Certificate: transaction approved
ARQC	Authorization Request Cryptogram (ARQC): online authorization requested (cannot be issued in the second **Generate Application Cryptogram** command)

Lc Length of the subsequent data field

Data Transaction-related data, coded as follows:

	Data Field	**Lc**
First Generate AC	Terminal Verification Results (5)	05h
Second Generate AC	Authorization Response Code	02h

TVR Terminal Verification Results. TVR in the first **Generate AC** is actually not processed by the card. However, Lc field must be equal to 5.

ARC Authorization Response Code. Indicates the disposition of the transaction. The possible values are as follows:

Values in ASCII (2 bytes)	**Meaning**
’Y1’ or ’00’	Offline approved
’Z1’	Offline declined
’Y3’	Unable to go online (offline approved)
’Z3’	Unable to go online (offline declined)

Any values other than listed above are considered not approved online (Offline Declined).

The data field corresponds to CDOL1 and CDOL2, each of which indicates the terminal data objects that are needed for the card risk management.

Response

Response	SW1	SW2

Where:

Response Data field retrievable by a **Get Response** command.

’80’	’12’	CID (1)	ATC (2)	AC (8)
IAD len (1)	DKI (1)	CVN (1)	CVR (4)	

CID	Cryptogram Information Data, dependent on the processing logic, as follows:

Bits								**Cryptogram**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
0	0							AAC
0	1							TC
1	0							ARQC
1	1							Not supported

Values not indicated in this table are not supported in this specification.

ATC	Value from the Application Transaction Counter
AC	A random number
IAD len	06h
DKI	00h
CVN	0Ch (proprietary cryptogram)
CVR	Card Verification Result. Visa proprietary data elements indicating the exception conditions that occurred during a transaction, coded as follows:

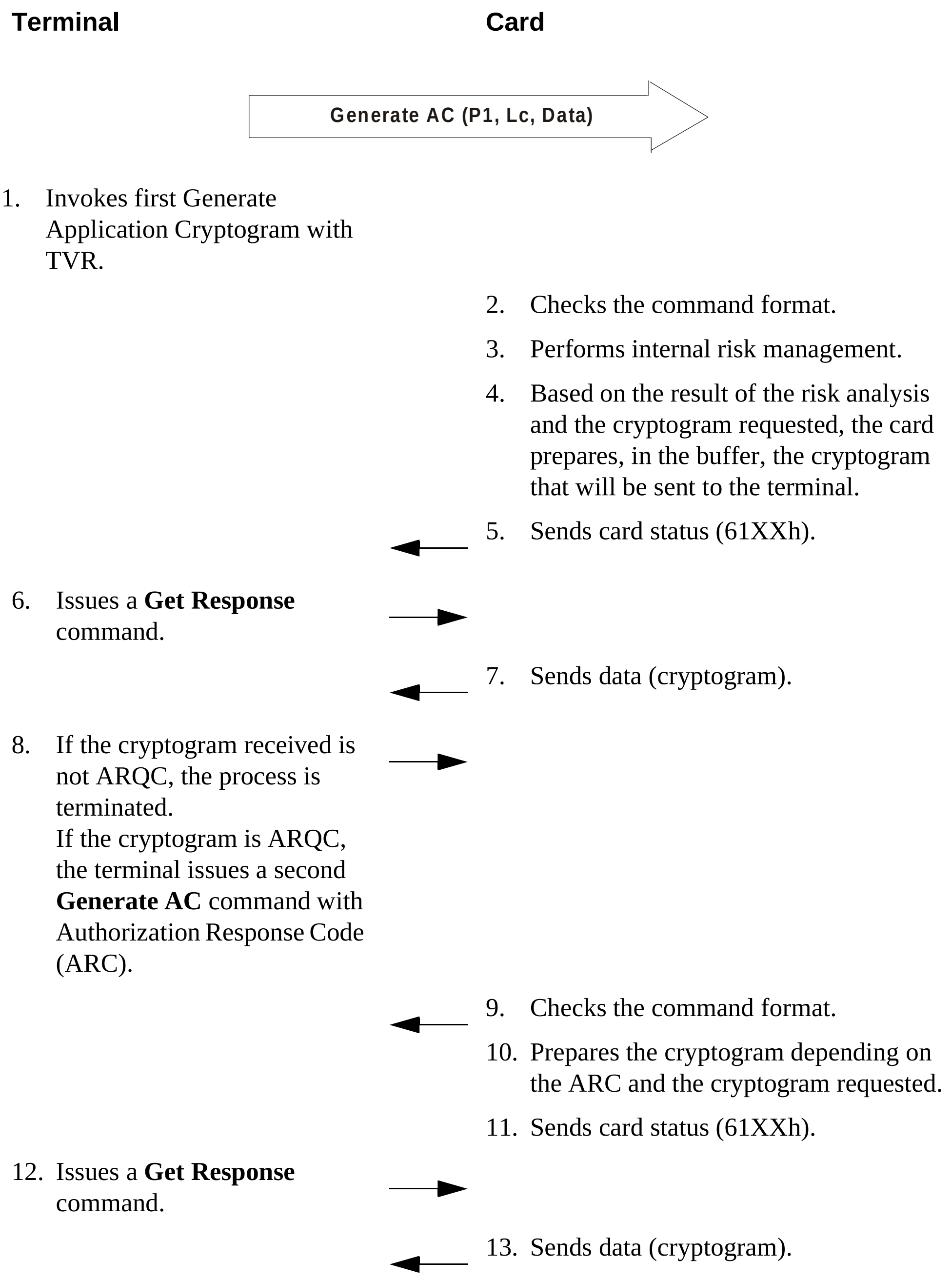
Byte	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**	**Cryptogram**
1	0	0	0	0	0	0	1	1	Length indicator ('03')
2	 0 0 1	 0 1 0							**Cryptogram returned in second GAC (C2G)** AAC TC NCR (No Cryptogram Requested)
2			 0 0 1	 0 1 0					**Cryptogram returned in first GAC (C2G)** AAC TC ARQC
2								1	UGO (Unable to Go Online)
3	1								LTI (Last online Transaction Incomplete)
4									Not supported

Values not indicated in this table are not supported in this specification.

SW1 and SW2 Hold the status byte values returned by the card. *“Appendix B - Card Contact Interface Return Codes”* lists the possible contents of SW1 and SW2.

Command Processing

“Figure 39 - Command Processing for Generate Application Cryptogram” shows a detailed procedure of the command processing.



Terminal

1. Invokes first Generate Application Cryptogram with TVR.

Card

2. Checks the command format.
3. Performs internal risk management.
4. Based on the result of the risk analysis and the cryptogram requested, the card prepares, in the buffer, the cryptogram that will be sent to the terminal.
5. Sends card status (61XXh).

Terminal

6. Issues a **Get Response** command.

Card

7. Sends data (cryptogram).

Terminal

8. If the cryptogram received is not ARQC, the process is terminated.
   If the cryptogram is ARQC, the terminal issues a second **Generate AC** command with Authorization Response Code (ARC).

Card

9. Checks the command format.
10. Prepares the cryptogram depending on the ARC and the cryptogram requested.
11. Sends card status (61XXh).

Terminal

12. Issues a **Get Response** command.

Card

13. Sends data (cryptogram).

Figure 39 - Command Processing for Generate Application Cryptogram

Get Card Information (GetInfo)

Use this command to retrieve card-related information. These data can be retrieved at any moment in a session, as follows:

- Data generated by the card in response to a command
- Generic system data
- Application data

The information can be retrieved at any moment in a session.

Note: If P2 = 05h, the EF must be selected first using the **Select File** command.

Format

CLA	INS	P1	P2	Le
80h	C0h	P1	P2	Le

Where:

P1, P2	Specifies the data to be retrieved.
Le	Specifies the response length.

Response

Response	SW1	SW2

Where:

Response	According to the values specified in P1 and P2. *“Table 19 - Get Card Information Command Response”* shows all the possible data.

P1	P2	Data Retrieved	Le (bytes)
00h	00h	Data prepared in the RAM in response to the previous command. For more details on how the command operates in this mode, see *“Get Response (GetResp)”* on page 104.	Variable
02h	A0h	Chip Serial Number	8
	A1h	Card Serial Number	8
	A2h	Issuer Serial Number	8
	A3h	Issuer Reference Number	4
	A4h	The following pre-issuing data: Family Name (FMN) * Product Name (PRN) * OS Version (OSV) * Program Version (PRV) * Chip Identification Value (CIV) * Gemplus Issuer Reference SECU byte TPRG byte LOCK1 byte LOCK2 byte	13 1 1 1 1 1 4 1 1 1 1
	05h	The application data of each EF_{sc} from the currently selected DF: Mod MPN | SCR | UCR | CKS	No. secret code x4
		The application data of each key from the currently selected EF: Key Type | ONC | Key | CKS	No. of key x4
		The application data of each EF_{purse} from the currently selected DF: Maximum Balance | Credit Key File Reference Maximum Free Debit Value | Access (Cd) | Access (Cb)	8
		The application data of each enhanced EF_{purse} from the currently selected DF, is the EF_{purse} information + enhanced EF_{purse} information. Maximum Balance | Credit Key File Reference Maximum Free Debit Value | Access (Cd) | Access (Cb) 00h | 00h | 00h | 000L | CAC	8 + 4
	06h	Descriptor address of the last selected EF or DF.	2

* See *“Table 20 - Product Identification Values”*
All other values of P1 and P2 are reserved for future use.

Table 19 - Get Card Information Command Response

Note: If a DF has just been selected and no EF has been selected, the last selected file is the DF. If an EF is then selected, the last selected file will be EF itself.

If application data is retrieved, the Le field value for the secret codes and keys must be consistent with the number of keys and secret codes in the EF_{sc} and EF_{key} from the currently selected DF. For example, if the EF_{key} contains three keys, the data length for the EF_{key} must be 0Ch (that is, 3 x 4 = 12).

The product identification values are listed in *"Table 20 - Product Identification Values"*:

Pre-Issuing Data	Descriptions
FMN (Family Name)	02h (Gemplus generic product)
PRN (Product Name)	01h (MPCOS-EMV range)
OSV (OS Version)	Reserved
Program Version (PRV)	01h (for 5 V range only)
Chip Identification Value (CIV)	2 KB 63h 8 KB 5Ch 32 KB 69h

Table 20 - Product Identification Values

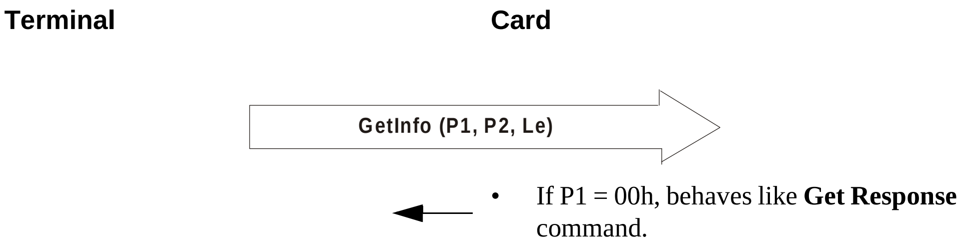
The FMN and PRN can be used to identify a card as MPCOS-EMV regardless of the EEPROM size, the operating system version or the chip type.

Note: It is not recommended to use the Chip Identification Value (CIV) to identify a card, as this may change if alternative chip suppliers are used.

SW1 and SW2	Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

"Figure 40 - Command Processing for Get Card Information" shows a detailed procedure of the command processing.



- If P1 = 00h, behaves like **Get Response** command.
- If P2 = 02h, retrieves the following data:
  - Chip serial number
  - Card serial number
  - Issuer serial number
  - Issuer reference number
  - Pre-issuing data
  - Secret codes, keys and purses information
- If P2 = 06h, returns descriptor address of the last selected EF or DF.

Figure 40 - Command Processing for Get Card Information

Get Challenge (GetChal)

Use this command to generate an eight-byte random number, prior to the **External Authenticate** command.

The random numbers are sent to the terminal in order to be ciphered by the terminal and it is kept in the RAM for verification later.

The eight-byte random number will be lost under any of the following circumstances:

- Card reset
- Unsuccessful **Get Challenge** command
- Unsuccessful **External Authenticate** command

Format

CLA	INS	P1	P2	Le
00h	84h	00h	00h	Le

Where:

Le	08h	Specifies the response length.

Response

Response	SW1	SW2

Where:

Response	An eight-byte random number.
SW1 and SW2	Hold the status byte values returned by the card. *“Appendix B - Card Contact Interface Return Codes”* lists the possible contents of SW1 and SW2.

Command Processing

“Figure 41 - Command Processing for Get Challenge” shows a detailed procedure of the command processing.

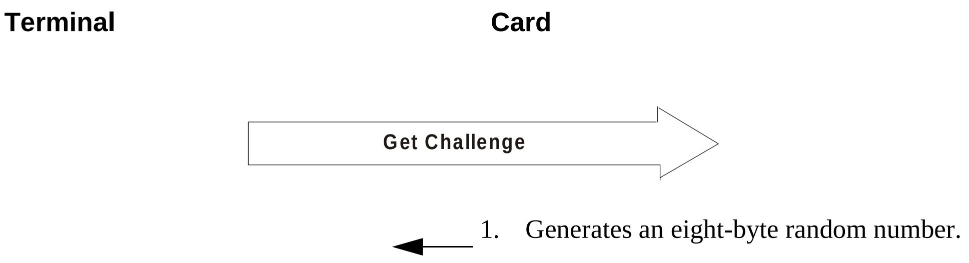


Figure 41 - Command Processing for Get Challenge

Get Data

Use this command to retrieve the Card Production Life Cycle (CPLC) data or the Application Transaction Counter (ATC).

The **Get Data** command for CPLC data can be invoked in any number of times and anywhere in the file hierarchy.

The **Get Data** command for ATC data can only be invoked under ADFs that are properly initialized.

Format

CLA	INS	P1	P2	Lc	Le
80h	CAh	9Fh	P2	00h	Le

Where:

P2= 7Fh, Le = 2Dh	To retrieve CPLC data.
P2= 36h, Le = 05h	To retrieve ATC data.

Response

Response	SW1	SW2

Where:

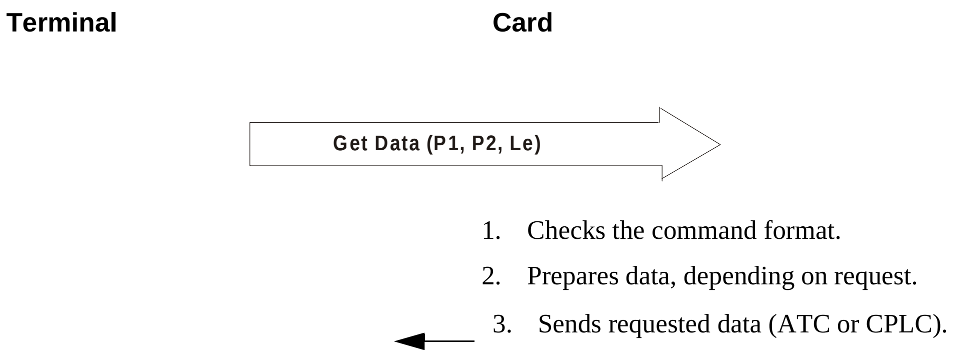
Response The data field of the response is in TLV format, as follows:

Data Field			**Le**
Tag	**Length**	**Value**	
9F7Fh	42 bytes (2Ah)	IC Fabricator (2) IC Type (2) Operating System Identifier (2) Operating System Release Date (2) Operating System Release Level (2) IC Fabrication Date (2) IC Serial Number (4) IC Batch Identifier (2) IC Module Fabricator (2) IC Module Packaging Date (2) ICC Manufacturer (2) IC Embedding Date (2) IC Pre-Personalizer (2) IC Pre-Personalization Date (2) IC Pre-Personalization Equipment Identifier (4) IC Personalizer (2) IC Personalization Date (2) IC Personalization Equipment Identifier (4)	2Dh
9F36h	1 byte (02h)	Application Transaction Counter (2)	05h

SW1 and SW2	Hold the status byte values returned by the card. *“Appendix B - Card Contact Interface Return Codes”* lists the possible contents of SW1 and SW2.

Command Processing

“Figure 42 - Command Processing for Get Data” shows a detailed procedure of the command processing.



1. Checks the command format.
2. Prepares data, depending on request.
3. Sends requested data (ATC or CPLC).

Figure 42 - Command Processing for Get Data

Get Processing Options

This command is used by the terminal to signal the card that transaction processing is beginning.

Note: The **Get Processing Options** command can only be invoked under a properly initialized Application Definition File (ADF).

Current transaction, if any, will be terminated if any of the following error occurs:

- Condition of use not satisfied error (6985h)
- IADF descriptor or ATC checksum error (6400h)
- ATC has reached maximum (6581h)

Format

CLA	INS	P1	P2	Lc	Data	Le
80h	A8h	00h	00h	02h	Data	00h

Where:

Data The data takes the form of the following structure:

83h	00h

Response

Response	SW1	SW2

Where:

Response The response message is coded in Format 1, starting with tag 80h

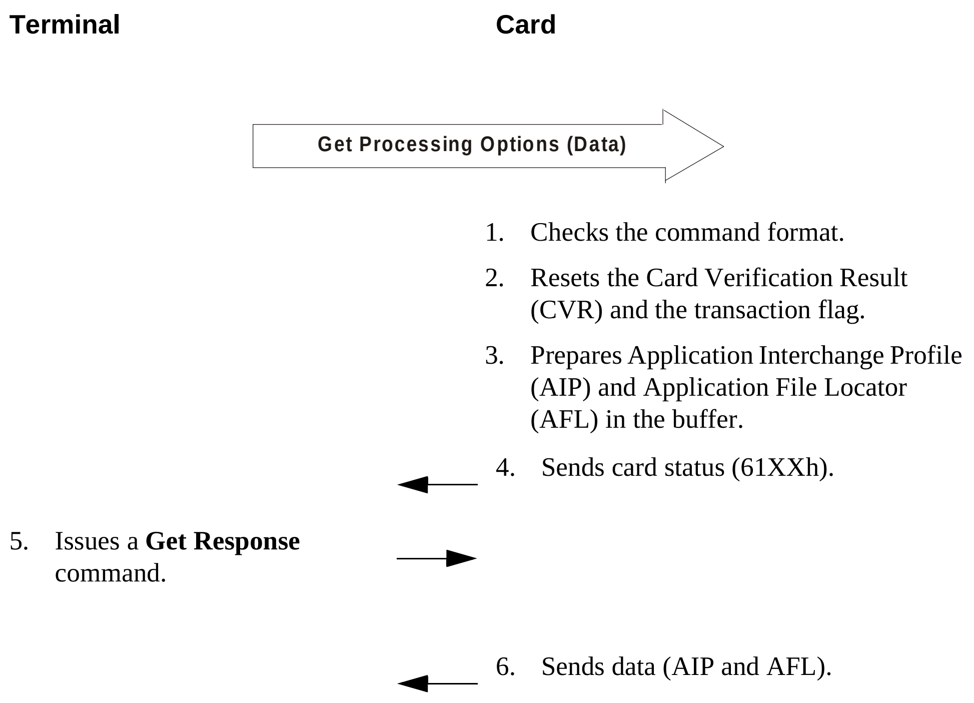
80h	LNG (var)	AIP (2)	AFL (var.)

