SchlumbergerSema

Cryptoflex™ Cards Programmer's Guide

Cyberflex® Access™
Software Development Kit 4.4

Trademarks

Document Edition	**Date**
C300474_rev2	February 2003

Your feedback about this manual is welcome! Comments, questions, and suggestions about any part of the Cyberflex Access documentation library can be posted to the Docs & Samples Conference of the User Discussion Forums: *www.flexforum.com/cgi-bin/dcforum/dcboard.cgi*.

Preface

Preface

THIS GUIDE, the *Cryptoflex Cards Programmer's Guide*, describes the Cryptoflex smart cards from SchlumbergerSema, which are designed to operate with information security applications.

Who Should Read This Guide

The *Cryptoflex Cards Programmer's Guide* is for programmers who are developing smart card applications for Cryptoflex cards. This guide describes the card, its file system, access conditions, commands, and conventions.

This guide is written for experienced programmers who are familiar with smart cards and cryptography, and have access to the ISO 7816 standards that are the basis for smart card technology.

What Is in This Guide

This guide documents characteristics of the Cryptoflex 16K card (version 1), Cryptoflex 32K cards (version 0 and version 1), and Cryptoflex e-gate™ 32K card, as well as the commands available to use in developing smart card applications for Cryptoflex cards. The version 1 Cryptoflex 32K card is the same as the Cryptoflex e-gate™ 32K card except that the Cryptoflex e-gate™ 32K card supports USB communication mode.

This manual does not document the Cryptoflex 4K smart card or the Cryptoflex 8K family of cards; documentation for these cards is available from the Cryptoflex Support website (*http://www.cryptoflex.com/Support/support.html*).

Most information contained in this manual applies to all Cryptoflex cards; wherever information is different for a specific Cryptoflex card type, this exception is highlighted in the following manner:

This guide contains information to help you develop applications that take advantage of SchlumbergerSema smart card security capabilities.

- **Section 1, "Card File System"** describes basic concepts of the card file system, such as types of card files and file IDs.
- **Section 2, "Key Files"** describes the structure and content of the files you use to store CHV keys, external keys, internal keys, and RSA key pairs.
- **Section 3, "Access Rights and Security"** describes the access rights that protect commands against unauthorized use. This section also contains a short overview of cryptographic concepts.
- **Section 4, "Cryptoflex Card Commands"** describes the card's operating system commands.
- **Section 5, "Writing a Card Application"** guides you step-by-step through creating a simple card application and includes some background information about the card.
- **"The Communication Interface"** describes the command and response format for data that passes between the host application and card (through the card reader).
- **"Status Words"** summarizes the meaning of the commonly used card status words, which notify you of a command's success or failure.

- **"Technical Details and Procedures"** summarizes the card's physical and electrical card specifications. This section also describes the procedure for changing the reader-to-card data transmission speed.

Document Conventions

This guide uses the following conventions:

Italic	Emphasizes text and distinguishes a new term or file name.
`monospace font`	Identifies command names, code examples, terminal output, and field entries.
🔑	Identifies the access condition (if any) you must satisfy in order to execute the specified command in a particular context.
⚠️	Marks critical information that can help you avoid potential problems.

NOTE *All of the table specifications throughout this document display values in hexadecimal format unless otherwise noted.*

Acronyms Used

The following acronyms are used in this guide. For more information about these and other terms, see the glossary.

0000 — reserved ID for CHV1 key file ($\mathrm{EF}_{\mathrm{CHV1}}$)

0001 — reserved ID for internal key file ($\mathrm{EF}_{\mathrm{Key\ Int}}$)

0002 — reserved ID for serial number file ($\mathrm{EF}_{\mathrm{ICC\ SN}}$)

0011 — reserved ID for external key file ($\mathrm{EF}_{\mathrm{Key\ Ext}}$)

0012 — reserved ID for private key file ($\mathrm{EF}_{\mathrm{RSA\ PRI}}$)

0100 — reserved ID for CHV2 key file ($\mathrm{EF}_{\mathrm{CHV2}}$)

1012 — reserved ID for public key file ($\mathrm{EF}_{\mathrm{RSA\ PUB}}$)

2F01 — reserved ID for ATR file ($\mathrm{EF}_{\mathrm{ATR}}$)

3DES — triple DES

3F00 — reserved ID for the card's master file (root directory)

3FFF — reserved file ID (RFU)

AAK — application authorization key (called the transport key in earlier cards)

AC — access condition

ADF — application definition file (EMV application)

AEF — application elementary file (EMV application)

AID — application identifier

ALW — Always (access condition)

APDU — application Protocol Data Unit

ATR — answer to reset

AUT — authenticate (access condition)

CA — certificate authority

CBC — cipher block chaining mode of DES encryption/decryption

CHV — cardholder verification

CLA — class (First byte of an APDU command)

COVE — Cryptographic Object Viewer and Editor (Cyberflex Access SDK application)

CR — characters remaining

CY — cyclic elementary file

DES — Data Encryption Standard

DF — dedicated file (card directory)

EBC — electronic book code mode of DES encryption/decryption

EEPROM — electrically erasable programmable read-only memory

EF — elementary file

EOF — end of file

FCI — file control information

FFFF — reserved file ID (RFU)

FID — file identifier

FS — file size

INS — instruction (second byte of an APDU)

IV — initialization vector

Lc — third parameter of an APDU command that includes input data

Le — third parameter of an APDU command that produces response data

LOFB — length of fill block

LOUD — length of useful data

LSB — least significant byte

LSN — least significant nibble

MF — master file

mod — modulus

MSB — most significant bit

MSN — most significant nibble

NEV — Never (access condition)

NR — number of records

P1, P2 — parameters 1 and 2 of an APDU command (third and fourth bytes)

PIN — personal identification number

PKCS #11 — Public Key Cryptography System specification #11

PKI — Public Key Infrastructure

PRO — Protected command mode (Cryptoflex access condition)

PSE — Payment System Environment (EMV structured environment)

RFU — reserved for future use

RL — record length

RNG — random number generator

SCOS — SchlumbergerSema Card Operating System

SFI — short file identifier

SHA-1 — Secure Hash Algorithm (SHA-1 is a technical revision of SHA)

XOR — exclusive OR

Other Sources of Information

For more information related to the Cyberflex Access Software Development Kit, see these manuals:

- *Cyberflex Access Software Development Kit User's Guide* — A developer's introduction to the SchlumbergerSema information security smart cards smart cards—Cyberflex Access and Cryptoflex—and the Cyberflex Access Software Development Kit that supports them.
- *Guide to SchlumbergerSema Smart Card Middleware* — Describes the interoperability layers that support higher-level functions in card programs for all the SchlumbergerSema information security smart cards.

If you select to install the documentation, you will find the manuals in .pdf format in the following directory on your system: *C:\Program Files\Schlumberger\Smart Cards and Terminals\Cyberflex Access Kits\v4\Documentation*. All manuals are also available on the distribution CD-ROM.

Current and previous manuals are linked from the Cyberflex Access Support website (*http://www.cyberflex.com/Support/support.html*) and the Cryptoflex Support website (*http://www.cryptoflex.com/Support/support.html*).