LNG	Length of the AIP abd AFL (2 + AFL_{lng}) (in bytes)
AIP	Application Interchange Profile. 0800h (contained in the IADF EF)
AFL	Application File Locator. Depends on the location of application files and records as initialized in the IADF EF.

SW1 and SW2 Hold the status byte values returned by the card. *“Appendix B - Card Contact Interface Return Codes”* lists the possible contents of SW1 and SW2.

Command Processing

"Figure 43 - Command Processing for Get Processing Options" shows a detailed procedure of the command processing.



1. Checks the command format.
2. Resets the Card Verification Result (CVR) and the transaction flag.
3. Prepares Application Interchange Profile (AIP) and Application File Locator (AFL) in the buffer.
4. Sends card status (61XXh).
5. Issues a **Get Response** command.
6. Sends data (AIP and AFL).

Figure 43 - Command Processing for Get Processing Options

Get Response (GetResp)

Use this command to retrieve and erase the data prepared in the RAM by the card in response to the previous command. It can also be issued immediately after an ATR (Answer To Reset).

The terminal can send the **Get Response** command whenever the previous command returns a status byte value of '61 nn'h or '9F nn'h, where nn specifies the response length. When the terminal executes the **Get Response** command, it enters the value nn in its Le parameter. This process enables you to incorporate a transparent response return mechanism into MPCOS-EMV applications, which can then use the APDU approach.

Note: When the **Get Response** command is executed, the card erases the response from the RAM, therefore the same response cannot be retrieved more than once.

Format

CLA	INS	P1	P2	Le
00h	C0h	00h	00h	Le

Where:

Le	Specifies the response length.

Response

Response	SW1	SW2

Where:

Response	Holds the card response. *“Table 21 - MPCOS-EMV Command Response”* lists all the commands that generate the responses and describes the response data.

Command	Le	Response
Select Purse & Key	08h	Four rightmost significant bytes of Cryptogram (CR_{3-0}) + 1 unused byte + three rightmost significant bytes of Card Transaction Counter (CTC_{2-0}) CR_{3-0} | 00 | CTC_{2-0}
Select File Key	0Ch	Four rightmost significant bytes of Cryptogram (CR_{3-0}) + four leftmost significant bytes of Terminal Random Number ($TRnd_{7-4}$) + four leftmost significant bytes of Card Random Number ($CRnd_{7-4}$) CR_{3-0} | $TRnd_{7-4}$ | $CRnd_{7-4}$

Debit, Cancel Debit	02h	Two leftmost significant bytes of Card Debit Certificate (CDC$_{7–6}$) CDC7 CDC6
Read Balance	06h	1 unused byte + three-byte Balance Value (BV$_{2–0}$) + the 4th and 5th bytes of the Card Balance Certificate (CBC$_{5–4}$) 00h BV$_{2–0}$ CBC$_{5–4}$
Update Binary, Write Binary, Freeze Access Conditions, Create File, Update Record, Append Record	03h	Three leftmost significant bytes of Cryptographic Checksum (CRYCKS$_{7-5}$) if secure messaging is in use. CRYCKS 7 CRYCKS 6 CRYCKS 5
Select File	nnh	See *“Table 22 - Data Returned by Select File”*.
Internal Authenticate	4	Four leftmost significant bytes of the card Internal Authentication Cryptogram (IAC$_{3-0}$) IAC3 IAC2 IAC1 IAC0
Generate AC	14h	See *“Generate Application Cryptogram (Generate AC)”* on page 91 for the response data.
Get Processing Options	4 + AFL$_{length}$	See *“Get Processing Options”* on page 102 for the response data.

Table 21 - MPCOS-EMV Command Response

Response for **Select File** command:

The data generated by the card when it executes **Select File** depends on the presence (or the absence) of the IADF and the type of file selected.

If the IADF is present in the current application, the response will depend on the coding and the content of the IADF.

If the IADF is absent in the current application, the data is returned in the TLV format and *"Table 22 - Data Returned by Select File"* shows the data generated for both file types:

Tag	Length	Value	Returned After Selecting
85h	10h	File descriptor	Dedicated or elementary file
84h	01h to 10h	Dedicated file name	Dedicated file

Table 22 - Data Returned by Select File

Answer To Reset: After the ATR, the card will return the descriptor of the implicitly selected MF or DF if you issue a **Get Response** command. The DF name is not accessible. This data is returned in the TLV format as follows:

Tag	Length	Value	Returned After Selecting
85h	10h	File descriptor	Master or dedicated file

SW1 and SW2	Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

"Figure 44 - Command Processing for Get Response" shows a detailed procedure of the command processing.

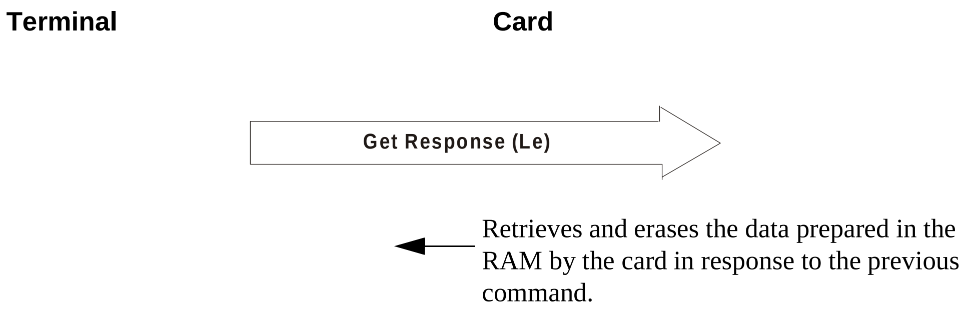

Terminal
Card
Get Response (Le)
Retrieves and erases the data prepared in the RAM by the card in response to the previous command.

Figure 44 - Command Processing for Get Response

Internal Authenticate (IntAut)

Use this command to authenticate the card with respect to the terminal.

The terminal asks the card to calculate an Internal Authentication Cryptogram (IAC) that is computed using the random number sent by the terminal and an authentication key held in the card. It then issues a **Get Response** command to retrieve the four rightmost significant bytes of the eight-byte IAC (that is, IAC_{0-3}) from the card.

The terminal calculates its own cryptogram and compares it with the IAC_{0-3}.

Note: Upon each execution of this command, the value of the Off-Nominal Counter (ONC) attached to the referenced key is decremented. See *"Off-Nominal Counter (ONC) Mechanism"* on page 45 for details.

Format

CLA	INS	P1	P2	Lc	Data	Le
00h	88h	00h	P2	0Ah	Data	04h

Where:

P2 Specifies the level of the relevant file.

Bit	7	6	5	4	3	2	1	0
	L	X	X	X	X	X	X	X

L Specifies the level of the relevant file that is holding the authentication key.
0: Global level
1: Local level

Data String of data contains the reference of the relevant file that stored the authentication key and the four leftmost significant bytes of the eight-byte EAC.

Kn	Kf	TRnd7	TRnd6	TRnd5	TRnd4	TRnd3	TRnd2	TRnd1	TRnd0

Kn Specifies the authentication key number and it is coded as:

Bit	7	6	5	4	3	2	1	0
	Key Number							0

Kf Specifies the Short File Identifier (SFI) of the relevant file that is holding the authentication key and it is coded as:

Bit	7	6	5	4	3	2	1	0
	0	0	0	SFI				

$TRnd_{7-0}$ Terminal Random Number.

Response

Response	SW1	SW2

Where:

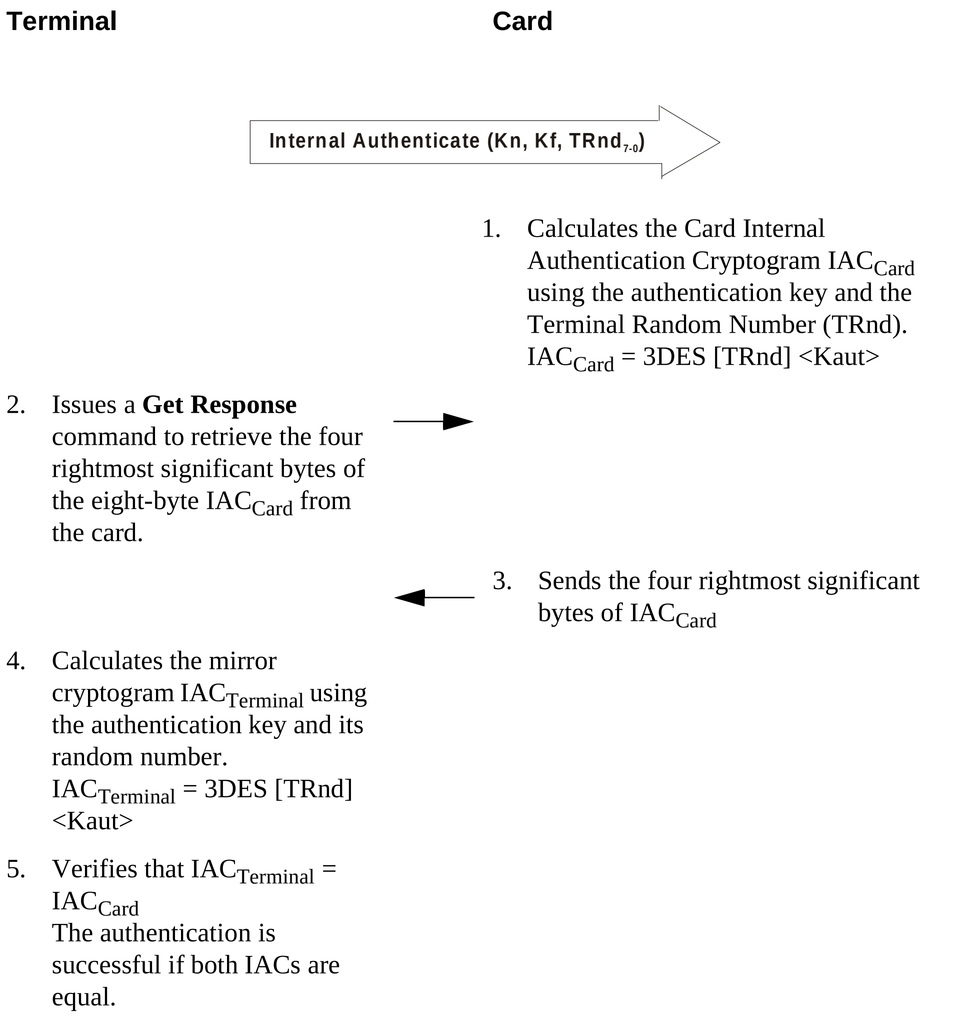
Response	The four rightmost significant bytes of the eight-byte Internal Authentication Cryptogram (IAC).

IAC3	IAC2	IAC1	IAC0

SW1 and SW2	Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

"Figure 45 - Command Processing for Internal Authenticate" shows a detailed procedure of the command processing.



1. Calculates the Card Internal Authentication Cryptogram IAC$_{Card}$ using the authentication key and the Terminal Random Number (TRnd).
   IAC$_{Card}$ = 3DES [TRnd] <Kaut>
2. Issues a **Get Response** command to retrieve the four rightmost significant bytes of the eight-byte IAC$_{Card}$ from the card.
3. Sends the four rightmost significant bytes of IAC$_{Card}$
4. Calculates the mirror cryptogram IAC$_{Terminal}$ using the authentication key and its random number.
   IAC$_{Terminal}$ = 3DES [TRnd] <Kaut>
5. Verifies that IAC$_{Terminal}$ = IAC$_{Card}$
   The authentication is successful if both IACs are equal.

Figure 45 - Command Processing for Internal Authenticate

Read Balance (RdBal)

Use this command to read the balance value of a specified file.

It can be executed at any time as long as the read balance access condition has been fulfilled for the purse whose balance is to be read (see *"Purse Files"* on page 12).

Read Balance not only returns the balance value, but the Card Balance Certificate (CBC).

Note: If this command fails, the current purse pointer, purse references and RAM indicators will be lost. The temporary payment transaction key will also be lost and the purse must be re-selected using the **Select Purse & Key** command.

If the temporary payment transaction key has not been established, (that is, using the **Select Purse & Key** command), the **Read Balance** certificate will have no effect.

Purse access for **Read Balance** is based on the Cb access condition, regardless of the purse file access condition for read.

The MRN mechanism will be activated if MRN value is non-zero. See *"Maximum Replay Number (MRN) Mechanism"* on page 46 for details.

Format

CLA	INS	P1	P2	Lc	Data	Le
80h	32h	00h	Pf	04h	Data	06h

Where:

P2 Pf Short File Identifier (SFI) of the purse file, whose balance is to be read. The identifier has the following format:

Bit	7	6	5	4	3	2	1	0
	0	0	0	Purse File				

Data Contains the four-byte Terminal Transaction Counter (TTC) value. This can be used to keep track of the commands sent by each terminal. If this feature is not required, the value 0 is entered.

Response

Response	SW1	SW2

Where:

Response Holds the current balance value and bytes 4 and 5 of the Card Balance Certificate (CBC).

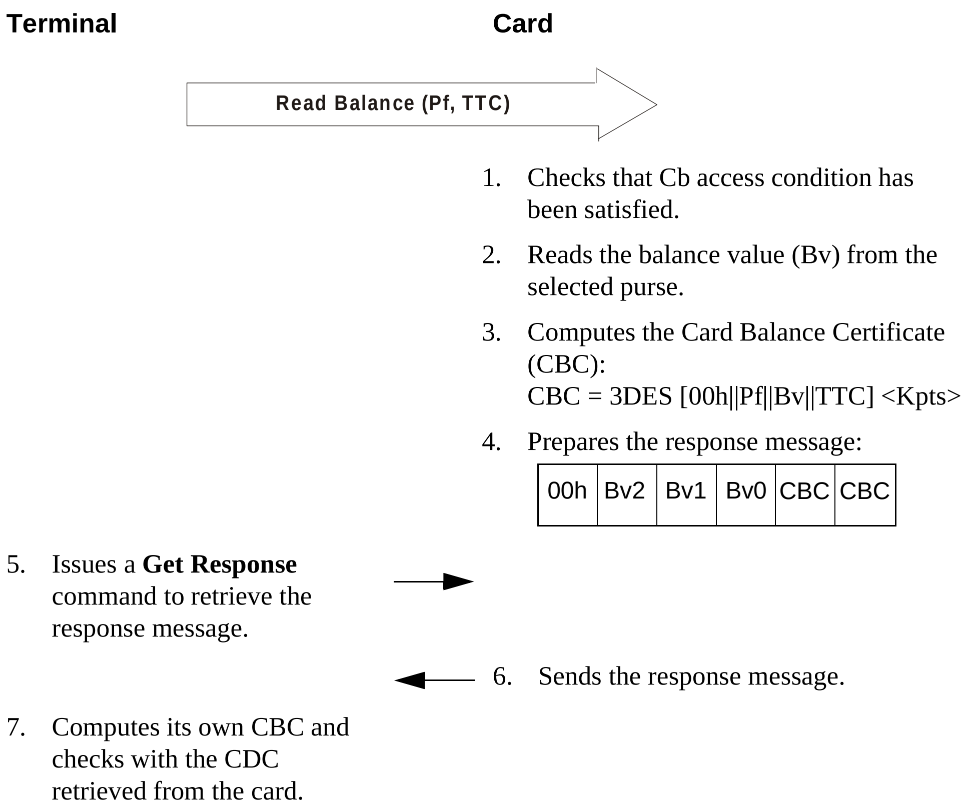
CBC = 3DES[	00h	Pf	Bv2	Bv1	Bv0	TTC2	TTC1	TTC0	] <Kpts>

Pf	The SFI of the purse file whose balance value is to be read.
BV_{2-0}	Balance value
TTC_{2-0}	Terminal transaction counter
Kpts	Temporary payment transaction key

SW1 and SW2	Hold the status byte values returned by the card. *“Appendix B - Card Contact Interface Return Codes”* lists the possible contents of SW1 and SW2.

Command Processing

*“Figure 46 - Command Processing for Read Balance”*shows a detailed procedure of the command processing.



1. Checks that Cb access condition has been satisfied.
2. Reads the balance value (Bv) from the selected purse.
3. Computes the Card Balance Certificate (CBC):
   CBC = 3DES [00h||Pf||Bv||TTC] <Kpts>
4. Prepares the response message:

00h	Bv2	Bv1	Bv0	CBC	CBC

5. Issues a **Get Response** command to retrieve the response message.
6. Sends the response message.
7. Computes its own CBC and checks with the CDC retrieved from the card.

Figure 46 - Command Processing for Read Balance

Read Binary (RdBin)

Use this command to read data from the transparent Elementary File (EF). It can only be executed if the Group 3 access conditions are fulfilled.

Read Binary can either read from the currently selected EF (direct selection) or select an EF by specifying its Short File Identifier (SFI) and then read from it (implicit selection). In an implicit selection, the newly selected EF will become the current EF. In case an error is encountered during the command, the newly selected EF will be the current EF.

Secure Messaging

This command can use secure messaging, as long as the record length and the certificate length do not exceed the internal buffer size (see *“IO Buffer Size”* on page 53).

See *“Secure Messaging”* on page 41 for further information on the conditions which must be met before secure messaging can be used.

In secure messaging mode, the card returns the data in clear text plus an additional three-byte cryptographic checksum. Additionally, the Maximum Replay Counter (MRC) decrements by one. See *“Maximum Replay Number (MRN) Mechanism”* on page 46 for details.

The checksum is calculated using the data field plus the command header.

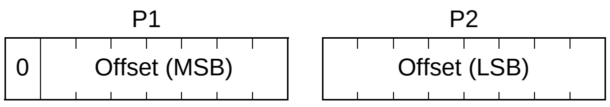
Note: Even when the access condition for a **Read Binary** command specifies a key number, it is still possible to read the file without secure messaging (CLA = 00h).