The following websites have information about smart cards and cryptography:

Website / Email Address	Contents
www.cryptoflex.com	The SchlumbergerSema Cryptoflex home page
www.cryptoflex.com/Support/support.html	The SchlumbergerSema Cryptoflex support page: links to a user's discussion forum, FAQ, technical support, the most current documentation, and an email link (*cryptoflex@slb.com*)
www1.slb.com/smartcards	Information about SchlumbergerSema smart card products
www.reflexreaders.com	Information about the SchlumbergerSema Reflex series smart card readers

Website / Email Address	**Contents**
www.reflexreaders.com/ Support/support.html	The SchlumbergerSema card reader support page: links to drivers, technical help, and documentation
www.scmegastore.com	The SchlumbergerSema smart card marketplace
www.pcscworkgroup.com/	Introductory page for the PC/SC Workgroup, which develops the specifications for the Personal Computer/Smart Card (PC/SC) standard
www.microsoft.com/ smartcard	Microsoft site for information about developing CryptoAPI-compliant smart card programs
www.rsasecurity.com	Introductory page for RSA Security, which sets standards for Secure Electronic Transactions (SET), Data Encryption Standards (DES), and Public Key Cryptography Standards (PKCS).
www.emvco.com/	Front page for the EMVCo, which develops EMV Integrated Circuit Card Specifications
www.iso.ch	Front page for the International Standardization Organization (ISO)

Card Features

Shorthand Names for Cryptoflex Cards

This guide sometimes refers to specific Cryptoflex card versions. This guide uses the following shorthand card names:

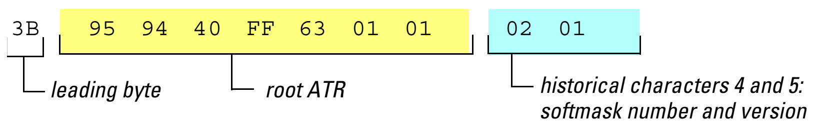
- 16K+SS V1 — Cryptoflex 16K card with Standard Softmask V1
- 32K+SS V1 — Cryptoflex 32K card with Standard Softmask V1
- 32K+e-gate — Cryptoflex e-gate 32K card

Default ATRs

You can use the default answer to reset (ATR) to determine which card version you have. The table that follows shows example default ATRs for recent versions of Cryptoflex cards. Note that the ATR's final two bytes are specific to the card's softmask number and version. The softmask bytes are subject to change.

Card Version	Default ATR (Examples)
16K+SS V1	`3B 95 15 40 FF 63 01 01 02 01`
32K+SS V1	`3B 95 18 40 FF 64 02 01 00 00`
32K+e-gate	`3B 95 18 40 FF 62 01 01 00 00`

Example ATR Components



NOTE *For more information about ATR components, see page 15.*

Cryptoflex 32K Card Support

Support for the Cryptoflex 32K card (32K+SS V1) was added in the Cyberflex Access Software Development Kit 4.2 release.

These new features were introduced with the Cryptoflex 32K card:

- 32KB of available EEPROM memory (previous standard for Cryptoflex cards was 16KB)
- Additional available RSA public key formats that offer more flexibility for a range of customer environments. Three public key formats are defined, and all three formats may co-exist within the same public key file:
  - All public components are stored in the public key file. This format ensures backward compatability with existing applications build on earlier versions of Cryptoflex.
  - Only the public modulus and exponent are stored in the public key file.
  - Only the public exponent is stored in the public key file.

  See “RSA Key Files,” on page 34 for additional information about the new public key formats. See `Generate RSA Keys` command (described on page 127) for information about new input parameters that allow the user to generate RSA keys in the three available key formats.
- New internal RSA signature verification to ensure the coherence of the signature in order to avoid a potential attack on the private RSA key. See *`RSA Signature (Internal Auth)`* command, described on page 164 for information about this check.

Cryptoflex 32K e-gate Card Support

Support for the Cryptoflex e-gate 32K card (32K+e-gate) was added in the Cyberflex Access Software Development Kit 4.2 release.

These new features were introduced with the Cryptoflex e-gate 32K card:

- Support for the USB V1.1 communication interface, as well as the standard ISO 7816-3 interface. In USB mode, the cards essentially plug directly into the PC using a simple passive connector, while the e-gate middleware simulates the presence of a PC/SC reader. In conventional ISO mode, the Cryptoflex e-gate cards behave like any other smart card and can be read using anyISO-compliant terminal or reader.
- 32KB of available EEPROM memory on the Cryptoflex e-gate 32K card.
- Additional available RSA public key formats that offer more flexibility for a range of customer environments. Three public key formats are defined, and all three formats may co-exist within the same public key file:
  - All public components are stored in the public key file. This format ensures backward compatability with existing applications build on earlier versions of Cryptoflex.
  - Only the public modulus and exponent are stored in the public key file.
  - Only the public exponent is stored in the public key file.

  See “RSA Key Files,” on page 34 for additional information about the new public key formats. See `Generate RSA Keys` command (described on page 127) for information about new input parameters that allow the user to generate RSA keys in the three available key formats.

  New internal RSA signature verification to ensure the coherence of the signature in order to avoid a potential attack on the private RSA key. See *`RSA Signature (Internal Auth)`* command, described on page 164 for information about this check.

Characteristics of Cryptoflex Cards

Cryptoflex is the ideal token for public-key infrastructures. Even if the user needs 512-, 768-, 1024-, or 2048-bit keys and several certificates for a variety of applications, they can all be stored on a single smart card—one secure place for secret information. Cryptoflex implements the most powerful security industry functions right on the card to enable single sign-on network access

General Characteristics

Cryptoflex cards have these general characteristics:

- *Communication protocol:* ISO T=0

32K+e-gate The Cryptoflex e-gate 32K card (32K+e-gate) card also supports USB mode.

- *Data transmission baud rate at 3.57 MHz:* 9600 bit/sec by default, configurable with Protocol Parameter Selection to the following maximum baud rates, as described on page 242.
  - *16K+SS V1 cards:* As high as 153,600 bit/sec, with a recommended transmission speed of 55,800 bit/sec
- *Nonvolatile memory:*
  - *16K+SS V1 cards*:
    14,400 bytes of EEPROM available for file architecture
  - *32K+SS V1 cards*:
    32,552 bytes of EEPROM available for file architecture
  - *32K+e-gate cards*:
    32,552 bytes of EEPROM available for file architecture

Cryptographic Features

Cryptoflex cards support a number of cryptographic features, including:

- On-card generation of DES keys (DES or double-length 3DES) and RSA keys (512-bit, 768-bit, 1024-bit, and 2048-bit keys)
- RSA signature calculation (key length 512, 768, 1024, and 2048 bits)
- Enciphering and deciphering data with DES or 3DES keys in cipher block chaining (CBC) mode
- External authentication (host-to-card) with DES or 3DES keys
- Internal authentication (card-to-host) with DES or 3DES keys in electronic book code (EBC) mode or with RSA digital signatures
- SHA-1 hashing
- The ability to restrict access to commands in a specified file set by requiring verification of a particular key
- A protected command mode for highly sensitive data, which requires protected commands to be signed with a DES or 3DES digital signature

Key and Challenge Lengths Supported

Cryptoflex cards support these key and challenge lengths:

- An 8-byte DES challenge
- DES and 3DES key operations that use one or two 8-byte keys
- For protected command mode, useful data with a maximum length of 240 bytes
- RSA keys that are 512, 764, 1024, and 2048 bits long

NOTE *Certain cryptographic card features may be restricted in some locations as required by export and import laws.*

1

Card File System

Overview

This section contains basic information about Cryptoflex card files:

- The file system on Cryptoflex cards
- A brief overview the stages of personalizing a card
- Card file types and characteristics
- Reserved file IDs
- File size calculations
- Special card files that are used to hold:
  - Card serial numbers
  - Answer to reset (ATR) values

NOTE *For information about the key files, see "Key Files" starting on page 17.*

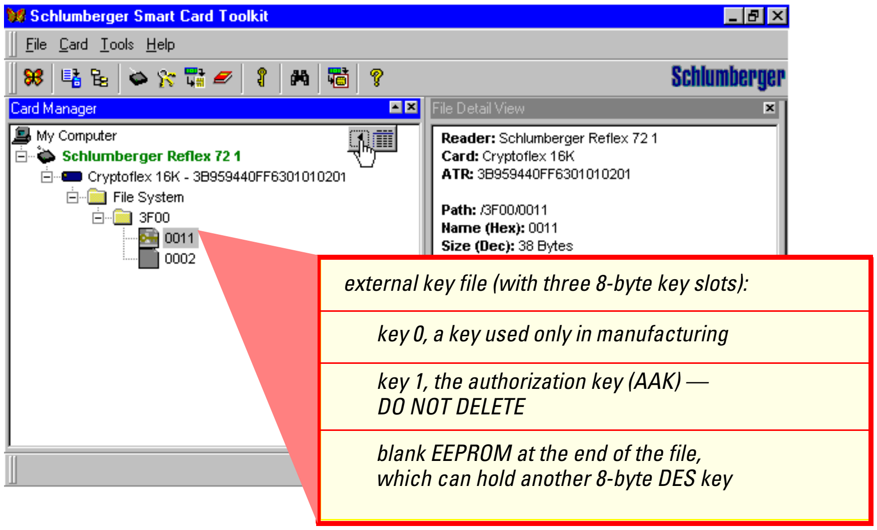
The File System on a New Card

When you receive a new Cryptoflex test card from SchlumbergerSema, it has a rudimentary file system. The following illustration shows the default file system of a new test card as it appears in the Smart Card Toolkit application of the Cyberflex Access Software Development Kit.

The card has a master file (MF), which acts as the root directory and contains these files by default:

- **0002** — Serial number file (page 13)
- **0011** — The default external key file (EF$_{\text{Key Ext}}$) at the root level. The EF$_{\text{Key Ext}}$ has three key slots:

  1. A key used only during manufacturing.
  2. The 8-byte DES key application authorization key (AAK), (also called the transport key).
  3. A slot for adding another 8-byte DES key.



Reader: Schlumberger Reflex 72 1
Card: Cryptoflex 16K
ATR: 3B959440FF6301010201

Path: /3F00/0011
Name (Hex): 0011
Size (Dec): 38 Bytes

external key file (with three 8-byte key slots):

key 0, a key used only in manufacturing

key 1, the authorization key (AAK) — DO NOT DELETE

blank EEPROM at the end of the file, which can hold another 8-byte DES key

NOTE *On Cryptoflex test cards, external and internal keys are numbered consecutively. These keys derive their number implicitly from their position in the key file.*

The card grants you access when you verify the AAK—enabling you to add other directories (DFs) and elementary files (EFs) and change the contents of files. *(For more information about the external key file, see page 24.)*

- *Do not delete the original AAK (transport key) or the external key file that holds it. The card stops working if you delete or disable the AAK.*
- *Do not delete the serial number file, or you will be unable to communicate with the card.*

Personalizing a Smart Card

A test smart card goes through default pre-personalization at the factory, which gives it just enough of a key file structure to keep the card secure in transit and enable you to unlock the card when you receive it.

Before you can begin to work with a smart card, you must perform custom personalization. Custom personalization involves building a file system on the card and adding the key files needed to support your card program's operations. If you use the Cyberflex Access SDK, you can use the Cryptographic Object Viewer and Editor (COVE) application for this purpose. *(For more information about COVE, see the Cyberflex Access Software Development Kit User's Guide.)*

Once the file structure is in place (along with any keys used to set up the file structure), you can perform final personalization. Final personalization involves adding the card-specific data you need for your programs, such as cardholder identification, cardholder and card administrator PINs, and cryptographic key data.

File Types

Card operating system files are divided into three types:

- The master file (MF)
- Dedicated files or directories (DFs)
- Elementary files (EFs)

NOTE *File IDs Reserved for Special Files — The table on page 8 shows the file IDs reserved for particular types of Cryptoflex cards.*

The Master File

The master file (MF) is a special dedicated file that is the root of the card's file system. (Container or directory files on the card are called dedicated files or DFs.) The MF can contain dedicated files and elementary files (data files). The MF's reserved file identifier is 3F00. The card selects the MF by default when the card's microprocessor is powered up—when the card is removed and inserted into a reader or when you perform a card reset.

Like any other dedicated file, the MF has an input parameter string, but no data body.

NOTE *For information about the master file's default AC values and AC key numbers, see page 68.*

Dedicated Files

A dedicated file (DF) is a subdirectory in the file hierarchy. A DF can contain elementary files and other directory files.

If a CHV key file, internal key file, or external key file exists within the selected DF, it is the *relevant* key file—the one you use to gain access to that directory, its contents, and the contents of child files. If no key file is present in a particular DF, the relevant key file is the first key file of the appropriate type that you find as you travel up the file hierarchy. In other words, a key file's domain extends throughout its parent directory and extends downward until another key file takes over.

NOTE *Relevance applies only to CHV, external, and internal keys—not to RSA private and public keys.*

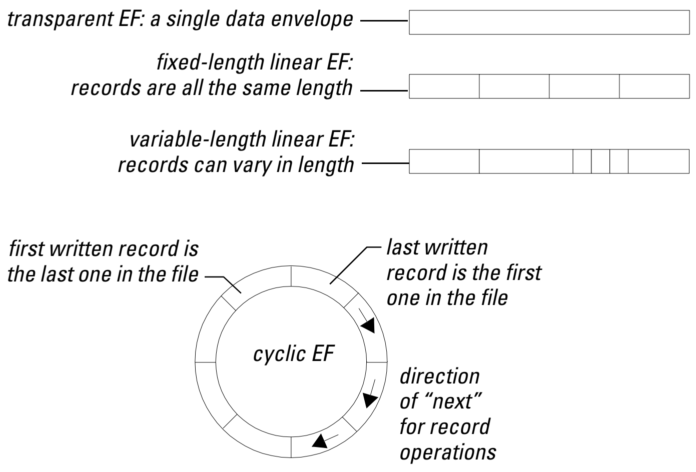
A single DF on a smart card can hold one of each of the following key files:

- External authentication key
- Internal authentication key
- CHV1 (user PIN)
- CHV2 (administrator PIN)
- RSA private key
- RSA public key

If you create a DF, base its size on the number and size of the files it will hold. If you do not allow sufficient room for all the files needed, you must delete the DF and create it again. You cannot alter the size of an existing DF. Remember to include the EEPROM memory required for the file input parameters. *(For more information about calculating file sizes, see page 9.)*

Elementary Files

Elementary files (EFs) can contain program data, such as names, dates, and serial numbers. Cards have four types of elementary files—transparent, fixed-length linear, variable-length linear, and cyclic (shown in the following illustration).


transparent EF: a single data envelope
fixed-length linear EF: records are all the same length
variable-length linear EF: records can vary in length
first written record is the last one in the file
last written record is the first one in the file
cyclic EF
direction of "next" for record operations

Transparent Elementary Files

Transparent elementary files contain a sequence of bytes. These files are called transparent because they have no interior structure—they contain a single data envelope. Transparent elementary files are useful for storing objects such as keys.