Format

CLA	INS	P1	P2	Le
CLA	B0h	P1	P2	Le

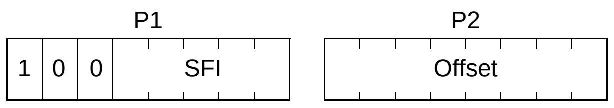
Where:

CLA	Specifies transmission mode (normal or secure messaging):
	00h Transmit normally.
	04h Transmit with secure messaging.
P1, P2	Specifies the file to be read.
	In a direct selection, the data is read from the currently selected EF by coding P1 and P2 as follows:



The seven lsb of P1 and the P2 byte specify the offset in data unit, from the beginning of the file to the first byte of the data.

In an implicit selection, the data is read from the selected EF by coding P1 and P2 as follows:



The five lsb of P1 specify the SFI of the file. The P2 byte specifies the offset in data unit, from the beginning of the file to the first byte of the data.

Le	Specifies the length of the data (in bytes) to be read. If secure messaging is used, a value of 03h bytes should be added.

Note: If Le is set to zero or if it exceeds the amount of available data, the card will return the status code “6C nn”, where “nn” is the length of the available data.

Response

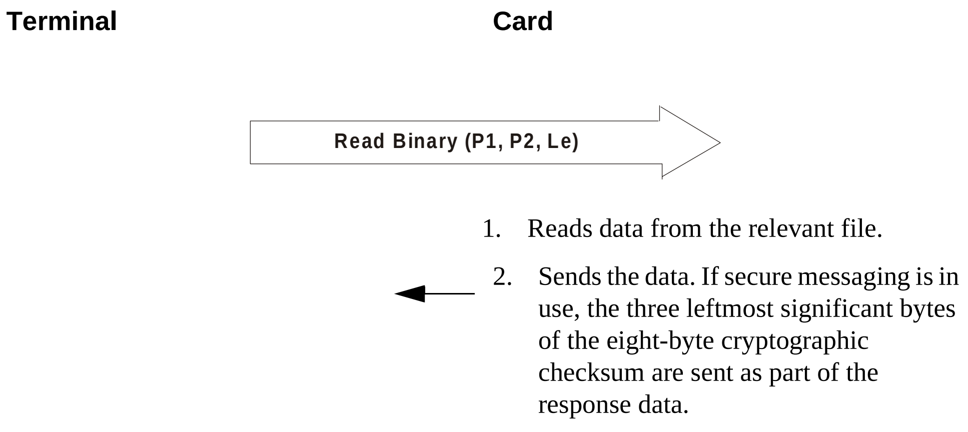
Response	SW1	SW2

Where:

Response	Holds the data returned by the card. If secure messaging is in use, the three leftmost significant bytes of the eight-byte cryptographic checksum are included.
SW1 and SW2	Hold the status byte values returned by the card. *“Appendix B - Card Contact Interface Return Codes”* lists the possible contents of SW1 and SW2.

Command Processing

“Figure 47 - Command Processing for Read Binary” shows a detailed procedure of the command processing.



1. Reads data from the relevant file.
2. Sends the data. If secure messaging is in use, the three leftmost significant bytes of the eight-byte cryptographic checksum are sent as part of the response data.

Figure 47 - Command Processing for Read Binary

Read Memory

Use this command to retrieve the Card Serial Number (CSN). The CSN can also be retrieved using the **Get Card Information** command.

Format

CLA	INS	P1	P2	Le	Le
80h	B8h	00h	00h	00h	08h

Response

The card returns the CSN.

Command Processing

"Figure 48 - Command Processing for Read Memory" shows a detailed procedure of the command processing.

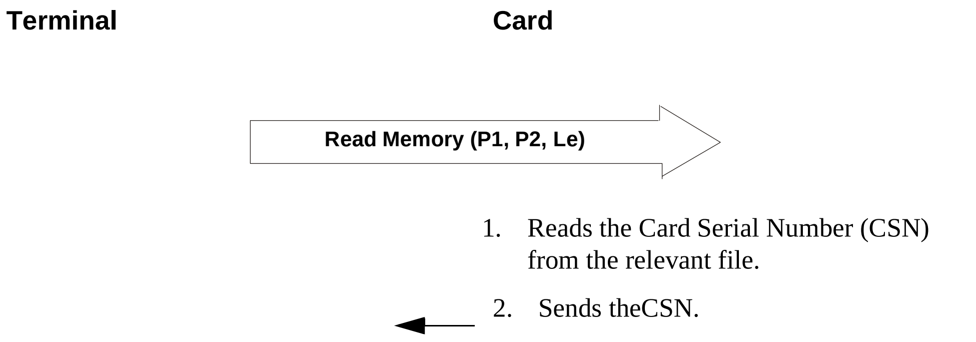

Terminal
Card
Read Memory (P1, P2, Le)
1. Reads the Card Serial Number (CSN) from the relevant file.
2. Sends theCSN.

Figure 48 - Command Processing for Read Memory

Read Record

Use this command to read data from a structured file.

Files can be selected directly or implicitly. When selected directly, the file must have been previously selected using the **Select File** command. In an implicit selection, the file is selected in the current DF, using the Short File Identifier (SFI). If the selection is successful, the selected file becomes the current file and the record pointer will be cleared.

Note: When reading data from cyclic files, the current record always returns the contents of last appended or updated record.

Note: If the access condition for a **Read Record** command specifies a key number, it is still possible to read the file without secure messaging (CLA = 00h).

Format

CLA	INS	P1	P2	Le
CLA	B2h	P1	P2	Le

This command can use secure messaging, as long as the record length and the certificate length do not exceed the internal buffer size (see *"IO Buffer Size"* on page 53).

See *"Secure Messaging"* on page 41 for further information on the conditions which must be met before secure messaging can be used.

In secure messaging mode, the card returns the data in clear text plus an additional three-byte cryptographic checksum. Additionally, the Maximum Replay Counter (MRC) decrements by one. See *"Maximum Replay Number (MRN) Mechanism"* on page 46 for details.

The checksum is calculated using the data field plus the command header.

Where:

CLA — Specifies transmission mode (normal or secure messaging):

00h	Transmit normally.
04h	Transmit with secure messaging.

P1 — Specifies the record number of the relevant file to be read and it is coded as:

Byte	7	6	5	4	3	2	1	0
	Record Number							

P2 — Specifies the Short File Identifier (SFI) of the relevant file in which the record is to be read. It is coded as:

Byte	7	6	5	4	3	2	1	0
	SFI					1	0	0

	In a direct selection, where the relevant file was previously selected by using the **Select File** command, the SFI is set to zero.
Le	Specifies the length of the data (in bytes) to be read. If secure messaging is used, a value of 03h bytes should be added.

Note: If Le is set to zero or if it exceeds the amount of available data, the card will return the status code “6C nn”, where “nn” is the length of the available data.

Response

Response	SW1	SW2

Where:

Response	Holds the data returned by the card. If secure messaging is in use, the three leftmost significant bytes of the eight-byte cryptographic checksum are included.
SW1 and SW2	Hold the status byte values returned by the card. *“Appendix B - Card Contact Interface Return Codes”* lists the possible contents of SW1 and SW2.

Command Processing

“Figure 49 - Command Processing for Read Record” shows a detailed procedure of the command processing.

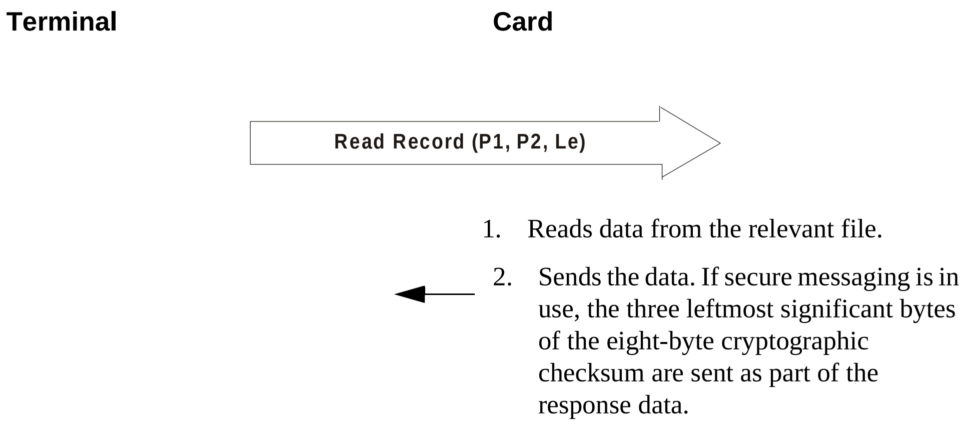

Terminal
Card
Read Record (P1, P2, Le)
1. Reads data from the relevant file.
2. Sends the data. If secure messaging is in use, the three leftmost significant bytes of the eight-byte cryptographic checksum are sent as part of the response data.

Figure 49 - Command Processing for Read Record

Select File (SelFil)

Use this command to select either a Dedicated File (DF) or an Elementary File (EF).

When developing an application you should take into account the effect of file selection on the authorization register:

- At the beginning of a session (after a reset), all authorization registers are set to 00h.
- When a file is selected, the card does not clear the MF authorization register.
- When a new DF is selected (even if it is the same file as the previously selected DF), the operating system will clear the local authorization register and the temporary key status.

Format

CLA	INS	P1	P2	Lc	Data	Le
00h	A4h	P1	P2	Lc	Data	Le

Where:

P1	Specifies the selection mode:	
	00h	Select either the MF or an EF using file identifier.
	01h	Select a child DF when the MF is currently selected. The selection is made using the file identifier of the DF coded on two bytes.
	02h	Select an EF under the current DF. The selection is made using the file identifier of the targeted EF coded on two bytes.
	03h	Select the MF automatically without providing the file identifier.
	04h	Select the DF by using a part or the entire of the file name.
P2	00h	First or only occurrence with response.
	02h	Next occurence with response (applicable only if P1 = 04h).
	0Ch	First or only occurrence without response.
	0Eh	Next occurrence without response (applicable only if P1 = 04h).

Note: To select the next file occurrence (that is, P2 = 02h or P2 = 0Eh), a file must be previously selected by name.

Lc Specifies the data-field length. The length is dependent on the value of P1.

To select the next file occurrence (that is, P2 = 02h or P2 = 0Eh), the Lc value must be equal to the one in the previous selection. Otherwise, a context error is returned.

Data Specifies the file to be selected. This is dependent on the value of P1 and is coded as follow:

P1	**Lc (bytes)**	**Data**
00h	02h	3F00h or EF identifier
01h	02h	DF identifier
02h	02h	EF identifier
03h	00h	Nil
04h	01h–10h	DF name

Note: To select the next file occurrence (that is, P2 = 02h or P2 = 0Eh), the Data must be the same as the previous partial name used to select the current DF. Otherwise, a context error is returned.

Le Variable, when P2 = 00h

Response

Response	SW1	SW2

Where:

Response (without IADF)	In standard mode (that is, there is no IADF), this response (without IADF) contains the selected file descriptor and the dedicated file name (if any) if a dedicated file has been selected. The terminal can retrieve this data by executing the **Get Response** command. Refer to *"Figure 50 - Select File Response Format in Standard Mode"*.

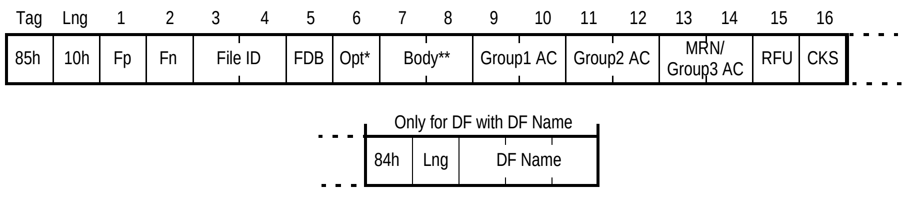


Figure 50 - Select File Response Format in Standard Mode

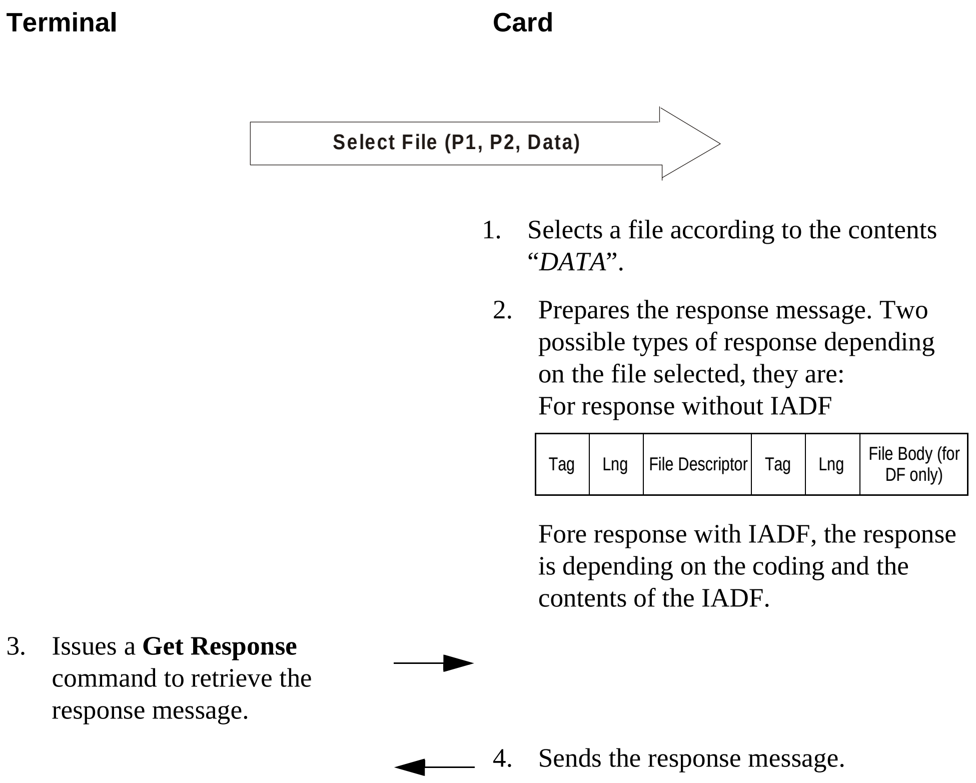
Opt	Optional byte for DF. * RecLen (Record Length) of the EF if an EF is selected.
Body	Length of the DF name. ** Body size of the EF if an EF is selected.
MRN	For DF only. Maximum Replay Number on one byte.
Group3 AC	Group 3 access condition (for EF only).

Note: The first 16 bytes format apply to both EF and DF without DF name. The extended portion applies only to DF with DF name.

Response (with IADF)	If the IADF is present in the current application, the response will depend on the coding and contents of the IADF (see *“Internal Application Data File (IADF)”* on page 20).
SW1 and SW2	Hold the status byte values returned by the card. *“Appendix B - Card Contact Interface Return Codes”* lists the possible contents of SW1 and SW2.

Command Processing

“Figure 51 - Command Processing for Select File” shows a detailed procedure of the command processing.



1. Selects a file according to the contents *“DATA”*.
2. Prepares the response message. Two possible types of response depending on the file selected, they are:
   For response without IADF

Tag	Lng	File Descriptor	Tag	Lng	File Body (for DF only)

   Fore response with IADF, the response is depending on the coding and the contents of the IADF.
3. Issues a **Get Response** command to retrieve the response message.
4. Sends the response message.

Figure 51 - Command Processing for Select File

Select File Key (SelFk)

Use this command to generate a session key (Temporary Administration Key, Kats) from an administration key, prior to issuing a non-payment related command with secure messaging. A cryptogram will be generated from Kats. The terminal retrieves the cryptogram using **Get Response** command, for authenticating the card and to compute its own session key.

Note: The Off-Nominal Counter (ONC) associated with the key decrements each time this command is executed. See *“Off-Nominal Counter (ONC) Mechanism”* on page 45 for details.

This command cannot be used for setting up a payment session key.

When issuing this command, the previous session key (if any) will be lost, whether the command has executed successfully or not. This is true even if the previous session key was set up with the **Select Purse & Key** command. That is, there can only be one session key at all times.

An external authentication may be required, depending on the setting in the option byte of the current dedicated file descriptor.

Format

CLA	INS	P1	P2	Lc	Data	Le
80h	28h	Kn	Kf	08h	Data	0Ch

Where:

P1 Kn Specifies the key number in the relevant key file that will be used to compute the Kats. This value is sent in the following format:

Bit	7	6	5	4	3	2	1	0
	Key Number							0

Key Number (7 bits) indicates the key position inside the relevant key file.

Note: The key number of a 3DES key must be an even number.

P2 Kf Specifies the relevant key file contains the administration key which will be used to compute the Kats. This value is sent in the following format:

Bit	7	6	5	4	3	2	1	0
	0	0	L	Key File				

L (1 bit)	Specifies the level of the relevant key file. 0: Global level 1: Local level
Key File (5 bits)	Specifies the Short File Identifier (SFI) of the relevant key file.

Note: To enable secure messaging, the key file specified in this field must be the same as that specified in the access condition of the relevant key file.

Data	The eight-byte Terminal Random Number ($TRnd_{7–0}$)

Response

Response	SW1	SW2

Where:

Response Holds the data returned by the card and it is coded as follows:

$CR_{3–0}$	$TRnd_{7–4}$	$CRnd_{7–4}$

$CR_{3–0}$	The four rightmost significant bytes of the eight-byte cryptogram.
$TRnd_{7–4}$	The four leftmost significant bytes of the eight-byte terminal random number.
$CRnd_{7–4}$	The four leftmost significant bytes of the eight-byte card terminal random number.

SW1 and SW2 Hold the status byte values returned by the card. *“Appendix B - Card Contact Interface Return Codes”* lists the possible contents of SW1 and SW2.

Command Processing

“Figure 52 - Command Processing for Select File Key” shows a detailed procedure of the command processing.

Terminal **Card**

Select File Key (Kn, Kf, TRnd)

1. Generates the eight-byte Terminal Random Number($TRnd_{7–0}$) $\longrightarrow$
2. Generates the eight-byte Card Random Number ($CRnd_{7–0}$).

Terminal

Card

3. Computes the session key (Kats).
   Kats = 3DES_16 [RN] <$Key_{Administration}$>
   RN: is the concatenation of the four leftmost significant byte of TRnd and CRnd

$TRnd_{7–4}$	$CRnd_{7–4}$

4. Activates the flag after the Kats has established.
5. It then computes the Authentication Cryptogram (CR).
   CR = 3DES [TRnd] <Kats>
6. Prepares the response message as:

$CR_{3–0}$	$TRnd_{7–4}$	$CRnd_{7–4}$

7. Issues a **Get Response** command to retrieve the response message. $\longrightarrow$

$\longleftarrow$ 8. Sends the response message.

9. Computes the session key. Kats.
   Kats = 3DES_16 [RN] <$Key_{Administration}$>
10. It then computes the Authentication Cryptogram (CR).
    CR = 3DES [TRnd] <Kats>
11. Checks $CR_{3–0}$ with those received from the card.