Linear Elementary Files

Linear elementary files contain subdivisions, or records, which can be fixed or variable in length. Each record in a file is identified by a number, assigned to the record sequentially as it is created. You can use the record number to read from or write to a specific record in a linear file.

In a *fixed-length linear elementary file*, all the records contain an equal number of bytes. You can create a maximum of 255 records in a single fixed-length linear file, and you can set the record length to any number of bytes between 1 and 255. Records are created consecutively from the beginning of the file. Record files (linear or cyclic EFs) have a record pointer that points to the currently selected record.

In a *variable-length linear elementary file*, you have the option to specify different lengths for the records in a particular file. You can create a maximum of 255 records in a variable-length linear file, with the records varying from 1 to 255 bytes in length. Variable-length linear files conserve valuable space in memory when you have various lengths of data to store in the same file. On the other hand, variable-length linear files require more seek time for read and write operations and require slightly more space for record headers.

Cyclic Elementary Files

Think of a cyclic EF as a ring of records, with all the records in the file equal in length. Each new write operation modifies the next record in the ring. If the circle of records is full, new data overwrites the oldest record data. The record pointer is set to the most recently written record, which becomes the new first record in the file.

Cyclic EFs are especially useful for storing “last ten” operations, such as dates or transaction amounts. You can create a maximum of 255 records in a single cyclic EF, and can set the record length to any number of bytes from 1 to 255. (Note that records that are less than 3 bytes long have limited use. For example, you cannot execute an `Increase` or `Decrease` command on a cyclic EF record unless it is at least 3 bytes long.)

Reserved File IDs

The following table shows the reserved file IDs for special types of card files.

If you create a file listed in the table, use its reserved file ID. The card's operating system refers to a file internally by its file ID (FID). Although you can create files with any hexadecimal identifier, using a reserved file ID inappropriately can cause the card to behave in an unexpected manner.

File ID	File Name	Type of Data	File Type	Level
0000	EF_{CHV1}	Identification PIN *(i.e. for card holder)*	trans. EF	*relevant*
0001	$EF_{Key\ Int}$	Internal keys	trans. EF	*relevant*
0002	$EF_{ICC\ SN}$	Card serial number	trans. EF	*under 3F00*
0011	$EF_{Key\ Ext}$	External keys	trans. EF	*relevant*
0012	$EF_{RSA\ PRI}$	Private keys	trans. EF	*any*
0100	EF_{CHV2}	Identification PIN *(i.e. for card administrator)*	trans. EF	*relevant*
1012	$EF_{RSA\ PUB}$	Public keys	trans. EF	*any*
2F01	EF_{ATR}	ATR	trans. EF	*under 3F00*
3F00	Master file	Card contents	root DF	*root*
3FFF	RFU	—	—	—
FFFF	RFU	—	—	—
FF*xx*	RFU *(FID begins w/ FF)*	—	—	—
*xx*FF	RFU *(FID ends w/ FF)*	—	—	—

Calculating File Sizes

It is important to carefully plan the size of files you want to create. You cannot change the size of a file once it is created. If you need to change a file's size, you must delete the file and replace it. Consider the following issues when you calculate file size:

- All files contain an input parameter string that defines the file attributes.

  (The size of the input parameters for various file types are shown in the following tables, along with the hexadecimal value you use to specify the file type when you create the file.)
- Elementary files (EFs) also have a body (or data envelope) to hold data.
- Dedicated files (DFs) have a container that holds all the DF's elementary files and subsidiary dedicated files. When you calculate the size of a DF, the total must be large enough to accommodate the EF file bodies and input parameters, as well as subsidiary DF containers and input parameters.

NOTE *A DF or EF occupies the amount of space allocated during file creation, even if the DF or EF is empty.*

When you plan the files for a card program, make sure the card has enough EEPROM memory available. If the card already contains some files, you may need to calculate the amount of memory they occupy.

Retrieving File Information — You can select a directory (by calling a `Select` command), then call a `Dir Next` command, and retrieve information for the files in the selected directory one by one.

To calculate the amount of memory the file occupies, add the input parameter string length (from the table that follows) to the file body length. The card returns the file body length in response to a `Select` command.

Cryptoflex Card File Input Parameter Lengths and File Type Values

File Type	Hexadecimal File Type Byte Value	Length of Input Parameters
Master file	`38`	24 B
Dedicated file	`38`	24 B
Transparent EF	`01`	16 B
Fixed-length linear EF	`02`	16 B

File Type	Hexadecimal File Type Byte Value	Length of Input Parameters
Variable-length linear EF	`04`	16 B
Cyclic EF	`06`	16 B

The next table contains formulas for calculating the size of different types of files. Use the second column to calculate the value for the record length you specify when you create the file. Use the third column to calculate the actual amount of EEPROM memory to allocate for the file. Be sure to include space for the input parameters of the file and its records (if any).

Cryptoflex Card File Size Calculations

File Type	Logical File Size	Actual File Size Allocated, Including Input Parameters
Transparent EF	*RL*	*(RL)Mod4* + 16 B
Fixed-length linear EF	*NR* x *RL*	*(NR* x *RL)Mod4* + 16 B
Variable-length linear EF	ΣRL	Σ([*RL*]*Mod4*+4) + 16 B This size must not exceed the size specified at file creation. Use the `Create Record` command to create records. Each record has an actual length of *(RL)Mod4* + 4.
Cyclic EF	*NR* x *RL*	*NR* x ([*RL*]*Mod4* +4) + 16 B Each record has an actual length of *(RL)Mod4* + 4.
DF	Sum of space allocated for subsidiary files	Total + 24 B

where:

RL	=	Record length
NR	=	Number of records (For a transparent EF, *NR* = 1.)
Mod4	=	Indication to increase the value, if necessary, to the nearest multiple of 4. For example, (3)*Mod4* = 4 and (4)*Mod4* = 4.

NOTE *Each record in a variable-length linear EF or cyclic EF has a 4-byte header.*

Standard Key File Sizes

The following table shows standard sizes for a range of commonly used files. All the file IDs shown here are reserved. Except for the external key file, the examples are based on one key per file.

Examples of Standard Key File Sizes

The total file size = (useful length)*Mod4* + 16-bytes of input parameters. The following table shows the EEPROM required for keys of various types.