Figure 52 - Command Processing for Select File Key

Select Purse & Key (SelPK)

Use this command to perform the following functions:

- Select the specified purse and key.
- Increment the Card Transaction Counter (CTC).
- Generate a session key. Temporary Payment Transaction Key (Kpts) using a payment key.
- Compute a cryptogram using Kpts. The terminal retrieves the cryptogram using **Get Response** command, for authenticating the card and to compute its own session key.

It is typically used to establish a payment transaction session, enabling transactions to be performed.

Note: When the terminal establishes a payment transaction session, it is recommended that the CTC is incremented each time a payment command is performed.

If the CTC reaches its maximum value, a specific error code will be returned and the command becomes locked.

When issuing this command, the previous session key (if any) will be lost, whether the command has executed successfully or not. This is true even if the previous session key was set up with the **Select File Key** command. That is, there can only be one session key at all times.

An external authentication may be required, depending on the setting in the option byte of the current dedicated file descriptor.

Note: The Off-Nominal Counter (ONC) associated with the key decrements on each execution of this command. See *"Off-Nominal Counter (ONC) Mechanism"* on page 45 for details.

Format

CLA	INS	P1	P2	Lc	Data	Le
80h	30h	Kn	Kf	0Ah	Data	08h

Where:

P1	Kn	Specifies the key number in the relevant key file that will be used to compute the Kpts. This value is sent in the following format:

Bit	7	6	5	4	3	2	1	0
	Key Number							0

Key Number (7 bits) indicates the key position inside the relevant key file.

Note: The key number of a 3DES key must be an even number.

P2 Kf Specifies the relevant key file contains the administration key which will be used to compute the Kpts. This value is sent in the following format:

Bit	7	6	5	4	3	2	1	0
	0	0	0	Key File				

Key File (5 bits) Specifies the Short File Identifier (SFI) of the relevant key file.

Note: This key file is always addressed at the local level.

Data The eight-byte Terminal Random Number (TRnd). Holds the following data:

00h	Pf	$TRnd_{7–0}$

Pf Specifies the Short File Identifier (SFI) of the purse file to be selected, in the following format:

Bit	7	6	5	4	3	2	1	0
	0	0	0	Purse File				

$TRnd_{7–0}$ Holds the eight-byte random value generated by the terminal.

Response

Response	SW1	SW2

Where:

Response Holds the data returned by the card and it is coded as follows:

$CR_{3–0}$	00h	$CTC_{2–0}$

$CR_{3–0}$ The four rightmost significant bytes of the eight-byte cryptogram.

$CTC_{2–0}$ The three rightmost significant bytes of the eight-byte card transaction counter value.

SW1 and SW2 Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

"Figure 53 - Command Processing for Select Purse & Key" shows a detailed procedure of the command processing.

Terminal **Card**

Set Purse & Key (Kf, Kn, Pf, TRnd) →

1. Generates the eight-byte Terminal Random Number(TRnd$_{7–0}$) →

2. Increments Card Transaction Counter (CTC) value.

3. Computes the session key (Kpts).
   Kpts = 3DES_16 [00h||00h||00h||00h||00h||CTC$_{2–0}$] <K>
   CTC$_{2–0}$: The three rightmost significant bytes of CTC.
   K: Either a payment or a log key.

4. Checks and selects the purse specified in Pf.

5. Computes the Authentication Cryptogram (CR).
   CR = 3DES [TRnd$_{7–0}$] <Kpts>

6. Prepares the response message as:

CR$_{3–0}$	00h	CTC$_{2–0}$

7. Issues a **Get Response** command to retrieve the response message. →

← 8. Sends the response message.

9. Computes the session key. Kpts.
   Kpts = 3DES_16 [CTC] <Key>

10. It then computes the Authentication Cryptogram (CR).
    CR = 3DES [TRnd$_{7–0}$] <Kpts>

11. Checks CR$_{3–0}$ with those received from the card.

Figure 53 - Command Processing for Select Purse & Key

Set Card Status

Use this command to set the status of the Lock byte. See *"Lock Byte"* on page 53 for more information.

The card status options are as follows and apply to all applications in the card:

- Personalization complete (or not)
- Data unit

Note: It is very important that you set the personalization bit once personalization has been completed so that you can trace which processes have been completed by your card.

Format

CLA	INS	P1	P2	Lc	Data
80h	D8h	FFh	FFh	01h	Data

Where:

Data	Lock byte. Holds the lock byte to be updated. See *"Lock Byte"* on page 53 for the format of the Lock byte.

Response

SW1	SW2

SW1 and SW2	Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

"Figure 54 - Command Processing for Set Card Status" shows a detailed procedure of the command processing.

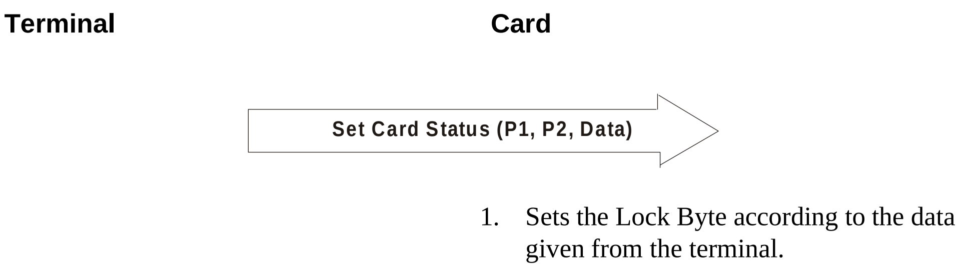

Terminal
Card
Set Card Status (P1, P2, Data)
1. Sets the Lock Byte according to the data given from the terminal.

Figure 54 - Command Processing for Set Card Status

Set Options (SetOpts)

Use this command to set the **Sign** command options for the currently selected Dedicated File (DF). The options are:

- Use current purse balance in the certificate calculation.
- Clear the following RAM parameters after executing the **Sign** command:
  - Last amount register
  - Purse pointer
  - Purse file reference
  - Authorization register
  - The indicators (key indicator, debit indicator, transaction OK, data valid)

Note: Once these options have been set, they remain unchanged until the terminal re-issues this command or starts a new session.

Format

CLA	INS	P1	P2
80h	3Ah	00h	P2

Where:

P1 The option byte, defining the options. This byte is sent in the following format:

Bit	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	RP	CB

RP To clear the RAM parameters after the **Sign** command is executed, that is, set bit 1 to 0. Otherwise, set bit 1 to 1.

CB Set bit 0 to 0, is to prevent the current purse balance from being used in the certificate calculation.

Set bit 0 to 1, is to include the current purse balance used in the certificate calculation.

Note: Modifying the CB bit will only be effective, if bit 2 in the parent DF “OPT” byte is set to 1. That is, allowing the inclusion of the current purse balance in certificate calculation. For more information on the DF descriptor, see *“File Descriptor”* on page 10.

Response

SW1	SW2

Where:

SW1 and SW2	Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

"Figure 55 - Command Processing for Set Options" shows a detailed procedure of the command processing.

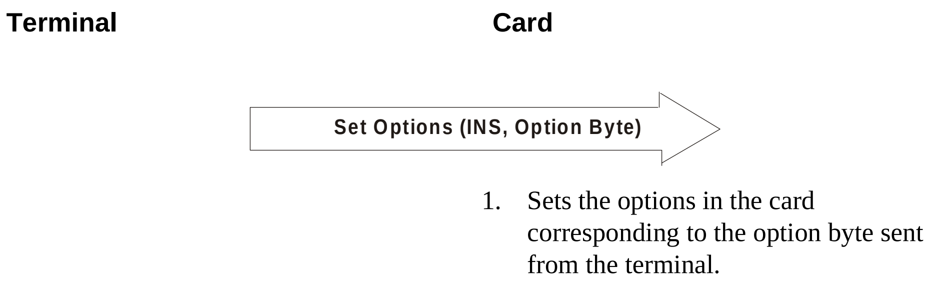

Terminal
Card
Set Options (INS, Option Byte)
1. Sets the options in the card corresponding to the option byte sent from the terminal.

Figure 55 - Command Processing for Set Options

Set Secret Code (SetCod)

Use this command to either unblock or to modify the value of an existing secret code with a new value. There is no limit to the number of the times that a secret code can be changed, as long as the access conditions are satisfied.

The command implicitly acts on the Secret Code EF (EF_{SC}) located in the current DF. This means that the local EF_{SC} containing the specified code (to be unblocked or changed) need not to be explicitly selected before issuing the command. If there is no EF_{SC} in the current DF, the command will be aborted.

Secure Messaging

When unlocking a secret code, the unlocked secret code value and the locked secret code replacement value sent in the command data field must be enciphered if the unlock secret code is flagged as enciphered (see *“Secret Code Files”* on page 18).

When replacing a secret code, the current secret code value and replacement secret code value in the command data field must be enciphered if the value of “*Mode*” in the secret code is 1.

Format

CLA	INS	P1	P2	Lc	Data	Le
CLA	24h	P1	P2	08h	Data	Le

Where:

CLA — Specifies transmission mode (normal or secure messaging):

80h	Transmit without secure messaging.
84h	Transmit with secure messaging.

P1 — Specifies the command execution mode.

00h	Change secret code
01h	Unblock secret code

P2 — Secret Code Number (SCN). Specifies the secret number (from 0 to 7) to be changed or unblocked in the secret code file for the currently selected DF.

Data — Holds the unlock or current secret code value and the replacement secret code value. It is coded as:

Bit	7	6	5	4	3	2	1	0
	SC Value				Replacement SC			

SC Value — If P1 = 00h, SC Value is the four-byte current secret code value.

If P1 = 01h, SC Value is the four-byte unlock secret code value.

Replacement SC Value	The four-byte replacement secret code value.

When changing a secret code or unlocking a secret code in a ciphered mode, the terminal enciphers the data field and sends it on eight-byte using the following process:

Enciphered Data = $3DES^{-1}$ [SC Value||Replacement SC Value]<Kats>

Kats is the Temporary Administration Key calculated by the card when the **Select File Key** or the **Select Purse & Key** command is executed.

Response

SW1	SW2

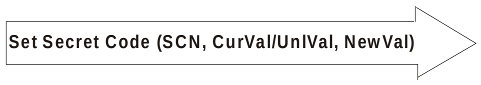
Where:

SW1 and SW2	Hold the status byte values returned by the card. *“Appendix B - Card Contact Interface Return Codes”* lists the possible contents of SW1 and SW2.

Command Processing

“Figure 56 - Command Processing for Set Secret Code” shows a detailed procedure of the command processing.

Terminal		Card
SCN: Secret Code Number in the EF_{SC} NewVal: Replacement Value		CurVal: Current Secret Code Value UnlVal: Unlock Secret Code Value
1. Sends the SCN and the enciphered eight-byte data, where the enciphered value is computed as: $3DES^{-1}$ [CurVal/UnlVal|| NewVal]<Kats>	$\longrightarrow$	2. Compares the CurVal or UnlVal with the current secret code value or the Unlock Code Reference (UCR) value stored in the secret code file respectively.



Terminal **Card**

3. If the values are identical, the following procedure will be carried out:

   Changing secret code values:

   – Sets Secret Code Ratification (SCR) counter to “0”.

   – If the authentication granted, replaces the original secret code value with the specified replacement secret code value.

Unlocking a secret code:

   – Sets the locked Secret Code Ratification (SCR) counter to “0”.

   – If the authentication granted, replaces the locked secret code value with the specified replacement secret code value.

4. If the values are identical, the following procedure will be carried out:

   Changing secret code values:

   – The SCR counter is incremented and it returns an appropriate error code. If the SCR counter reaches the Maximum Presentation Number (MPN) value, the secret code will be locked.

Unlocking a secret code:

   – The SCR counter in the unlock-code is incremented and it returns an appropriate error code.

Figure 56 - Command Processing for Set Secret Code

Sign

Use this command to generate a Card Sign Certificate (CSC) for the previous transaction, provided that a transaction has been successfully completed in the current payment transaction session. This signature may be used at a later stage to verify the validity of transaction logs in the terminals. The CSC is based on the purse, transaction value and optionally the purse balance after the transaction.

Note: All the RAM parameters can be cleared after executed the **Sign** command, depending on the option byte setting defines in the **Set Options** command.

The Off-Nominal Counter (ONC) associated with the key decrements upon each execution of the command. See *"Off-Nominal Counter (ONC) Mechanism"* on page 45 for details. The ONC does not decrement if there is no transaction to sign, since no cryptographic computation is done.

Format

CLA	INS	P1	P2	Le
80h	38h	P1	P2	04h

Where:

P1	Kn	Specifies the key number in the relevant key file to be used to compute the temporary key. This value is sent in the following format:

Bit	7	6	5	4	3	2	1	0
	Key Number							0

Note: The key number of a 3DES key must be an even number.

P2	Kf	Specifies the Short File Identifier (SFI) of the key file that holds the key to be used to compute the temporary key. This key file is always addressed at the local level (that is, within the current DF).

Bit	7	6	5	4	3	2	1	0
	X	X	X	Key File				

Response

Response	SW1	SW2

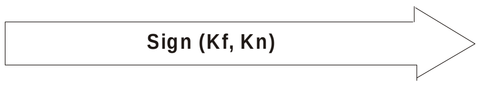
Where:

Response	Holds the four rightmost significant bytes of the Card Sign Certificate (CSC).
SW1 and SW2	Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

"Figure 57 - Command Processing for Sign" shows a detailed procedure of the command processing.

Terminal **Card**



1. Computes the Temporary Signature Key (Ks):
   Ks = 3DES_16 [CTC] <K>
   CTC. Card Transaction Counter
   K. The relevant key that specifies in Kf and Kn. It can be either a signature key, a payment key or a log key.
2. It then calculates the Card Sign Certificate (CSC):
   CSC=3DES [00h||Pf||Tv_{2-0}||Bal_{2-0}] <Ks>

00h	Pf	Tv2	Tv1	Tv0	Bal2	Bal1	Bal0

Pf. The purse for which the CSC is computed and is formatted as:

Bit	7	6	5	4	3	2	1	0
	0	0	0	Purse File				

Tv_{2-0}. The three LSB of the previous transaction value.
Bal_{2-0}. The three LSB of the balance value after the previous transaction. This is optionally included, depending on the option setting in **Set Options** command.

Terminal		**Card**
	$\longleftarrow$	3. Prepares the response message as:

CSC3	CSC2	CSC1	CSC0

Figure 57 - Command Processing for Sign

Switch Protocol (SwtPrt)

Use this command to selects the card protocol or speed. Once this command has been executed, the card implements the new protocol or speed until either the next **Switch Protocol** command is executed or a new session is started.

Format

CLA	INS	P1	P2
80h	14h	P1	P2

Where:

P1 This byte indicates transmission bit rate as defined in ISO 7816-3 as the coding for the TA1 byte.

Assuming **F** is the clock conversion factor, **D** is the bit and **fs** is the terminal clock frequency, the bit duration (called ETU for Elementary Time Unit) is calculated as follows:

ETU = F/(D * fs)

The **F/D** ratio identifies the number of clock cycles in one ETU and is the absolute reference used by the card to set the communication speed.

Example:

F/D = 372 is the default value and corresponds to a baud rate of 9600 bds when **fs** = 3.5795 MHz.

F and **D** are referenced by the integers **Fi** and **Di** respectively (see *ISO 7816-3*). **Fi** and **Di** are coded on four bits and are read from a table as defined in ISO 7816-3. The **F/D** ratio is coded on one byte and is the concatenation of **Fi** || **Di**.

Note: Unlike the ISO specifications, **F** does not specify any maximum clock frequency and is only used to define the ETU.

"Table 23 - F/D Values for the 5V MPCOS-EMV Range" lists the values for **F/D** that are available for the 5V MPCOS-EMV range:

			D							
			1	**2**	**4**	**8**	**16**	**32**	**12**	**20**
		TA1 coding	.0001b	.0010b	.0011b	.0100b	.0101b	.0110b	.1000b	.1001b
	372	.0001b	**372**							
	558	.0010b								
	744	.0011b			**186**				**62**	
	1116	.0100b								
F	**1488**	.0101b								
	1860	.0110b								
	512	.1001b				**64**	**32**			
	768	.1010b		**384**	**192**	**96**				
	1024	.1011b								
	1536	.1100b								
	2048	.1101b								

Table 23 - F/D Values for the 5V MPCOS-EMV Range

"Table 24 - F/D Values for the 5V MPCOS-EMV Range at Different Speeds" lists the values for **F/D** that are available for the 5V MPCOS-EMV range at different communication speeds:

Comm. Speed Bauds	Reader Freq. MHz	F / D - ISO7816-3	P1 Value
19,200	3.6864	768 / 4	A3h
38,400	3.6864	768 / 8	A4h
57,600	3.6864	512 / 8	94h
115,200	3.6864	512 / 16	95h
9,600	3.5795	372 / 1	11h
19,200	3.5795	744 / 4	33h
57,600	3.5795	744 / 12	38h*

* These values are not supported by Gemplus readers.

Table 24 - F/D Values for the 5V MPCOS-EMV Range at Different Speeds

In these examples, 3.68 MHz corresponds to the clock frequency available from the Gemplus GCR400.

P2 Specifies a new protocol and card extra guard time. This has the following format:

Bit	7	6	5	4	3	2	1	0
	PRT		CEGT					

PRT specifies the new protocol, as follows:

This Value	**Protocol Specification**
0xb	T = 0
10b	T = 1
11b	T = 14 (additional protocol; not available for MPCOS-EMV, but can be implemented in a filter. See your technical consultant for further details.)

CEGT Specifies the Card Extra Guard Time. This is the number of extra Elementary Time Units (ETU)s to be inserted between each character in transmissions from the card. This can be used when the card is configured for high transmission speeds to reduce the rate of data flow from the card to a rate that is compatible with the terminal. This value should be 0h in most cases.

Response

SW1	SW2

Where:

SW1 and SW2 Hold the status byte values returned by the card. *“Appendix B - Card Contact Interface Return Codes”* lists the possible contents of SW1 and SW2.

Update Binary (UpdBin)

Use this command to update data in the transparent Elementary File (EF). This command can be executed only when the Group 1 access conditions are fulfilled.

Update Binary can be used to update the currently selected file (direct selection) or to select an EF by specifying its Short File Identifier (SFI) and then updated it (implicit selection). In implicit selection, the newly selected EF will become the current EF. In case of any error encounters during the command, it leaves the newly selected EF as the current EF.