File	File ID	Useful Length		
CHV key (identification key)	0000	23 B	=	23 B
Internal key (1 DES)	0001	1 + (N x 12) + 1 B	=	14 B
External key (3 8-byte keys)	0011	1 + (N x 12) + 1 B	=	38 B
Private key (1 RSA 512-bit key)	0012	(N x163) + 3 B	=	166 B
Private key (1 RSA 768-bit key)	0012	(N x 243) + 3 B	=	246 B
Private key (1 RSA 1024-bit key)	0012	(N x 323) + 3 B	=	326 B
Private key (1 RSA 2048-bit key)	0012	(N x 643) + 3 B	=	646 B
Public key (1 RSA 512-bit key), all components present	1012	(N x 167) + 3 B	=	170 B
Public key (1 RSA 512-bit key), public modulus and exponent only[1]	1012	(N x 71) + 3 B	=	74 B
Public key (1 RSA 512-bit key), public exponent only[1]	1012	(N x 7) + 3 B	=	10 B
Public key (1 RSA 768-bit key), all components present	1012	(N x 247) + 3 B	=	250 B
Public key (1 RSA 768-bit key), public modulus and exponent only[1]	1012	(N x 103) + 3 B	=	106 B
Public key (1 RSA 768-bit key), public exponent only[1]	1012	(N x 7) + 3 B	=	10 B
Public key (1 RSA 1024-bit key), all components present	1012	(N x 327) + 3 B	=	330 B

File	File ID	Useful Length		
Public key (1 RSA1024-bit key), public modulus and exponent only[1]	1012	(N x 135) + 3 B	=	138 B
Public key (1 RSA 1024-bit key), public exponent only[1]	1012	(N x 7) + 3 B	=	10 B
Public key (1 RSA 2048-bit key), all components present	1012	(N x 647) + 3 B	=	650 B
Public key (1 RSA 2048-bit key), public modulus and exponent only[1]	1012	(N x 263) + 3 B	=	266 B
Public key (1 RSA 2048-bit key), public exponent only[1]	1012	(N x 7) + 3 B	=	10 B

1 Available for 32K+SS V1 and 32K+e-gate cards only.

where:

N	=	Number of keys stored
$Mod4$	=	Indication to increase the value, if necessary, to the nearest multiple of 4. For example, (3)$Mod4$ = 4 and (166)$Mod4$ = 168.

NOTE *For more information about key files, see the next section, beginning on page 17.*

Special Card Files

The next two topics describe special files found only on Cryptoflex cards, which are used for storing card serial number and answer to reset (ATR) data.

Serial Number File

On a Cryptoflex card the serial number is stored in the integrated circuit card serial number elementary file ($EF_{ICC\ SN}$). This is a transparent EF that has the reserved file ID 0002. The serial number uniquely identifies each card among the millions of smart cards that SchlumbergerSema manufactures. Each of the serial number's four bytes can have a value as high as 232. You can also specify a value to identify the card series (in byte 5). The following table shows the format of the serial file data. The file has 16 bytes of input parameters.

File Format

Byte	Description	Length
1 – 4	Serial number	4 B
5 – 6	Customer code (customer card series identifier)	1 B
7	Manufacturing site code	1 B
8	Usage code	1 B

Answer To Reset

If you insert a smart card into a reader or reset the card, the card sends a data string known as an Answer To Reset (ATR) to the host application. The ATR signals the reader that the power-up sequence was successful and establishes a communication pathway between the card and reader. The ATR contains a variety of information that SchlumbergerSema specifies during manufacture, such as card protocols, chip and soft mask identifiers, and version numbers.

Changing the ATR

To change the card's ATR value, select the ATR file (EF_{ATR}), call an `Update Binary` command, and construct the file as shown in this topic.

A SchlumbergerSema card always generates a leading ATR byte with a value of `3B`. If you create an alternate ATR, ignore this leading byte. Once you create a custom EF_{ATR} on the card, whenever the card is reset the card returns your custom string (with the value `3B` in front of it) instead of the original ATR.

The default EF_{ATR} has the reserved file ID 2F01 and conforms to the structure shown in the following table. The file has 16 bytes of input parameters.

File Format

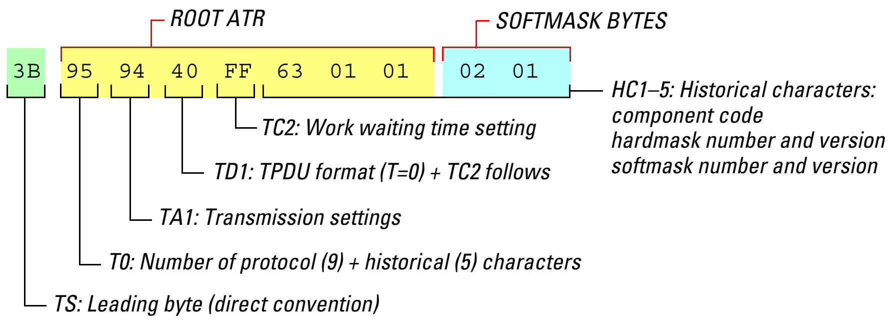
Byte	Description	Length
1	Length of ATR string (x), without the leading byte, `3B`h.	1 B
2 – (x + 1)	ATR string data (without the leading byte, `3B`h).	x B

The card always uses the ATR from the ATR file (EF_{ATR}) located under the master file (MF). The card does not search for an ATR file in other directories. If you delete the ATR file from the MF, the card uses the default ATR in EEPROM. If the data inside the EF-ATR is empty or inconsistent, the card becomes mute (does not return an ATR and cannot communicate with the reader).

If you change the ATR value, make sure the new value is valid. If the ATR data stream is not in the approved format or uses unsupported values, the reader cannot interpret the ATR, and the card is mute.

ATR Content: Example

Example ATR Components


ROOT ATR
SOFTMASK BYTES
3B
95 94 40 FF 63 01 01
02 01
HC1–5: Historical characters: component code hardmask number and version softmask number and version
TC2: Work waiting time setting
TD1: TPDU format (T=0) + TC2 follows
TA1: Transmission settings
T0: Number of protocol (9) + historical (5) characters
TS: Leading byte (direct convention)

- Length of ATR = `09`h.
- ATR leading byte = `3B`h.
- ATR string = `95 94 40 FF 63 01 01 02 01`h.

The ATR byte values convey the following information:

ATR Value	Meaning
`3B`h	**TS**: Direct convention
`95`h	**T0**: • `9`h = TA1 and TD1 protocol characters follow • `5`h = 5 historical characters transmitted
`94`h	**TA1**: Protocol character that indicates Fi_{max} = 512 and Di_{max} = 8 (if f_{clock} = 3.57 Mhz) *(For information about the PPS command values the card supports, see page 242.)*
`40`h	**TD1**: Protocol character for TPDU format (T = 0). Also indicates TC2 follows: No more interface characters are included.
`FF`h	**TC2**: Codes value of WI to define work waiting time (960*D*WI) = 27s
`63`h	**HC1** (historical character 1): Component code = ST19KF16
`01`h	**HC2**: Hardmask number. `01`h = First hardmask of this component
`01`h	**HC3**: Hardmask version. `01`h = First hardmask version

ATR Value	Meaning
`02h`	**HC4**: Softmask number. `02h` = Second softmask
`01h`	**HC5**: Softmask version. `01h` = First softmask version

ATR File Access Condition Settings

The following table shows recommended settings for an EF_{ATR}. *(For more information about access conditions, see "Access Rights and Security" starting on page 61.)*

Nibble	Commands Affected	Setting	Hex Value
1	`Read Binary`	NEV	`F`
2	`Update Binary`	AUT	`4`
3	`Read Binary Enciphered`	NEV	`F`
4	`Update Binary Enciphered`	AUT	`4`
5	`Rehabilitate`	AUT	`4`
6	`Invalidate`	AUT	`4`

2

Key Files

This section describes the key files the Cryptoflex card supports:

- Cardholder verification files (EF$_{CHV1}$ and EF$_{CHV2}$)   page 21
- External key files (EF$_{Key\ Ext}$)   page 24
- Internal key files (EF$_{Key\ Int}$)   page 31
- RSA private and public key files (EF$_{RSA-PRI}$ and EF$_{RSA-PUB}$)   page 34

File descriptions cover:

- File type and reserved file ID
- Uses for the keys
- Key domains
- File format
- Recommendations for setting access rights

Key File Summary

Identification keys and cryptographic keys are stored in specific types of files, as summarized below.