Secure Messaging

- If the terminal is updating any file other than EF_{sc} or EF_{key}:
  It sends the data to be updated in plain text with the three rightmost significant bytes of the eight-byte cryptographic checksum. The amount of data that can be sent to the card is limited to 61 bytes (buffer size of 64 bytes minus three bytes for the cryptographic checksum). See *"IO Buffer Size"* on page 53 for further details.
- If the terminal is updating EF_{sc} or EF_{key}:
  It encrypts the updated data before sending it. The following shows the encryption process:
  Data field = Data field 1 || Data field 2
  Where:
  Data field 1 = $3DES^{-1}$ [Data 1 || Data 2] <Kats>
  Data field 1 = $3DES^{-1}$ [Data 3 || Data 4] <Kats>
  Kats is the temporary administration key computed by the card when it executed the **Select File Key** command.
  The contents of Data 1 to Data 4 will depend on the actual file being updated. See *"Table 25 - Contents of Data 1 to Data 4 for Update Binary"*.

	When Updating an EF_{sc}	**When Updating an EF_{key}**
Data 1	The values from the secret code file descriptor. See *"Secret Code Files"* on page 18 for further details. Mode MPN | SCR | UCR | 00h	The values from the key descriptor. See *"Key Files"* on page 15 for further details. Key Type | ONC | Key Version | Cks
Data 2	The four-byte secret code.	The four leftmost significant bytes of the key.
Data 3	Data 2	The four rightmost significant bytes of the key.
Data 4	The P1 & P2 parameters + a two-byte sum of the 14 previous data bytes.	The P1 & P2 parameters + a two-byte sum of the 14 previous data bytes.

Table 25 - Contents of Data 1 to Data 4 for Update Binary

Note: The terminal can only update one key or one secret code at a time. In EFkey, if the key file indicated in the access condition is 0, normal processing is used, otherwise secure messaging is used. See *"Chapter 3 - Access Conditions"*.

When the card receives the command, it decrypts the data and performs a redundancy check on it. If the redundancy check does not match the data, access will be denied and an appropriate error will be returned. If the redundancy check does match the data, the command will execute.

Format

CLA	INS	P1	P2	Lc	Data	Le
CLA	D6h	P1	P2	Lc	Data	00h/03h

Where:

CLA	Specifies transmission mode (normal or secure messaging):	
	00h	Transmit normally.
	04h	Transmit with secure messaging.

P1, P2 Specifies the file to be updated.

In a direct selection, the data is read from the currently selected EF by coding P1 and P2 as follows:

P1		P2
0	Offset (MSB)	Offset (LSB)

The seven lsb of P1 and the P2 byte specify the offset in data unit, from the beginning of the file to the first byte of the data.

In an implicit selection, the data is read from the selected EF by coding P1 and P2 as follows:

P1				P2
1	X	X	SFI	Offset

The five lsb of P1 specify the SFI of the file. The P2 byte specifies the offset in data unit, from the beginning of the file to the first byte of the data.

Note: If a key or a secret code is being updated using secure messaging, the offset must specify the position of the first byte of the code or key to be updated. This value will therefore be a multiple of two for a secret code and a multiple of three for a key.

Lc Specifies the length of the data (in bytes). The possible values at this field are:

Length of Update Data:	If transmitting without secure messaging.
16 bytes:	If updating EF_{sc} or EF_{key} with secure messaging.
Length of update data + 3 bytes:	Updates any file other than EF_{sc} or EF_{key} with secure messaging.

Note: When updating secret code in plain mode, the length must be eight bytes. The card internally computes the checksum for the secret code.

Le	00h	No response is expected, if transmitting without secure messaging.
	03h	Three bytes of response messaging, if transmitting with secure messaging.

Response

Response	SW1	SW2

Where:

Response — If secure messaging is used, the card generates a cryptographic response (see *"Secure Messaging"* on page 41). The terminal retrieves the three leftmost significant bytes of this checksum by executing the **Get Response** command.

The card generates the certificate as follows:

Cryptogram = 3DES [Data 1||Data 2] <Kats>

Certificate = Cryptogram $_{7–5}$

Kats is the temporary administration key computed by the card when it executed the **Select File** Key command.

The contents of Data 1 and Data 2 are as follows:

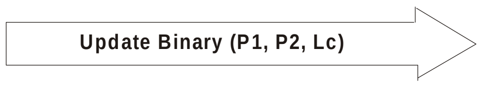
	EFsc	**EFkey**
Data 1	The values from the secret code file descriptor. See *"Secret Code Files"* on page 18 for further details.	The values from the key descriptor. See *"Key Files"* on page 15 for further details.
	Mode | MPN | SCR | UCR | 00h	Key Type | ONC | Key Version | Cks
Data 2	The P1 & P2 parameters + a two-byte sum which was sent with the incoming cryptogram (in Data 4).	The P1 & P2 parameters + a two-byte sum which was sent with the incoming cryptogram (in Data 4).

SW1 and SW2 — Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

“Figure 58 - Command Processing for Update Binary” shows a detailed procedure of the command processing.

Terminal **Card**



1. Updates data in the relevant file.
2. Prepares the response message only for transmission with secure messaging.

CRYCKS 7	CRYCKS 6	CRYCKS 5

CRYCKS$_{7–5}$ are the three leftmost significant bytes of the eight-byte cryptographic checksum.

3. Issues a **Get Response** command to retrieve the response message. This applies only to transmission with secure messaging.

4. Sends the response message.

Figure 58 - Command Processing for Update Binary

Update Record (UpdRec)

Use this command to update data in the structure Elementary File (EF). This command can be executed only when the Group 1 access conditions are fulfilled.

Update Record can be used to update the currently selected file (direct selection) or select an EF by specifying its Short File Identifier (SFI) and then updated it (implicit selection). In implicit selection, the newly selected EF will become the current EF. If an error is encountered during the command, it leaves the newly selected EF as the current EF.

Secure Messaging

This command can use secure messaging, as long as the length of the record plus the certificate does not exceed the internal buffer size. See *"IO Buffer Size"* on pag e53 for further details.

See *"Secure Messaging"* on page 41 for further information on the conditions which must be met before secure messaging can be used.

Format

CLA	INS	P1	P2	Lc	Data	Le
CLA	DCh	P1	P2	Lc	Data	Le

Where:

CLA Specifies transmission mode (normal or secure messaging):

00h	Transmit normally.
04h	Transmit with secure messaging.

P1, P2 Specifies the file to be updated, coded as follows:

P1	P2			
Record Number	SFI	1	0	0

P1 is the number of the record to be updated. For cyclic files, P1 must be zero which refers to the current record.

P2 specifies the SFI of the file for which the record is to be updated. In case of direct selection, SFI is set to zero which the file has been previously selected using **Select File** command.

Lc Specifies the length of the record to be updated. If secure messaging is being used, a value of 03h should be added. If Lc is greater than the available data length or is set to zero, the card will return the code "6Cnn", where "nn" is the available data length.

Le

00h	No response is expected, if transmission without secure messaging.
03h	Three bytes of response messaging, if transmission with secure messaging.

Response

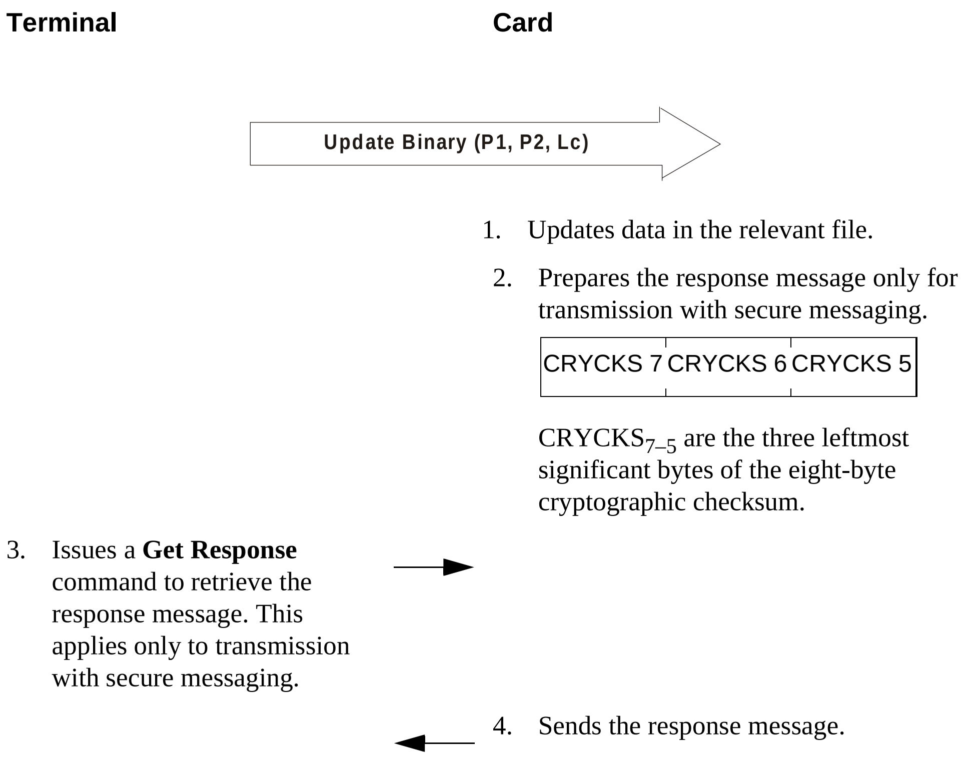
Response	SW1	SW2

Where:

Response	If secure messaging is used, the card generates a cryptographic response (see *"Secure Messaging"* on page 41). The terminal retrieves the three leftmost significant bytes of this checksum by executing the **Get Response** command.
SW1 and SW2	Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

"Figure 59 - Command Processing for Update Record" shows a detailed procedure of the command processing.



1. Updates data in the relevant file.
2. Prepares the response message only for transmission with secure messaging.

CRYCKS 7	CRYCKS 6	CRYCKS 5

CRYCKS$_{7-5}$ are the three leftmost significant bytes of the eight-byte cryptographic checksum.

3. Issues a **Get Response** command to retrieve the response message. This applies only to transmission with secure messaging.
4. Sends the response message.

Figure 59 - Command Processing for Update Record

Verify

Use this command to verify secret code to give authorization for some access conditions.

The command acts implicitly on the Secret Code File (EF_{sc}) located in the currently selected Dedicated File (DF). If there is no secret code file in the currently selected DF, the card returns an error code.

Note: After each execution of the **Verify** command in secure messaging mode, the values of the Off Nominal Counter (ONC) at the session key for all keys referenced are restored back, and the ONC mechanism is disabled. See *"Off-Nominal Counter (ONC) Mechanism"* on page 45 for details.

Once the card received the command, following steps will be carried out:

1. Increments the Secret Code Ratification (SCR) counter value. See *"Secret Code Files"* on page 18 for more information on the secret code ratification mechanism.
2. Compares the submitted secret code to the secret code number (from 0–7) in the EF_{sc} in the currently selected DF.
3. If the comparison is successful, the card set the appropriate bit to 1 in the local authorization register and then sets the SCR counter value to 0.

   If the comparison fails, the card returns an appropriate error code and checks if the SCR counter value reaches the Maximum Presentation Number (MPN). The secret code is locked, if the SCR counter value reaches MPN value, in this case, the unlock secret code has to be submitted.

Note: Locking a secret code will only block the functions protected by this code. Functions that protected by other codes will not be affected.

Secure Messaging

If the secret code is flagged as enciphered (see *"Secret Code Files"* on page 18), the terminal has to encipher the secret code value before submitting it. In case, the terminal establishes a temporary administration key (Kats) before executing the command.
The key to be used for this can be of any type. The terminal enciphers the secret code to be submitted using the following process:

Enciphered secret code = $3DES^{-1}$ [secret code] <Kats>

Kats is the Temporary Administration Key computed by the terminal and the card, when it executed the **Select File Key** or **Select Purse & Key** (using a log key) command.

Format

CLA	INS	P1	P2	Lc	Data
CLA	20h	00h	P2	08h	Data

Where:

CLA	Specifies transmission mode (normal or secure messaging):	
	00h	Transmit normally.
	04h	Transmit with secure messaging.
P2	SCN	Specifies the secret code number (from 0 to 7) in the EF_{sc} for the currently selected DF. The value FFh specifies the transport secret code.
Data	SCvalue	Holds the secret code to compare to the secret code specified in the SCN. If secure messaging is in use, this secret code is enciphered.

Response

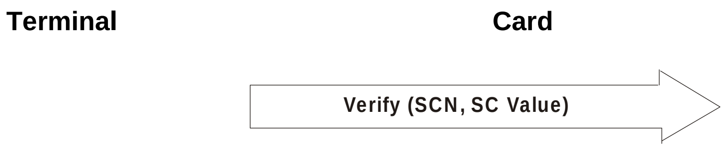
SW1	SW2

Where:

SW1 and SW2	Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

"Figure 60 - Command Processing for Verify" shows a detailed procedure of the command processing.



1. Increments the SCR counter value.
2. Compares the submitted secret code to the secret code number (from 0–7) in the EF_{sc} in the currently selected DF.
3. If the comparison is successful, the card set the appropriate bit to 1 in the local authorization register and then sets the SCR counter value to 0.
   If the comparison fails, the card returns an appropriate error code and it checks if the SCR counter value reaches MPN value. The secret code is locked, if the SCR counter value reaches MPN value.

Figure 60 - Command Processing for Verify

Write Binary (WrBin)

Use this command to write data to a transparent Elementary File (EF), by performing a logical OR operation using the current value of the write area and the value being written. This command can be executed only when the Group 2 access conditions are fulfilled.

Write Binary can be used to write to the currently selected file (direct selection) or select an EF by specifying its Short File Identifier (SFI) and then write to it (implicit selection). In implicit selection, the newly selected EF will become the current EF. If an error is encountered during the command, it leaves the newly selected EF as the current EF.

Secure Messaging

If the terminal is writing to any file other than EF_{sc} or EF_{key}:

It sends the data to be written in plain text with the three rightmost significant bytes of the eight-byte cryptographic checksum. The amount of data that can be sent to the card is limited to 61 bytes (buffer size of 64 bytes minus three bytes for the cryptographic checksum). See *"IO Buffer Size"* on page 53 for further details.

If the terminal is writing to an EF_{sc} or EF_{key}, it encrypts the data to be sent before sending it. The following shows the encryption process:
Data field = Data field 1 || Data field 2

Where:

Data field 1 = $3DES^{-1}$ [Data 1 || Data 2] <Kats>

Data field 1 = $3DES^{-1}$ [Data 3 || Data 4] <Kats>

Kats is the temporary administration key computed by the card when it executed the **Select File Key** command.

The contents of Data 1 to Data 4 will depend on the actual file being written to. See *"Table 26 - Contents of Data 1 to Data 4 for Write Binary"*

	When Updating an EF_{sc}	**When Updating an EF_{key}**
Data 1	The values from the secret code file descriptor. See *"Secret Code Files"* on page 18 for further details. Mode MPN SCR UCR 00h	The values from the key descriptor. See *"Key Files"* on page 15 for further details. Key Type ONC Key Version Cks
Data 2	The four-byte secret code.	The four leftmost significant bytes of the key.
Data 3	Data 2	The four rightmost significant bytes of the key.
Data 4	The P1 & P2 parameters + a two-byte sum of the 14 previous data bytes.	The P1 & P2 parameters + a two-byte sum of the 14 previous data bytes.

Table 26 - Contents of Data 1 to Data 4 for Write Binary

Note: The terminal can only write one key or one secret code at a time.
In EF_{key}, if the key file indicated in the access condition is 0, normal processing is used, otherwise secure messaging is used. See *"Chapter 3 - Access Conditions"*.

When the card receives the command, it decrypts the data and performs a redundancy check on it. If the redundancy check does not match the data, access will be denied and an appropriate error will be returned. If the redundancy check does match the data, the command executed.

Format

CLA	INS	P1	P2	Lc	Data	Le
CLA	D0h	P1	P2	Lc	Data	Le

Where:

CLA Specifies transmission mode (normal or secure messaging):

00h	Transmit normally.
04h	Transmit with secure messaging.

P1, P2 Specifies the file to be written.

In a direct selection, the data is written to the currently selected EF by coding P1 and P2 as follows:

P1		P2
0	Offset (MSB)	Offset (LSB)

The seven lsb of P1 and the P2 byte specify the offset in data unit, from the beginning of the file to the first byte of the data.

In an implicit selection, the data is written to the selected EF by coding P1 and P2 as follows:

P1				P2
1	0	0	SFI	Offset

The five lsb of P1 specify the SFI of the file. The P2 byte specifies the offset in data unit, from the beginning of the file to the first byte of the data.

Note: If a key or a secret code is being written using secure messaging, the offset must specify the position of the first byte of the code or key to be updated. This value will therefore be a multiple of two for a secret code and a multiple of three for a key.

Lc Specifies the length of the data (in bytes) to be written. The possible values at this field are:

Length of Update Data:	If transmitting without secure messaging.
16 bytes	If written to an EF_{sc} or EF_{key} with secure messaging.
Length of update data + 3 bytes:	Writing to any file other than EF_{sc} or EF_{key} with secure messaging.

Note: When writing a secret code in plain mode, the length must be eight bytes. The card internally computes the checksum for the secret code.

Le	00h	No response is expected, if transmitting without secure messaging.
	03h	Three bytes of response messaging, if transmitting with secure messaging.

Response

Response	SW1	SW2

Where:

Response — If secure messaging is used, the card generates a cryptographic response (see *"Secure Messaging"* on page 41). The terminal retrieves the three leftmost significant bytes of this checksum by executing the **Get Response** command.

The card generates the certificate as follows:

Cryptogram = 3DES [Data 1||Data 2] <Kats>

Certificate = Cryptogram $_{7-5}$

Kats is the temporary administration key computed by the card when it executed the **Select File** Key command.

The contents of Data 1 and Data 2 are as follows:

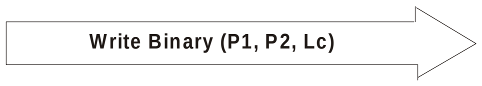
	EFsc	**EFkey**
Data 1	The values from the secret code file descriptor. See *"Secret Code Files"* on page 18 for further details.	The values from the key descriptor. See *"Key Files"* on page 15 for further details.
	Mode MPN | SCR | UCR | Cks	Key Type | ONC | Key Version | Cks
Data 2	The P1 & P2 parameters + a two-byte sum which was sent with the incoming cryptogram (in Data 4).	The P1 & P2 parameters + a two-byte sum which was sent with the incoming cryptogram (in Data 4).

SW1 and SW2 — Hold the status byte values returned by the card. *"Appendix B - Card Contact Interface Return Codes"* lists the possible contents of SW1 and SW2.

Command Processing

“Figure 61 - Command Processing for Write Binary” shows a detailed procedure of the command processing.