Types of Key Files

Key File	File ID	Description
EF_{CHV1}	0000	Contains CHV1 data: The PIN, unblocking PIN, pre-set value for the number of verification attempts allowed, a counter for the number of remaining verification attempts, the maximum number of unblocking mechanisms, and a counter for the number of remaining unblocking mechanisms.
EF_{CHV2}	0100	Contains CHV2 data: The PIN, unblocking PIN, pre-set value for the number of verification attempts allowed, a counter for the number of remaining verification attempts, the maximum number of unblocking mechanisms, and a counter for the number of remaining unblocking mechanisms.
$EF_{Key\ Ext}$	0011	External key file – Contains one or more DES or 3DES keys used for external authentication, key verification, PRO mode computation, and ciphering for `Update Binary Enciphered` and `Read Binary Enciphered` commands.
$EF_{Key\ Int}$	0001	Internal key file – Contains one or more DES or 3DES keys used for internal authentication and operations such as cipher block chaining (CBC).
$EF_{RSA\ PRI}$	0012	Private key file – Contains the private key algorithm for one or more RSA key pairs (512-bit, 768-bit, 1024-bit, or 2048-bit).
$EF_{RSA\ PUB}$	1012	Public key file – Contains the public key modulus for one or more RSA key pairs (512-bit, 768-bit, 1024-bit, or 2048-bit).

Keys Included by Default on Cryptoflex Cards

The following table shows keys included by default in the $EF_{Key\ Ext}$ of a new Cryptoflex test card. The import file notations refer to files in the Cyberflex Access Software Development Kit Key Manager database, which is added to your host system when you install the software.

Key #	Description / Use	SDK Import File	Key Blob Contents
0	Manufacturing key used only at the factory	*(none)*	—
1	Application authorization key (AAK), also called the transport key	*transport.key*	*example:* `2C15E526E93E8A19`
2	Empty key slot	*(none)*	—

Cyberflex Access SDK Smart Card Toolkit — *To gain access to a new Cryptoflex card in the Smart Card Toolkit, you must verify AUT1 (the application authorization key, AAK, also known as the transport key). To verify AUT1 by selecting a key in the Select Key dialog box, select the key labelled for your test card, for example, Cryptoflex 16K Transport Key.*

If you attempt to verify AUT0 (the manufacturing key), the operation fails. If verification fails three times, you irretrievably lock yourself out of the card. If you have used several of the available verification attempts, it is best to switch to another card.

Occasionally, the pre-seeded AAK (transport key) value for a card series does not match the AAK applied to the card during manufacturing pre-personalization. In this case, even if you select the correctly labelled key from the Select Key dialog box, verification fails. If this mismatch occurs, you must manually specify the transport key. Typically, the transport key is included in printed information supplied with the test card. The value is usually provided in hex format. (If the transport key is provided in ASCII format, you must convert the ASCII value to hex.) Enter the hex value in the Verify Key dialog box. After you have confirmed that the transport key is correct, create a new key using the correct hex value, add the key to the Key Manager database, and select the newly created key when you need to access the card again.

Key Type Algorithm IDs

DES and 3DES key types are identified by their algorithm ID, as shown in the following list.

- `00h` = Single-length DES key
- `02h` = Double-length 3DES key

Key File Domains

You can create different security models by varying the number and location of the key files. For example, suppose you use a single external key file located under the master file. The keys in this file have the entire card as their domain.

As another example, let’s say you create multiple external key files and place them in a number of directories, which each contain files for a different card program. In this case, each key file’s domain extends to the files in its local directory and to files lower in the file hierarchy. The key file’s domain continues to spread downward until an external key file occurs at a lower level. The key file is *relevant* to the files in its domain.

To continue the example, let’s say you call a `Verify Key` command. The card looks for the relevant external key file—by looking in the directory that contains the currently selected file. If no external key file exists in this location, the card moves up the file hierarchy until it finds the appropriate key file type or reaches the top of the file hierarchy (the master file).

If the card does not find any external key file, it returns an error. If the card does find a relevant key file, it looks for the specified key. If the file does not contain the key number specified in the command, the card returns an error. (The card does not look in any other key file.)

Each relevant key file you create has a corresponding access zone. Multiple access zones are useful in some circumstances, but can also slow down operations. If you move through multiple access zones, you must verify access rights repeatedly.

NOTE *External, internal, and CHV key files are relevant, but RSA key sets are always absolute. The card never searches for RSA keys outside the local directory.*

Recommended Access Rights for Key Files

SchlumbergerSema recommends that you allow card administrators to have *write* access for the contents of most key files, but no *read* access. Access rights are discussed in more detail in the next section, “Access Rights and Security.”

An internal key file is an exception to this rule, if you add keys to the file by calling the `Generate DES Key` command. In this case, the card administrator should have read access in order to retrieve the key values generated and added to the internal key file.

Cardholder Verification Files (CHV1 and CHV2)

The cardholder verification files (EF_{CHV1} and EF_{CHV2}) are transparent elementary files. EF_{CHV1} files have the reserved file ID 0000; EF_{CHV2} files have the reserved file ID 0100. You can use CHV files to control access to card resources. For example, enable one set of access rights for a cardholder and other access rights for a card administrator.

Verifying a CHV Key — To use a CHV key to gain access to the card, satisfy the CHV1 or CHV2 access condition (AC) by calling the `Verify CHV` command and correctly presenting the appropriate PIN.

Creating a CHV Key File — To create a CHV key file, call a `Create File` command and use the file structure described in the next topic. If you create an EF_{CHV1} with a CHV1 AC logged in, the new EF_{CHV1} becomes the relevant one—terminating any CHV1 AC that is logged in. To avoid terminating the CHV1 AC, you can create the EF_{CHV1} without initializing the data field and activation byte. The new EF_{CHV1} is invalid until you initialize it properly.

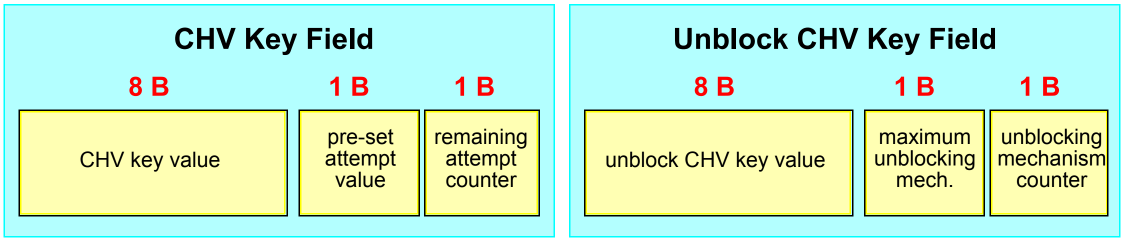
CHV File Structure

The structure of EF_{CHV1} and EF_{CHV2} files are identical, and each one has 16 bytes of input parameters. The format is shown in the following table and illustration.

CHV File Format

Byte(s)	Description	Length
1	Activation byte. The 0 bit value sets the file to be active (1) or inactive (0). To set a file to be active, use any hexadecimal value that corresponds to a decimal value with 1 in the 0 bit position, such as `01h` (000000001 decimal).	1 B
2 –3	RFU	2 B
4–11	PIN value. (Use hexadecimal values to specify the PIN.)	8 B
12	Pre-set value for the number of verification attempts allowed.	1 B
13	Counter for the number of remaining verification attempts. If this value = `FFh`, the CHV key is blocked.	1 B
14–21	Unblocking PIN value.	8 B
22	Number of unblocking mechanisms allowed. The default value if CHV key is unblocked = 10 (`0Ah`). You cannot change this value.	1 B
23	Counter for the number of remaining unblocking mechanisms. If this value = `FFh`, the unblock key is blocked.	1 B

CHV Key Block Format

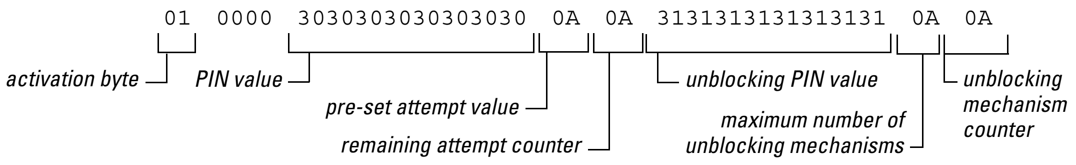


CHV File Access Condition Settings

The following table shows recommended settings for EF_{CHV1} and EF_{CHV2} files. *(For more information about access conditions, see "Access Rights and Security" starting on page 61.)*

Nibble	Commands Affected	Setting	Hex Value	Meaning
1	`Read Binary`	NEV	`F`	*Never allowed*
2	`Update Binary`	AUT	`4`	*Requires AUT*
3	`Read Binary Enciphered`	NEV	`F`	*Never allowed*
4	`Update Binary Enciphered`	AUT	`4`	*Requires AUT*
5	`Rehabilitate`	AUT	`4`	*Requires AUT*
6	`Invalidate`	AUT	`4`	*Requires AUT*

Example of a CHV File Data Field


01 0000 3030303030303030 0A 0A 3131313131313131 0A 0A
activation byte
PIN value
pre-set attempt value
remaining attempt counter
unblocking PIN value
maximum number of unblocking mechanisms
unblocking mechanism counter

The data field in the example CHV file indicates the file has these characteristics:

- The file is active.
- PIN value = 00000000 (`3030303030303030` in ASCII format). (You can use either numeric or standard ASCII characters for the PIN value.)
- Pre-set value for the number of attempts allowed to verify the PIN = 10, and all 10 attempts are currently available. (Counter is set to 10, `0A`h.)
- Unblocking PIN value = 11111111 (`3131313131313131` in ASCII format).
- Number of mechanisms allowed to unblock a CHV key = 10, and all 10 mechanisms are currently available. (Counter is set to 10, `0A`h.)

External Key File

Cryptoflex test cards come with a simple file system that contains a master file and an external key file ($EF_{Key\ Ext}$). The external key file is a transparent elementary file with the reserved file ID 0011. External key files can contain one or more DES or double-length 3DES keys.

The Application Authorization Key

When you receive a new card, it is locked to prevent tampering. You use the application authorization key (AAK), also called the transport key, to gain access to the card. If you order a shipment of cards, you typically receive the AAK value in a non-electronic format to prevent eavesdropping.

The AAK is stored in the external key file, which is part of the card's default file structure. On Cryptoflex cards, the AAK is key 1 and is in key slot 2 (as shown in the illustration on page 207). Verify the AAK in the Smart Card Toolkit or by calling the `Verify Key` command if you are in a secure physical environment. In an unsecured environment use the `External Authenticate Using DES` command.

You have three chances to present the AAK correctly. The key has a counter that tracks the number of failed attempts. If you enter the key incorrectly until the counter reaches its minimum value, the key is blocked. (You cannot unblock a blocked AAK.) The minimum attempt counter value is 0 (`00`h).

The AAK on a new test card is the only key you can use to gain access to the card. This key is also essential for continued use of the card, so be careful to make a record of any change you make to its value. Never delete the root-level external key file or the AAK, or you will permanently lock yourself out of the card.

The default external key file on a test card has space for a third key (key 2, which occupies bytes 42–53 of the file's data body).

Uses for the Keys in External Key Files

You can specify a key in the external key file for use in the following operations.

Authenticating the Terminal's Identity with a Plaintext Key

For easy access during program development, you can satisfy the AUT access condition (AC) by calling the `Verify Key` command and correctly presenting the AAK key in the external key. The card grants you the AUT access rights associated with the key you present.

Authenticating the Terminal's Identity Cryptographically

To authenticate the terminal's identity to the card in a secure way, use a DES or 3DES key in the external key file in one of these ways:

- Standard external authentication—Call the `External Authenticate Using DES` command and correctly present the required key.
- Digitally signing commands protected by a PRO AC— Correctly present the required key in the PRO AC procedure described in the next section.

Unlocking a New Card

To unlock a new card, send a `Verify Key` command with the AAK value.

External Key File Format

The following tables describe the default external key file's 16-byte input parameter string and file body. If an external key file contains multiple keys, the data string continues in the same pattern, beginning with the new key's length.

Example: Format of External (or Internal) Key File Input Parameter String

Byte(s)	Description	Example Value(s)	Length
1–2	RFU: Must = `FF`h	`FF FF`	2 B
3–4	File size (See page 98.) *Example: 26 bytes, which can store 2 DES keys*	`00 1A`	2 B
5–6	File ID: `0011`h (EF$_{\text{Key Ext}}$)	`00 11`	2 B
7	File type: `01`h = Transparent EF	`01`	1 B
8–11	Access conditions (See page 99.) *Example values*: • *B1 – No `Increase` or `Decrease` (`00`)* • *B2 MSN – `Read Binary` = NEV (`F`)* • *B2 LSN – `Update Binary` = AUT (`4`)* • *B3 MSN – `Read Binary Enciphered` = NEV (`F`)* • *B3 LSN – `Update Binary Encipher` = AUT (`4`)* • *B4 MSN – `Rehabilitate` = AUT (`4`)* • *B4 LSN – `Invalidate` = AUT (`4`)*	`00 F4 F4 44`	4 B
12	File status • `00`h = Invalidated • `01`h = Activated *(Example value)*	`01`	1 B
13	Length of following data (AC key settings): `03`h	`03`	1 B
14–16	Key numbers for ACs set in bytes 9–11. (Note that no key numbers are included for byte 8, which enables or disables the `Increase` and `Decrease` commands, but does not set any ACs. *Example Case: Use key number 1 in the external key file for all AUT commands.*	`11 11 11`	3 B