Terminal **Card**



1. Writes data in the relevant file.
2. Prepares the response message only for transmission with secure messaging.

CRYCKS 7	CRYCKS 6	CRYCKS 5

CRYCKS$_{7-5}$ are the three leftmost significant bytes of the eight-byte cryptographic checksum.

3. Issues a **Get Response** command to retrieve the response message. This applies only to transmission with secure messaging. →
4. ← Sends the response message.

Figure 61 - Command Processing for Write Binary

A

Default Answer To Reset

When a terminal powers up an MPCOS-EMV card, it responds by returning a standard Answer To Reset (ATR) at 9600 Bauds. The terminal can retrieve additional information using the **Get Response** command.

MPCOS-EMV Answer To Reset is compliant with the ISO 7816 standard.

Interface characters specify physical parameters of the card and logical characteristics of the subsequent exchange protocol.

Historical characters designate general information implemented by the card manufacturer. The specification for these characters is outside the scope of the standard specifications.

The ATR of MPCOS-EMV cards can thus be modified by Gemplus according to the card features. Future card versions or chip sources may require that these characters be changed.

Application software must not reject a card on the basis of a given ATR (to allow the use of future card versions).

However, it should be noted that a specific ATR could be customized to meet your requirements. Please contact Gemplus for more information.

Character Type	Byte	Value	Description
Initial and Format	TS	3Bh	Bit synchronization and structure, direct convention.
	T0	2Ah	TB1 present, 10 historical bytes.
Interface	TB1	00h	Vpp not connected.
Historical	T1	80h	Category indicator (status information in compact TLV object).
	T2	65h	Compact TLV for "Pre-issuing Data."
	T3	A2h	Family name (generic Gemplus products).
	T4	01h	Product name.
	T5	Reserved	Operating system (OS) version.
	T6	01h	Program version (5v range).
	T7	3Dh 31h	2 kilobytes* 8 or 32 kilobytes*
	T8	72h	Card capabilities Tag followed by two bytes.
	T9	D6h	DF selection by partial file name and file ID. EF management with short EF Id supported.
	T10	41h 43h	Data unit size is one byte. Data unit size is four bytes.

* Please use the **Get Card Information** command to obtain the actual memory size.

Table 27 - MPCOS-EMV AnswerTo Reset

This ATR shows the default protocol (T = 0, F = 372, D = 1, N = 0). The terminal can switch to another communication protocol (if necessary) using the **Switch Protocol (SwtPrt)** command.

The FMN and PRN can be used to identify a card as MPCOS-EMV R4 regardless of the EEPROM size, operating system version or chip type.

Warning: The Chip Identification Values (CIV) shown (T7) correspond to the same values used by the previous version of MPCOS-EMV. The **Get Card Information** command should be used to get the actual CIV to identify the card.

B

Card Contact Interface Return Codes

This appendix lists and describes the card status codes or "status bytes" that can be sent back by MPCOS-EMV R4 cards in response to a command.

Transmission Protocol Related Codes SW1 = 6xh

61 nnh	Command processed without error. The command prepared nn bytes of data that can be retrieved using the **Get Response** command.
61 03h	Command using secure messaging processed without error. Three-byte acknowledgement of cryptographic checksum is available.
62 81h	IADF error; a problem occurred during IADF interpretation and the relevant data may be corrupted.
62 83h	Warning: Invalid DF.
62 84h	File descriptor error.
63 00h	Authentication failed.
63 Cnh	Incorrect secret code submission. 'n' represents the number of tries left.
64 00h	Data integrity error.
65 81h	Write problem/Memory failure (ISO class byte).
67 00h	Incorrect length or address range error.
69 00h	No successful transaction executed during session.
69 81h	Cannot select indicated file, command not compatible with file organization.
69 82h	Access conditions not fulfilled: Secure messaging required and no key specified in access conditions. Secure messaging required and no temporary administration key established.
69 83h	Secret code blocked.
69 85h	No currently selected elementary file, no command to monitor (Monitor Command)/no transaction manager file (for **Select Purse & Key**).

69 87h	The value of the Off-Nominal Counter (ONC) has reached zero. The key cannot be used anymore.
69 89h	Limit on the maximum number of key usage in session is exceeded.
6A 80h	Incorrect parameters in data field (**Create File** command).
6A 82h	File not found.
6A 83h	Record not found.
6A 84h	Not enough memory to **Create File/Append Record.**
6A 88h	Key selection error (file not found, descriptor error, file is not a key file, key type error).
6B 00h	Incorrect reference (address, number or segment); illegal address.
6C nnh	Communication buffer exceeded; where nn is the maximum command length in the case of outgoing commands without secure messaging.
6C 40h	Wrong length (in secure messaging mode).
6D 00h	Command not allowed.
6E 00h	Incorrect application (CLA parameter of a command).

Application Related Return Codes SW1 = 9xh

90 00h	Command executed without error.
91 00h	Purse balance error (insufficient or exceeded) cannot perform transaction (for **Cancel Debit** command, this corresponds to an abort).
91 02h	Purse balance error occurred during the replacement debit step of a **Cancel Debit** command.
94 04h	Purse selection error or invalid purse (for **Cancel Debit** command, this is in the cancellation step).
94 06h	Invalid purse detected during the replacement debit step of a **Cancel Debit** command.
94 08h	Key file selection error; for example, file not found, offset incorrect, descriptor error or specified file not key file.
98 04h	Access authorization not fulfilled, wrong certificate, key file or Terminal Transaction Counter (TTC).
98 06h	Access Authorization in Debit not fulfilled for the replacement debit step of a **Cancel Debit** command.
98 20h	No temporary transaction key established (for **Cancel Debit** this means there was no temporary transaction key established or the debit certificate has already been retrieved).

C

Implementation of EMV Level I Features

Note: For further information, please refer to the EMV ’96 specifications.

File Structure

The following figure shows a two-level hierarchy structure of MPCOS-EMV R4.

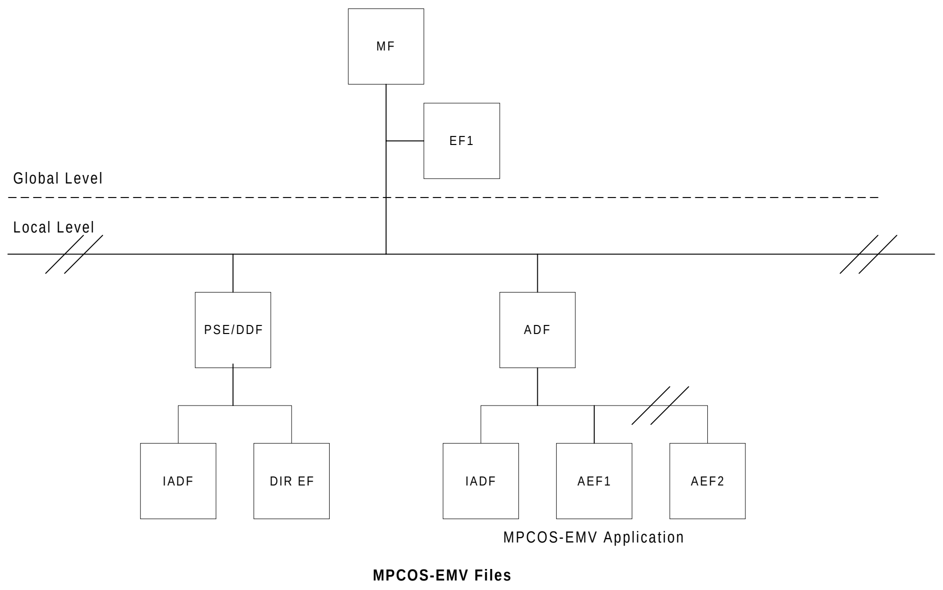


Figure 62 - File Structure of MPCOS-EMV R4

MPCOS-EMV R4 fully complies with Part I & III of the EMV ‘96 IC Card Specification for Payment Systems by supporting the approved physical characteristics and application selection procedure. The mapping of the card must include the structure of the Payment System Environment.

- The Payment System Environment DDF (Directory Definition File) called '1PAY.SYS.DDF01'.
- The Payment System Environment's File Control Information (FCI) which is contained in an IADF EF.
- The Payment System's Directory EF, containing the directory records of the Application Data File (ADF) held in the card.

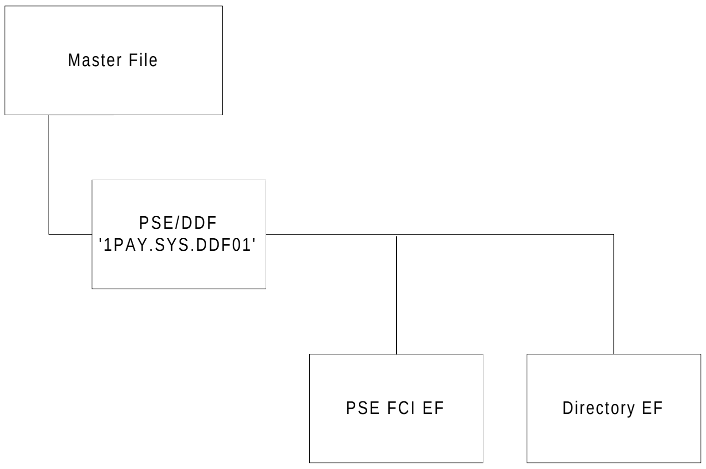


Figure 63 - File Structure for Payment System

This architecture is normally created during personalization. See *"MPCOS-EMV R4 Personalization"* on page 159 for details.

The MPCOS-EMV R4 card mapping also includes the ADFs, referred to in the Payment System Directory, which hold the elementary files of data dedicated to one application.

Each dedicated file must contain at least one system file called File Control Information (FCI) EF, which is needed to process the **Select File** command. This file is an IADF file and must be created using the structure described for IADFs in *"Chapter 2 - File and Data Structure"*.

Payment System Environment DDF

This Directory Definition File (DDF) is mapped onto a Dedicated File (DF) called '1PAY.SYS.DDF01'.

This DDF contains the PSE's FCI and Directory EFs, which are described in the next section.

Payment System Environment FCI EF

The selection of the Payment System Environment DDF (PSE DDF) returns its File Control Information (FCI). This FCI data should be in TLV format and it must be stored in Block 1 of an IADF EF. This file is an IADF file and must be created using the structure described for IADFs in *"Chapter 2 - File and Data Structure"*.

The following table defines the FCI returned by a successful selection of the PSE:

Tag	Length	Value	Presence
6Fh	variable	FCI Template	Mandatory
84h	14	‘1PAY.SYS.DDF01’	Mandatory
A5h	variable	FCI Proprietary Template	Mandatory
88h	1	SFI of the Directory Elementary File	Mandatory
5F2Dh	variable	Language Preference	Optional
9F11h	1	Issuer Code Table Index	Optional
BF0Ch	variable	FCI Issuer Discretionary Data	Optional

Table 28 - FCI of a Payment System Environment

Payment System Directory EF

The Payment System Directory contains entries to the ADFs in the card. Each ADF entry in a Payment System directory is encapsulated in an Application Template (tag 61h) coded within a record and holding information according to *“Table 30 - ADF Entry TLVs”*.

Tag	Length	Value
61h	variable	ADF Entry TLVs

Table 29 - ADF Entry Encapsulation, to Be Applied to Each ADF

Tag	Length	Value	Presence
4Fh	5–16	ADF name (**AID**)	Mandatory
50h	1–16	Application Label	Mandatory
9F12h	1–16	Application Preferred Name*	Optional
87h	1	Application Priority	Optional

*If the Application Preferred Name is defined in PSE Directory EF, Issuer Code Table Index is required.

Table 30 - ADF Entry TLVs

Note: Additional TLVs are described in the EMV '96 specifications and they can be implemented in MPCOS-EMV R4 cards.

Where the Application Priority byte is present, it will be coded according to the following rules:

b7	b6–b4	b3–b0	Definition
01b			Application cannot be selected without cardholder confirmation.
00b			Application may be selected without cardholder confirmation.
	xxx		RFU
		0000b	No priority assigned.
		xxxxb (except 0000b)	Order in which the application is to be listed or selected, ranging from 1–15, with 1 being the highest priority.

Table 31 - Rules Governing Application Priority

Payment System Application Identifiers (AIDs)

The structure of the **Application Identifier (AID)** is defined in the ISO 7816-5 standard and it consists of two parts:

- A Registered Application Provider Identifier (RID), 5 bytes, specific to a given application provider and assigned according to ISO 7816-5.
- An optional field of up to 11 bytes and assigned by the application provider. This field is known as a Proprietary Application Identifier Extension, (PIX) and it may contain any 0–11 byte value specified by the provider. The meaning of this field is defined only for the specific RID and does not need to be unique across different RIDs.

Implementation in the Terminal

This section briefly describes how commands and the mandatory data would be implemented by an EMV terminal to execute a transaction with the MPCOS-EMV R4 card.

Use of the Payment System Directory

In compliance with EMV specifications, a terminal supporting a large number of applications may use the Payment System Directory EF to determine the applications supported by the card.

1. The terminal begins with an explicit selection of the Payment System Environment DDF using the **Select File** command and the file name '1PAY.SYS.DDF01'.
2. The terminal reads the Payment System Directory EF the SFI of which has been coded in the File Control Information retrieved during the previous **Select File** command.
3. If the name of an ADF stored in the Payment System Directory EF matches one of the applications supported by the terminal, this application joins the 'candidate list' for final application selection.

4. When the terminal completes the list in the Payment System Directory EF, all ADFs held in the card will have been defined. The search is complete.

Selecting the Application to Be Run

Once the terminal determines the list of mutually supported applications, it must determine the single application to be run.

The terminal will either display the list to the cardholder and let him select the application or it will select the highest priority application from the list of mutually supported applications.

Bit 7 of the Application Priority byte determines which of the above methods is used. Bits 3–0 contain the priority number (where 1 is the highest priority).

Once the application to be run has been defined, the application must be selected, using the **Select File** command.

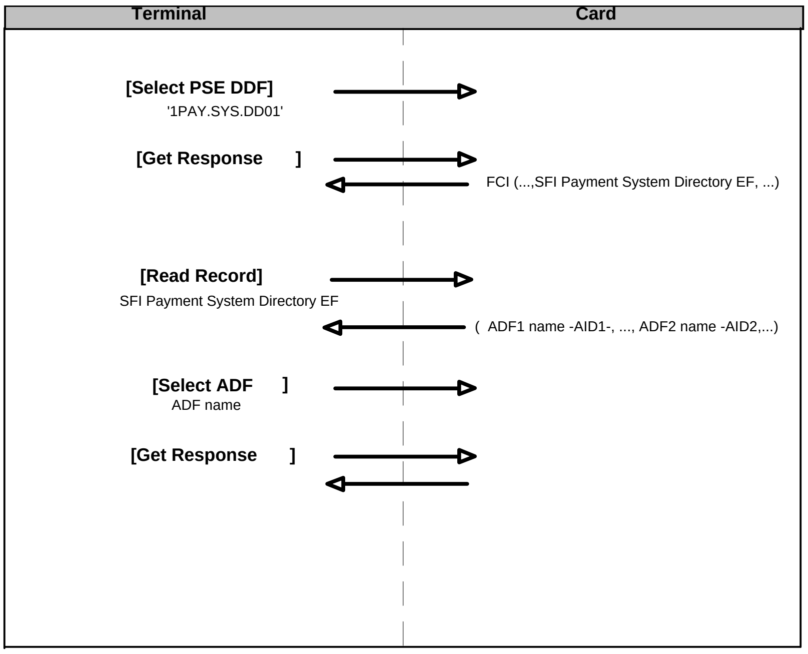

Terminal
Card
[Select PSE DDF]
'1PAY.SYS.DD01'
[Get Response]
FCI (...,SFI Payment System Directory EF, ...)
[Read Record]
SFI Payment System Directory EF
(ADF1 name -AID1-, ..., ADF2 name -AID2,...)
[Select ADF]
ADF name
[Get Response]

Figure 64 - Command Processing

At this point, the terminal will process the transaction by using the functions implemented for the selected application.

Examples of Implementation in Cards

This section describes two examples of implementation of the MPCOS-EMV R4 card.

The first example consists of a single E-Purse application implemented in the MPCOS-EMV R4 card.

The second example consists of implementing the MPCOS-EMV R4 card in a multi-application context: one is a MPCOS-EMV R4 E-Purse application and the other is an EMV-compatible application.

Single Application Context: E-Purse

In this example, the MPCOS-EMV R4 card only holds a private E-Purse application, which is implemented in an ADF. Only the application selection can call upon the use of the EMV functions. For the rest of the transaction, the card will make use of the MPCOS payment functions.

The Payment System Directory EF will only contain the E-Purse application AID and its related label, coded according to the rules described earlier in this appendix. It will be one of the following:

- The AID of the E-Purse Application Provider AID, registered with ISO.
- A concatenation of the Gemplus RID (assigned according to ISO) and a PIX precisely identifying the given application.

Finally, the Payment System Environment DDF and the E-Purse ADF will both contain a FCI File so that they can process the **Select File** command.

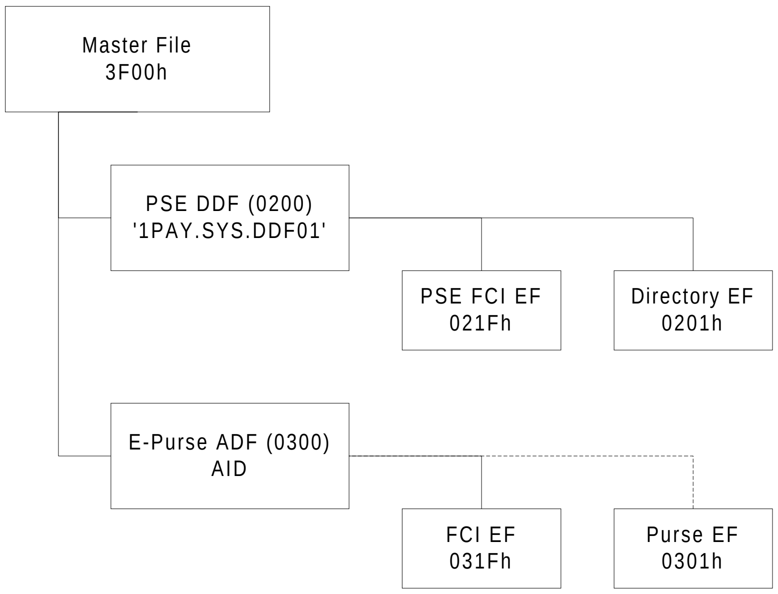


Figure 65 - Files Structure for E-Purse Single Application Context

Multi-Application Context

In this example, the MPCOS-EMV R4 card can hold several applications, including the following:

- A private MPCOS Electronic Purse application
- An EMV-compatible application using a single record

Each of these applications will be implemented in a separate ADF.

The Payment System Directory EF will contain the following:

- The E-Purse application AID and its related label, coded according to the rules described earlier in this appendix.
- The EMV application AID and its label.

The Payment System Directory EF could optionally store the Priority byte related to each application.

The Payment System Environment DDF, the E-Purse ADF and the EMV application ADF will all contain a FCI File, so that the **Select File** command can be processed.

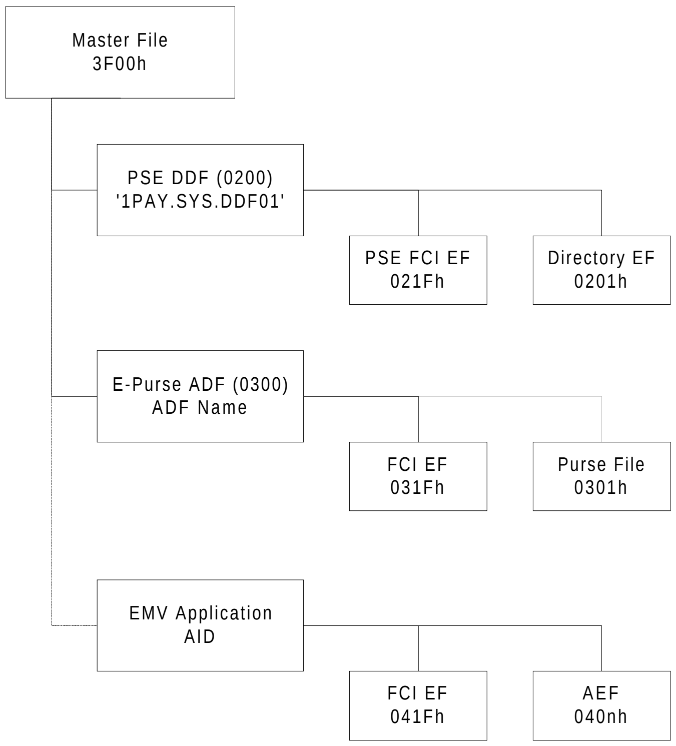


Figure 66 - File Structure for Multi-Application Context

MPCOS-EMV R4 Personalization

This section draws the developer's attention to the elements that must be taken into account in order to personalize the MPCOS-EMV R4 cards correctly.

The personalization process will initialize the EMV memory structure, as well as cardholder-related programming data. This takes into account the Directory Definition File (DDF); the Payment System Directory EF and the ADF initialization in accordance with the rules already described in this appendix.

Creating DDF and ADF

Use the standard MPCOS-EMV **Create File** command with the appropriate options to create the Payment System DDF and an ADF for each application.

Creating FCI Files

In order to make a DF accessible by the **Select File** command, you will need to create an FCI EF under that DF. You must therefore create an FCI EF for the DDF and for each ADF.

You will use the standard MPCOS-EMV **Create File** command, but with 09h as the FDB value.

The length of the FCI file will take into account the overhead associated with the TLV coding.

Filling FCI File

Once created, the FCI EF needs to be completed with the relevant information by using the standard MPCOS-EMV **Update Binary** command.

DDF FCI File

The DDF FCI file must be completed with the DDF Name, '1PAY.SYS.DDF01' and the Short File Identifier of the Payment System Directory EF, 01h.

The data will be encapsulated in a template with a 6Fh Tag and a Length. It will then be coded as a Tag Length Value constructed data object, as follows:

Tag	Length	Value
6Fh	variable	FCI Template
84h	14	1PAY.SYS.DDF01
A5h	variable	FCI Proprietary Template
88h	1	SFI of the Payment Directory EF

Table 32 - TLV of a DDF FCI File

The following is an example of personalization (all values are in hexadecimal):

1A	17	55	17
▨ 6F	15	▨ 84	0E
31	50	41	59
2E	53	59	53
2E	44	44	46
30	31	▨ A5	03
▨ 88	01	01	

Table 33 - Personalization Example

ADF FCI File

The ADF FCI file(s) must be completed with the ADF Name and the Application Priority Indicator.

The data will be encapsulated in a template with a Tag 6Fh and a Length and coded as a Tag Length Value constructed data object, with the following tags:

Tag	Length	Value
6Fh	variable	FCI Template
84h	5–16	DF name
A5h	variable	FCI Proprietary Template
87h	1	Application Priority Indicator

Table 34 - TLV of an ADF FCI File

The following is an example of the contents of an ADF FCI EF data field (all values are in hexadecimal):

13	10	55	10
6F	0E	84	07
	Application	Identifier	
			A5
03	87	01	00

Table 35 - Example of an ADF FCI EF Data Field

Creating the Payment System Directory EF

Use the standard MPCOS-EMV **Create File** command with the appropriate options, notably with '05h' as FDB value, reserved by ISO for an elementary file with linear variable record structure and supporting simple TLV template.

Once created, the structured file should be completed with the appropriate data. Use the **Update Binary** command to create the record structure and record data field, according to the *“Figure 67 - Record Structure”*:

Record Length (1–255)	Record Template

Figure 67 - Record Structure

Filling the Payment System Directory EF

This file must be completed with the entries to the ADFs held in the card, as described in *“Payment System Environment FCI EF”* on page 154: the AID(s), the label(s) and optionally the Application Preferred Name(s) and the Application Priority Indicators.

The data will be encapsulated in a template with a 70h Tag and a Length. It will then be coded as a Tag Length Value constructed data object, with the following tags:

Tag	Length	Value	Presence
4Fh	5–16	ADF name (AID)	Mandatory
50h	1–16	Application Label	Mandatory
9F12h	1–16	Application Preferred Name	Optional
87h	1	Application Priority	Optional

Table 36 - TLV of an ADF FCI EF File

The following is an example of a Payment System Directory EF data field format, containing the entries to the MPCOS-EMV R4 E-Purse application (all values are in hexadecimal):

13	70	11	61
0F	4F	07	
	Application	Identifier	
		50	04
	Application	Label	

Table 37 - Example of a Payment System Directory EF Data Field

D

Implementation of EMV Level II Features

Note: For further information, please refer to the EMV '96 specifications.

File Structure

"Figure 68 - File Structure of MPCOS-EMV R4" shows a two-level hierarchy structure that includes MPCOS-EMV files for Magnetic Stripe Interface (MSI) applications.

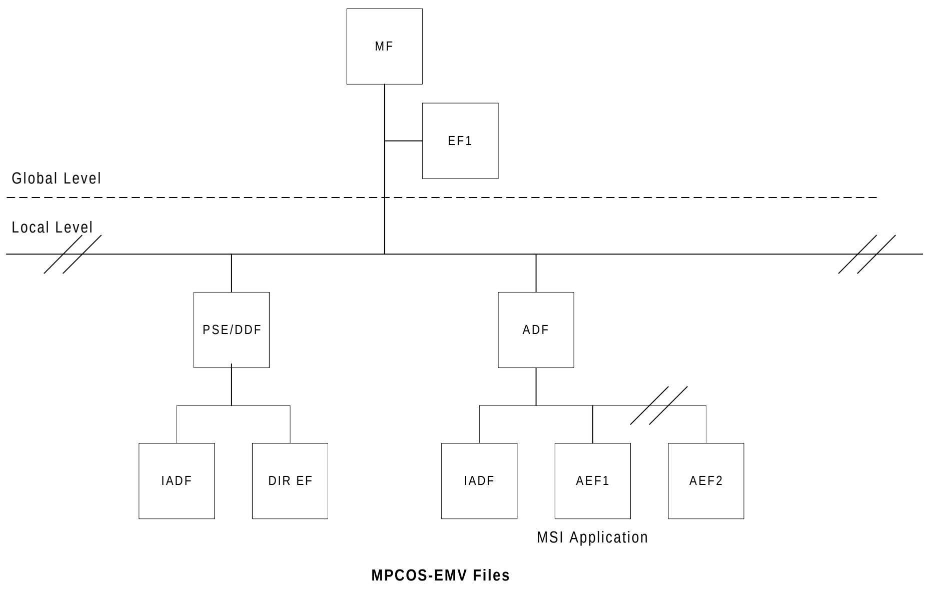


Figure 68 - File Structure of MPCOS-EMV R4

The MPCOS-EMV R4 files are mapped as follows:

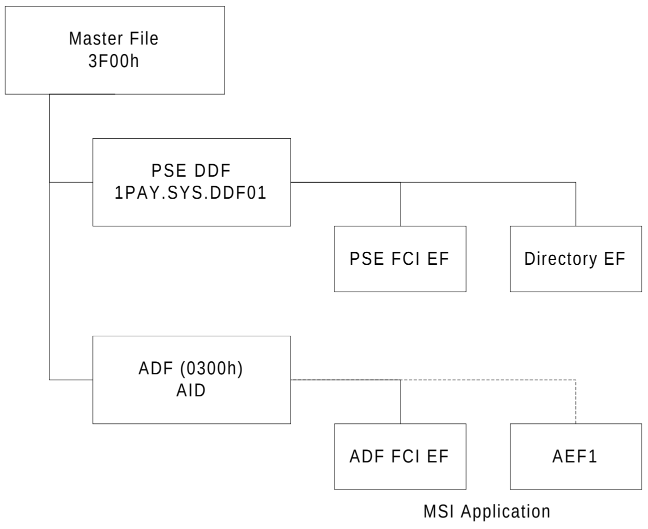


Figure 69 - MPCOS-EMV R4 Card Mapping

Payment System Environment DDF

To ensure compatibility with EMV specifications, this Directory Definition File (DDF) is mapped onto a Dedicated File (DF) called '1PAY.SYS.DDF01'.

This DDF contains the Payment System Directory EF, which is described in the next section.

Payment System Environment FCI EF

The selection of the Payment System Environment DDF (PSE DDF) returns its File Control Information (FCI). This FCI data should be in TLV format and it must be stored in Block 1 of an IADF EF. This file is an IADF file and must be created using the structure described for IADFs in *“Chapter 2 - File and Data Structure”*.

The following table defines the FCI returned by a successful selection of the PSE:

Tag	Length	Value	Presence
6Fh	variable	FCI Template	Mandatory
84h	14	DF Name	Mandatory
A5h	variable	FCI Proprietary Template	Mandatory
88h	1	SFI of the Directory Elementary File	Mandatory
5F2Dh	variable	Language Preference	Optional
9F11h	1	Issuer Code Table Index	Optional
BF0Ch	variable	FCI Issuer Discretionary Data	Optional

Table 38 - FCI of a Payment System Environment

Payment System Directory EF

The Payment System Directory contains entries to the ADFs in the card. Each ADF entry in a Payment System directory is encapsulated in an Application Template (Tag 61h) coded within a record and holding information according to *"Table 40 - ADF Entry TLVs"*.

Tag	Length	Value
61h	variable	ADF Entry TLVs

Table 39 - ADF Entry Encapsulation, to Be Applied to Each ADF

Tag	Length	Value	Presence
4Fh	5–16	ADF name (AID)	Mandatory
50h	1–16	Application Label	Mandatory
9F12h	1–16	Application Preferred Name	Optional
87h	1	Application Priority	Optional

Table 40 - ADF Entry TLVs

Note: Additional TLVs are described in the EMV '96 specifications and they can be implemented in MPCOS-EMV R4 cards.

Where the Application Priority byte is present, it will be coded according to the following rules:

b7	b6–b4	b3–b0	Definition
1b			Application cannot be selected without cardholder confirmation.
0b			Application may be selected without cardholder confirmation.
	xxx		RFU
		0000b	No priority assigned.
		xxxx (except 0000b)	Order in which the application is to be listed or selected, ranging from 1–15, with 1 being the highest priority.

Table 41 - Rules Governing Application Priority

Payment System Application Identifiers (AID)

The structure of the **Application Identifier (AID)** is defined in the ISO 7816-5 standard and it consists of two parts:

- A Registered Application Provider Identifier (RID), 5 bytes, specific to a given application provider and assigned according to ISO 7816-5.
- An optional field of up to 11 bytes and assigned by the application provider. This field is known as a Proprietary Application Identifier Extension, (PIX) and it may contain any 0–11 byte value specified by the provider. The meaning of this field is defined only for the specific RID and does not need to be unique across different RIDs.

Application Data File (ADF)

An Application Data File (ADF) contains the entry points to the application.

ADFs are Dedicated File (DFs), the name of which is coded in the Payment System Directory EF.

Application Data File FCI EF

Like DDFs, an ADF returns its File Control Information (FCI) when selected. This data is coded in TLV format and it is stored in Block 1 of an IADF.

The following table defines the FCI returned by a successful selection of an ADF.

Tag	Length	Value	Presence
6Fh	variable	FCI Template	Mandatory
84h	7	A0 00 00 00 03 10 10	Mandatory
A5h	variable	FCI Proprietary Template	Mandatory
87h	1	Application Priority Indicator	Optional
5F2Dh	variable	Language Preference	Optional
9F11h	1	Issuer Code Table Index	Optional
9F12h	variable	Application Preferred Name	Optional

Table 42 - FCI of a Application Data File

MPCOS-EMV R4 supports the full EMV transaction flow with the necessary personalization data specified in Template 1 - Magnetic Stripe Image (MSI) of the *Visa Smart Debit/Smart Credit Personalization Templates*. Part of this personalization data is stored in Block 2 of the IADF. Please refer to *“Internal Application Data File (IADF)”* on page 20 for more information about IADFs.

Block 2

Tag	Length	Value	Presence
		Application Flag	Mandatory
		ATC + Checksum	Mandatory
		Backup ATC + Checksum	Mandatory
		Maximum ATC	Mandatory
		Derivation Key Index	Mandatory
		Cryptogram Version Number	Mandatory
80h	06h	GPO Response Message in Format 1	Mandatory
		Application Interchange Profile	Mandatory
		Application File Locator	Mandatory

Table 43 - Block 2 of the ADF

Application Elementary File (AEF)

To complete the personalization requirements for an MSI, the rest of the data can be entered into the Application Elementary Files (AEFs). AEFs contain the data elements used by the application it is processing. Specifically, these elementary files are record files and the data in TLV format can be retrieved using the **Read Record** command. Please refer to *Visa Smart Debit/Visa Smart Credit Personalization Templates* for more information about the data requirements of an MSI application.

Example of an MSI Implementation

This section describes an example of an MSI implementation based on the following file structure:

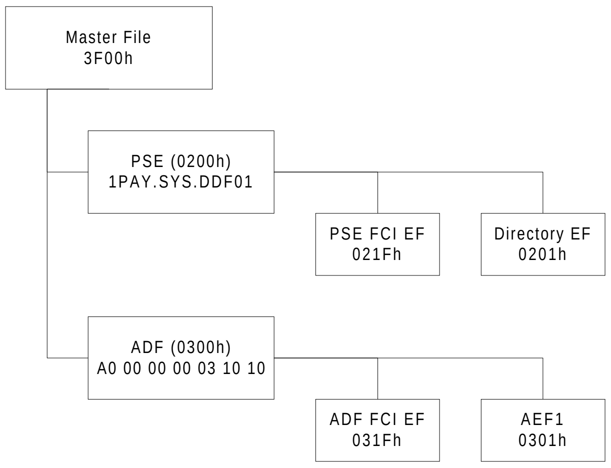


Figure 70 - MSI Mapping

1. Set the PSE FCI EF (021Fh) with the following attributes:
   – FDB = 09h (IADF)
   – Locked for read, write and update

   Enter the data contents in Block 1, in TLV format, as follows:

Tag	Length	Value (in hexadecimal)	Description
55h	38		MPCOS-EMV R4 Proprietary Template
6Fh	36		FCI Template
84h	14	31 50 41 59 2E 53 59 53 2E 44 44 46 30 31	DF Name (1PAY.SYS.DDF01)
A5h	18		FCI Proprietary Template
88h	1	01	SFI of Directory EF
5F2Dh	8	65 73 65 6E 66 72 64 65	Language Preference (en es fr de)
9F11h	1	01	Issuer Code Table Index

2. Set the PSE Directory EF with the following attributes:
   - SFI = 01h
   - FDB = 05 (linear variable record structure supporting simple TLV template)
   - Locked for update and append

   Enter the following TLVs in Record 1:

Tag	Length	Value (in hexadecimal)	Description
70h	38		AEF Data Template
61h	36		Application Template
4Fh	7	A0 00 00 00 03 10 10	ADF Name (AID)
50h	11	56 49 53 41 20 43 52 45 44 49 54	Application Label (VISA CREDIT)
9F12h	8	56 49 53 41 20 4D 53 49	Application Preferred Name
87h	1	01	Application Priority Indicator

3. Set the ADF FCI EF with the following attributes:
   - FDB = 09h (IADF)
   - Locked for read, write and update

   Enter the following TLVs in Block 1:

Tag	Length	Value (in hexadecimal)	Description
55h	44		MPCOS-EMV Proprietary Template
6Fh	42		FCI Template
84h	7	A0 00 00 00 03 10 10	DF Name
A5h	31		FCI Proprietary Template
50h	11	56 49 53 41 20 43 52 45 44 49 54	Application Label (VISA CREDIT)
87h	1	01	Application Priority Indicator
5F2Dh	8	65 73 65 6E 66 72 64 65	Language Preference (en es fr de)
9F11h	1	01	Issuer Code Table Index

Enter the following TLVs in Block 2:

Tag	Length	Value (in hexadecimal)	Description
		00	Application Flag
		00 00 00	ATC + Checksum
		00 00 00	Backup ATC + Checksum
		FF FF	Maximum ATC
		00	Derivation Key Index
		0C	Cryptogram Version Number
80h	6		GPO Response Message in Format 1
		18 00	Application Interchange Profile
		08 01 02 00	Application File Locator

4. Set the AEF (0301) with the following attributes:
   - SFI = 01h
   - FDB = 05h (linear variable record structure supporting simple TLV template)
   - Locked for update and append

   Enter the following TLVs in Record 1:

Tag	Length	Value (in hexadecimal)	Description
70h	55		AEF Data Template
57h	17	47 61 73 90 01 01 00 10 D1 01 22 01 01 23 45 67 89	Track 2 Equivalent Data*: • Primary Account Number • Field Separator ('D') • Expiration Date (YYMM) • Service Code • PIN Verification Field • Discretionary Data • Padding ('F')
5F20h	8	4D 53 49 20 43 41 52 44	Cardholder Name* (MSI Card)
9F1Fh	22	30 31 30 32 30 33 30 34 30 35 30 36 30 37 30 38 30 39 30 41 30 42	Track 1 Discretionary Data*

* The given values are for illustration purpose only. Please refer to the issuer profile for the actual values.

Enter the following TLVs in Record 2:

Tag	Length	Value	Description
70h	120		AEF Data Template
9F08h	2	00 84	Application Version Number (VIS 1.3.2)
8Ch	2	95 05	Card Risk Management Data Object List 1
8Dh	2	8A 02	Card Risk Management Data Object List 2
9F0Eh	5	00 10 00 00 00	Issuer Action Code—Denial
9F0Fh	5	10 40 00 98 00	Issuer Action Code—Online
9F0Dh	5	10 40 00 88 00	Issuer Action Code—Default
5F28h	2	08 40	Issuer Country Code
9F07h	2	FF00 (for Service Code = 2)	Application Usage Control
5F24h	3	10 12 31 (31 December 2010)	Application Expiration Date (YYMMDD)
5Ah	8	47 61 73 90 01 01 00 10	Application Primary Account Number (PAN)
5F34h	1	00	Application PAN Sequence #
9F0Bh	34	47 45 4D 50 4C 55 53 20 4D 41 47 4E 45 54 49 43 20 53 54 52 49 50 45 20 49 4D 41 47 45 20 43 41 52 44	Cardholder Name Extended (GEMPLUS MAGNETIC STRIPE IMAGE CARD)
8Eh	14	00 00 00 00 00 00 00 00 1E 03 02 03 1F 00	Cardholder Verification Method List - Signature, online PIN, no CVM required.

E

T = 1 Protocol

Scope

This transmission protocol is defined as the T = 1 protocol in the TD_i byte of the Answer To Reset.

T = 1 Principles

T = 1 is a half duplex protocol. It is a block-oriented protocol, designed according to the principle of layering of the OSI reference model.

The T = 1 protocol features are fully described in ISO/IEC 7816-3 standard (Clause 9) and ISO/IEC 7816-4 standard.

Selecting T = 1

The T = 1 protocol can be selected using two mechanisms (see relevant clauses):

- **Switch Protocol** command, a proprietary command enabling the user to change the current protocol any time during a session.
- Double Reset mechanism, allowing the card to change the current protocol according to the reset occurrences (please contact a Gemplus technical consultant for further details).

Block Frame

A block is a series of characters containing data bytes. A block consists of the following fields:

- Prologue field (mandatory)
- Information field (optional)
- Epilogue (mandatory)

Prologue Field			Information Field	Epilogue Field
NAD	PCB	LEN	INF	EDC
1 byte	1 byte	1 byte	0 to IFSC bytes	1 byte

Basic Elements of a Block

Prologue Field

This field is mandatory and consists of three bytes: the node address (NAD), the protocol byte (PCB) and the length (LEN).

Node Address (NAD)

The Node Address (NAD) byte is used to identify the source and the intended destination of the block. It may be used to distinguish between multiple logical connections when they coexist.

The bits b0 to b2 indicate the source node address (SAD) and the bits b4 to b6 indicate the destination node address (DAD). The bits b3 and b7 must be set to 0.

The NAD of the first block sent by the interface device defines a logical connection between the SAD and DAD addresses. Subsequent blocks in which the NAD field contains the same SAD/DAD address pair will have the same logical connection.

Warning: Other logical connections (including wrong NAD bytes) will be disregarded by the card.

When addressing is not used, the values of SAD and DAD are set to 0.

Protocol Byte (PCB)

Protocol Byte (PCB) is used to transmit the information that is used to control data transmission:

- Block type identification (I-blocks, R-blocks or S-blocks)
- Sequence control (R-blocks and I-blocks)
- Chaining control (I-blocks)
- Error identification (R-blocks)

Length (LEN)

Length (LEN) indicates the number of bytes transmitted in the information field of the block (from 0 to 254).

Information Field (INF)

Information Field (INF) is optional. When present, INF transmits either application data in I-blocks or control and status information in S-blocks. The number of bytes transmitted is indicated by LEN.

Epilogue Field (EDC)

This field is mandatory. It contains the error detection code (EDC) of the transmitted block, in LRC mode. This field is one byte in length and is calculated as the XORing of all the bytes starting with the NAD through to the last byte of the information field.

Supported Blocks

I-Blocks

I-blocks are used to transmit applicative information in both directions:

The interface device uses I-blocks to transmit command headers and incoming data, the card uses I-blocks to transmit outgoing data and the status bytes SW1, SW2.

Warning: When a command is transmitted, the first block must contain either the whole command header (case 1s) or the first five bytes of the block (cases 2s, 3s and 4s).

In compliance with IFSC, the info field of an incoming I-block must not exceed 69 bytes.

R-Blocks

R-blocks are used either to convey block acknowledgment or to initiate block repetition in case of transmission error.

In error-free operation, R-blocks are used for chaining (block sequencing).

In error mode, R-blocks may be used to indicate the error type (EDC/Parity error, Other error).

S-Blocks

S-blocks are supervisory blocks used to exchange control information between the interface device and the card. Its information field may be present depending on the controlling function.

The card accepts the following S-blocks:

- S(RESYNCH request):The IFD initiates a protocol resynchronization.
- S(ABORT request):Chaining abortion initiated by the IFD.
- S(ABORT response):IFD acknowledgment after reception of a S(ABORT request) from the card.
- S(IFS request):Block indicating IFS offer. INF (1 byte in length) transmits the new IFSD value, in the range [16, 254]. Value 255 is forbidden.

The card can send the following S-blocks:

- S(RESYNCH response):Acknowledgment after reception of a S(RESYNCH request) from the IFD. The card resets all the communication parameters to the initial values.
- S(ABORT request):Chaining abortion initiated by the card.
- S(ABORT response):Acknowledgment after reception of a S(ABORT request) from the IFD.

- S(IFS response):Acknowledgment after reception of a S(IFS request) from the IFD. The information field contains the IFSD value accepted by the card. It can either echo the requested value or offer the minimal value accepted by the card (that is, 16).

Specific Interface Parameters

This section describes the protocol initial values in use in the card.

IFSD and IFSC

In compliance with the standard, the default IFSD value is 32 (20h). This value may be changed during the session by sending an S(IFS request) block to the card.

The IFSC value is 69 (45h), which means that the card can accept I-blocks with up to 64 bytes of application data (case 3s). When processing commands with higher data flows, the chaining mechanism must be used.

Character Waiting Time

The character waiting time (CWT) is defined as the maximum time between the leading edges of two consecutive characters in the same block.

CWT is calculated by the following formula:

CWT = (2cwi + 11) work ETU, where CWI is the character waiting time integer.

The card uses CWI = 2, so CWT = 15 ETUS.

CWT is used by the card to detect the end of the block.

Block Guard Time and Block Waiting Time

The Block Waiting Time (BWT) is defined as the maximum time between the leading edges of the last character which gave access to the card and the first character to be sent by the card.

BWT is calculated using the following formula:

BWT = 11 etu + 2 bwi * 960 * (372/f)

BWT is used by the interface device to detect an unresponsive card.

The card uses a default value of bwi=4, therefore BWT is approximately 1.6s when
f = 3.57 MHz.

The block guard time is defined as the minimum time between the leading edges of two consecutive characters sent in opposite directions. The value of BGT is 22 etu (compliant with ISO 7816-3 standard, Clause 9.5.2.3).

Consequently, the delay between the last character of a received block and the first character of a transmitted block is greater than BGT and is less than BWT.

Chaining Rules

Chaining is a function of the block transmission protocol. It allows either the card or the IFD to split the information into several blocks.

The chaining of blocks is controlled by the M-bit in the PCB of an I-block:

M = 1	Chained data follows in subsequent block(s)
M = 0	Last block of chain

I-blocks are sent with alternating sequence numbers, 0, 1, 0, 1,…. Each block is acknowledged by the receiver using an R-block, the value of which will be the number of the next expected I-block. Therefore an I-block with a sequence number of 0 will be acknowledged by an R-block with a sequence number of 1 and vice-versa.

If an I-block is incorrectly received, the R-block will return the value it has received. For example, if an I-block with a sequence number of 0 is incorrectly received, the R-block will return a 0 instead of a 1. This indicates that the I-block was not received correctly and must be resent.

If an inconsistency is detected in the length of the transmitted data, the card aborts the process using a S(ABORT request).

Warning: In the case of incoming chaining, the card accepts blocks with LEN in the [0, IFSC] range. However, the first block (conveying the command header) must contain at least the first five bytes of the command (cases 2s, 3,s and 4s) or the first four bytes of the command (case 1s).

In the case of outgoing chaining, the card may split the information data in the following circumstances:

- Data length exceeds IFSD
- Data length exceeds IFSC
- The application layer separated the data into different blocks.

Error Handling

The card detects the transmission and sequence errors (parity or EDC error, invalid PCB, invalid LEN, invalid format, invalid sequence number).

Resynchronization in this protocol is attempted at three consecutive levels. If one level is unsuccessful, then the next level is tried.

For the card, the three synchronization levels are as follows:

- Retransmission of blocks
- Use of S(RESYNCH response)
- If no action is received from the interface device, the card becomes unresponsive

Terminology

Abbreviations

3DES	Triple Data Encryption Standard
AAC	Application Authentication Cryptogram
AC	File Access Condition
ACD	Access Control Descriptor
ACM	Applicative Countermeasure
ADF	Application Data File
AEF	Application Elementary File
AFL	Application File Locator
AID	Application IDentifier
AIP	Application Interchange Profile
APDU	Application Protocol Data Unit
ARQC	Authorization Request Cryptogram
ATC	Application Transaction Counter
ATR	Answer To Reset
BCD	Binary Coded Decimal
BSN	Batch Secret Number
CAC	Credit Access Condition
CBC	Card Balance Certificate

CC	Credit Cryptogram
CDC	Card Debit Certificate
CLA	Application Class byte of the command message
CPLC	Card Production Life Cycle
CR	Card Cryptogram
CRnd	Card Random Number
CRYCKS	Cryptographic Checksum
CSC	Enciphered Secret Code
CSN	Card Serial Number
CTC	Card Transaction Counter
CV	Counter Value
CVN	Cryptogram Version Number
CVR	Card Verification Result
DDF	Directory Definition File
DF	Dedicated File
DKI	Derivation Key Index
DPA	Differential Power Analysis
DV	Debit Value
EAC	External Authentication Cryptogram
ECB	Electronic Code Book
EF	Elementary File
EMV	Europay Mastercard Visa
ETU	Elementary Time Unit
FCI	File Control Information
FDB	File Descriptor Byte
FN	File Number
FP	File Pedigree
GMKadm	Grandmother Administration Key
GMKaut	Grandmother Authentication Key
GMKaward	Grandmother Award Key
GMKpay	Grandmother Payment Key

GMKsign	Grandmother Signature Key
GMsc	Grandmother Secret Code
GMseed	Grandmother Seed Data
IAC	Internal Authentication Cryptogram
IADF	Internal Application Data File
IC	Integrated Circuit
ID	Identifier
IFD	Interface Device
IFS	Information Field Size
IFSC	Information Field Size of the Card
IFSD	Information Field Size for the Interface Device
INF	Information Field
INS	Instruction Code
Kadm	Administration Key
Kaut	Authentication Key
Kaward	Award Key
Kmac	MAC Key
Kpay	Payment Key
Ksign	Signature Key
KVC	Key Verification Code
Lc	Data Field Length
Le	Expected Data Length
lsb	Least Significant Bit(s)
LSB	Least Significant Byte(s)
LTI	Last online Transaction Incomplete
MAC	Message Authentication Code
MF	Master File
MKadm	Mother Administration Key
MKaut	Mother Authentication Key
MKaward	Mother Award Key
MKpay	Mother Payment Key

MKsign	Mother Signature Key
MPCOS	Multi-application Payment Chip Operating System
MPN	Maximum Presentation Number
MRN	Maximum Reply Number
msb	Most Significant Bit(s)
MSB	Most Significant Byte(s)
Msc	Mother Secret Code
ONC	Off-Nominal Counter
OS	Operating System
P1/P2	Parameter 1/ Parameter 2 of the command header
PIN	Personal Identification Number
PIX	Proprietary application Identifier eXtension
PSE	Payment System Environment
RF	Radio Frequency
RFU	Reserved for Future Use
RID	Registered application provider IDentifier
Rnd	Random number
ROM	Read Only Memory
RSC	Ratification Secret Code
SAM	Security Access Module
SC	Secret Code
SCR	Secret Code Reference
SDC	Cancel Debit Cryptogram
SFI	Short File Identifier
SK	Session Key
SM	Secure Messaging
SMC	Secure Messaging Cryptogram
SW1/SW2	Status Word 1/Status Word 2
TC	Transaction Certificate
TLV	Tag Length Value
TMC	Terminal MAC counter

TPDU	Transmission Protocol Data Unit
TRnd	Random number generated by the terminal
TSC	Transport Secret Code
TSK	Temporary Session Key
TTC	Terminal Transaction Counter
UCR	Unblocking Code Reference
WTX	Waiting Time Extension

Glossary

Access Conditions	Access conditions specify the following parameters: - The secret codes to be submitted before access can be granted to a file. - The key file that holds the key to be used for secure messaging to a file.
Administration Commands	Commands that perform administrative function such as creating files, writing to files and reading from files.
Administration Key	Type of cryptographic key used for the computation of temporary administration keys, authentication and secure messaging.
Administration Session	Session during which administration commands is performed.
Authentication	The process of comparing two cryptograms, one generated by the terminal, the other by the card, to make sure that the card has not been forged.
Authorization Register	Eight-bit RAM registers that keeps a record of the secret codes that have been successfully submitted. MPCOS-EMV compares the contents of the register with a file’s access conditions to establish whether or not access should be granted to that file.
Batch Card	The card that holds the Mother System Key for a set of MPCOS-EMV cards.
Certificate	The MPCOS-EMV system generates certificates after performing certain transactions such as reading a purse balance or debiting a purse. They confirm the authenticity of a transaction. You may use such certificates at a later date to verify a transaction.
Credit Key	Key used by the **Credit** command to generate the temporary credit key.

CTC	Card Transaction Counter. Three-byte counter that the card increments each time a payment session is executed that is, the **Select Purse & Key** command is executed. MPCOS-EMV uses the Card Transaction Counter when it computes the temporary transaction key.
Data File	A file containing data which is not interpreted by the operating system. Data files can be transparent files or structured files (linear fixed files, linear variable files and cyclic files).
Data Unit	Data unit is either one byte or one word (four bytes), depending on the value of T10 in the ATR (see Appendix A, "*MPCOS-EMV Default Answer To Reset*").
Decryption	The transformation of encrypted code into plain data.
Dedicated File (DF)	Files that hold groups of elementary files.
DPA	Differential Power Analysis. An attack against smart card security whereby secret keys are derived when the card performs commands that require secret keys.
Elementary File (EF)	Part of the MPCOS-EMV file hierarchy; elementary files hold data (cf ISO 7816-4).
Encryption	The transformation of plain data into ciphered data that cannot be read without the cryptographic key used to encrypt the data.
FCI	File Control Information. This information is returned after a file is selected (an Answer To Select) and includes the file’s name, type (DF or EF), level (global or local) and short file identifier.
File Descriptor	Element generated at file creation and used by the MPCOS-EMV operating system to manage files.
IADF	Internal Application Data File. This file determines the contents of the FCI.
Indicators	Internal flags recording the following: - Whether or not the temporary transaction key has been established. - Whether or not a debit transaction has been executed in the current transaction session. - Whether or not the command data is valid.

Key	Cryptographic keys that MPCOS-EMV uses for encrypting and decrypting data along with the 3DES algorithm (using 16-byte keys). Keys come from two sources; they are either retrieved from key files, which are initialized during card personalization or they are generated by MPCOS-EMV. Different types of keys exist and have different purposes for example, payment keys and administration keys.
Key File	Key files store the cryptographic keys used in all MPCOS-EMV cryptographic functions. Each key file can store up to four 3DES keys (16 bytes).
Log Key	Type of payment key that can be used for the computation of temporary payment keys and for the secure messaging process. The log key cannot be used to compute temporary administration keys.
Master File (MF)	This is the root of the MPCOS-EMV file structure. The MF is also a Dedicated File (DF) (cf ISO 7816-4).
Mother System Key	The key from which the system key for each card in a batch of MPCOS-EMV cards is derived. It is stored in the batch card that corresponds to the batch of MPCOS-EMV cards.
MPCOS	Multi-application Payment Chip Operating System is an operating system adapted to multi-purpose and payment applications.
MRN	Maximum Replay Number. A security mechanism that puts an upper limit to the usage of temporary keys in a secure session. Used against Differential Power Analysis (DPA) attacks.
ONC	Off-Nominal Counter. A security mechanism that restricts the number of times a permanent key is used. Used against Differential Power Analysis (DPA) attacks.
OTP Memory	One-time programmable area in the card's memory, that is, it is irreversible.
Payment Command Cryptograms	Payment Command Cryptograms (PCCs) are generated by MPCOS-EMV cards and terminals during payment transactions and can be used to verify the integrity of transmissions.
Payment Key	MPCOS-EMV cryptographic key used for payment commands such as temporary certification key computation.
Payment Session	Session during which payment commands is performed.

Purse	A conceptual entity with which financial transactions are performed. See document entitled *Electronic Purse Architecture for MPCOS-EMV* for more details.
Purse File	File, which contains purse data, such as its balance, credit and debit security attributes and the pointer to the cryptographic key, used to generate credit certificates.
Secret Code Descriptor	Each secret code has a descriptor which defines the following information: - Ratification status - Whether or not it has to be entered encrypted - Whether or not it has been initialized
Secure Messaging	Process used by MPCOS-EMV to protect administration command transmissions between cards and terminals. This can be done either by encrypting the data, which is sent to the card, or by sending three bytes of a cryptographic checksum with the command.
Sensitive File	Sensitive files include all dedicated files, key files, secret code files, purse files, transaction manager files and internal application data files. More generally, sensitive files contain data, which is interpreted by the operating system, or they can be involved in security mechanisms.
Signature Key	Key used to generate a signature.
System Key	Also referred to as "Transport Key", this key is derived from the mother system key in the batch card. It is used to compute temporary keys for secure messaging when creating DFs. System keys are either diversified from the mother system key or a constant TEST KEY.
Temporary Key	A cryptographic key that MPCOS-EMV cards use during sessions involving encryption and decryption operations to generate certificates, cryptographic checksums or secure messaging.
TLV	Tag, Length, Value. A data format, where tag is a number, length is length of data and value is the data itself.

3DES	Triple DES Symmetric and secret key encryption/ decryption algorithm based on 16-byte keys. It produces the most secure cryptographic mode in MPCOS-EMV cards.
TTC	Terminal Transaction Counter (TTC) is a three-byte value that MPCOS-EMV cards use to keep track of transactions executed with each terminal. The terminal must increment the TTC each time it establishes a new payment transaction session.

Index