Format and Contents of the Default External Key File Body

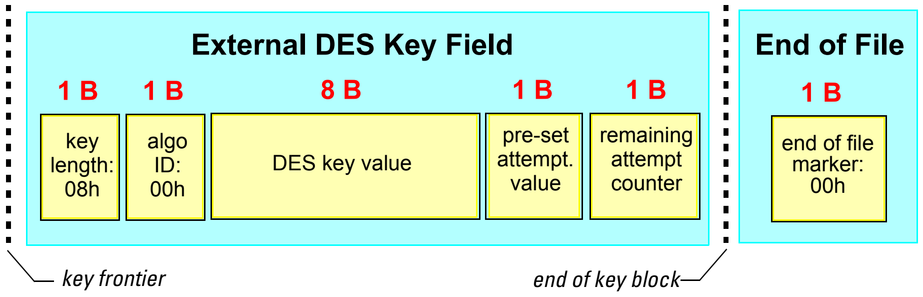
Byte(s)	Description	Example Value	Length
1	**RFU:** Any value	—	1 B
2	**Key length** The card uses the key length byte to jump between key slots when it searches for a key. • `00`h = No more keys in file. • `01`h = No key exists for the key slot, so the card can skip this byte and read the next key length byte. (Used to advance to a later key number without creating a full key slot, so you can number keys nonconsecutively without wasting file space.) • `08`h = 8-byte DES key *(Example value)* • `10`h = 16-byte double-length 3DES key *Example: The first key in the default EF$_{Key\ Ext}$ is an 8-byte key used only during manufacturing.*	`08`	1 B
3	**Algorithm ID** Identifies the key type. (See page 98.) • `00`h = Single-length DES key *(Example value)* • `02`h = Double-length 3DES key	`00`	1 B
4–11 *or* 4–19	**Key value** If the key is a double-length 3DES key, its two 8-byte keys are stored as a single 16-byte key value. *Example: Key value varies*	—	*x* B
12 *or* 20	**Pre-set attempt value** Value set for the number of verification attempts allowed. *Example value: 10*	`0A`	1 B
13 *or* 21	**Remaining attempt counter** Counter for the number of remaining verification attempts allowed before the key is blocked. If this value is `FF`h, the key is blocked. (One blocked key in the file does not cause any other keys to become blocked.) *Example value: 10*	`0A`	1 B
14 *or* 22	**Key length (2)** Length of next key *Example value: The second key in the default EF$_{Key\ Ext}$ is the 8-byte application authorization key (AAK), known as the transport key in earlier cards.* For information about verifying this key in the Smart Card Toolkit application, see page 19.	`08`	

Byte(s)	Description	Example Value	Length
15 *or* 23	**Algorithm ID (2)** See algorithm ID description in byte 3. *Example value:* `00`*h (Single-length DES)*	`00`	1 B
16–23 *or* 24–35	**Key value (2)** *Example: The default AAK for the Cryptoflex 16K card shipped with the Cyberflex Access SDK =* `2C 15 E5 26 E9 3E 8A 19`*. (Check your card's AAK value.)*	—	8/16 B
24 *or* 36	**Pre-set attempt value (2)** *Example value: 10 (by default)*	`0A`	1 B
25 *or* 37	**Remaining attempt counter (2)** *Example value: 10*	`0A`	1 B
26 *or* 38	**Key length (2)** Length of next key. *Example value: None.* The `00`h value marks the end of the file.	`00`	1 B
...	... *(If additional keys are included, key data follows)*	...	...

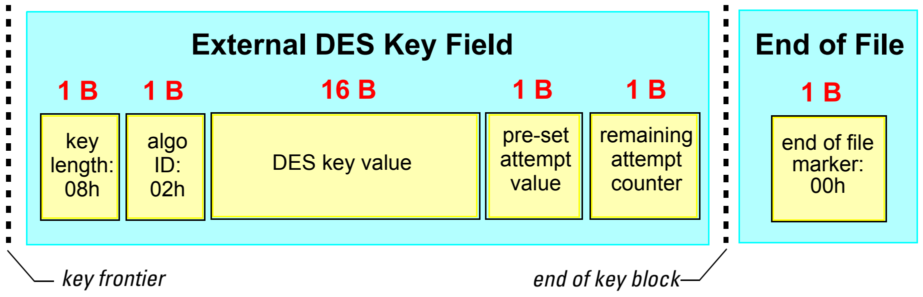
NOTE *Each directory on the card can have its own EF$_{Key\ Ext}$. Each EF$_{Key\ Ext}$ can contain a maximum of 16 keys.*

Illustrations of External Key Data

Single-Length DES Key — The following illustration shows the format of a single-length DES key field in an $EF_{Key\ Ext}$.



Double-Length 3DES Key — The following illustration shows the format of a double-length 3DES key field in an $EF_{Key\ Ext}$.



External Key File Access Condition Settings

The following table shows recommended settings for an $EF_{Key\ Ext}$. *(For more information about access conditions, see “Access Rights and Security” starting on page 61.)*

Byte	Commands Affected	Hex Value	Setting	Meaning
B1	`Increase/Decrease`	`00`	—	*Disabled: Not applicable*
B2 MSN	`Read Binary`	`F`	NEV	*Never allowed*
B2 LSN	`Update Binary`	`4`	AUT	*Requires AUT (See page 81)*
B3 MSN	`Read Binary Enciphered`	`F`	NEV	*Never allowed*
B3 LSN	`Update Binary Enciphered`	`4`	AUT	*Requires AUT*
B4 MSN	`Rehabilitate`	`4`	AUT	*Requires AUT*
B4 LSN	`Invalidate`	`4`	AUT	*Requires AUT*

Internal Key File

The internal key file ($EF_{Key\ Int}$) is a transparent elementary file (EF), with the reserved file ID 0001. Internal key files typically contain one or more DES or double-length 3DES keys.

Uses for Internal Keys — The card uses internal keys for the following types of operations:

- **Cipher Block Chaining (CBC) Operations** — (`DES Block Init` and `DES Block` commands). Use internal keys to compute CBC hashes on sensitive data you want to transmit or store.
- **Internal Authentication** — You typically use internal cryptographic keys to authenticate the card's identity to the terminal (`Internal Authenticate Using DES` command).

The following table describes the format of the first key stored in a Cryptoflex 16K card's internal key file. If the file contains multiple keys, the data string continues in the same pattern. (An internal key file has a 16-byte input parameter string.)

Byte(s)	Description	Length
1	RFU (Must be set to a value other than `00`h.)	1 B
2	Key length. When the card searches for a key, it uses the key length byte to jump from key to key. • `00`h = File contains no more keys. • `01`h = No key exists for the key slot, so the card can skip one byte and read the next key length byte. *Note: You can use the `01` value for byte 2 to advance to a later key number without creating a full key slot. This enables you to apply custom numbering to keys without wasting file space.*	1 B
3	Algorithm ID, which identifies the key type: • `00`h = Single-length DES • `02`h = Double-length 3DES	1 B
4–11 *or* 4–19	Key value (An 8-byte DES or 16-byte 3DES key). If the key is a 3DES key, its two 8-byte keys are stored as a single 16-byte key value.	x B

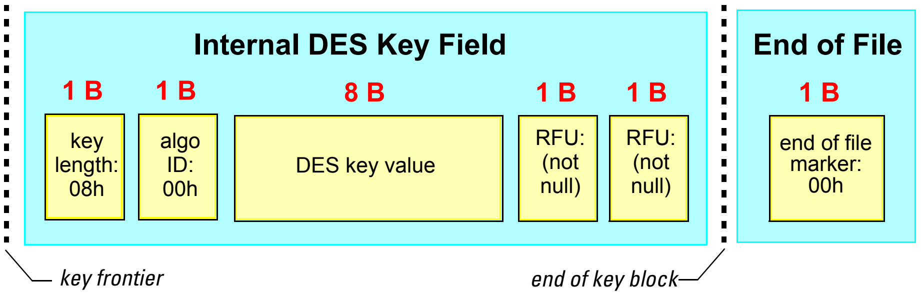
Byte(s)	Description	Length
12 *or* 20	RFU (Must be set to a value other than `00`h, or the internal keys are blocked.)	1 B
13 *or* 21	RFU (Must be set to a value other than `00`h, or the internal keys are blocked.)	1 B
14 *or* 22	Length of next key	1 B
...	... *(If additional keys are included, key data follows)*	...

NOTES

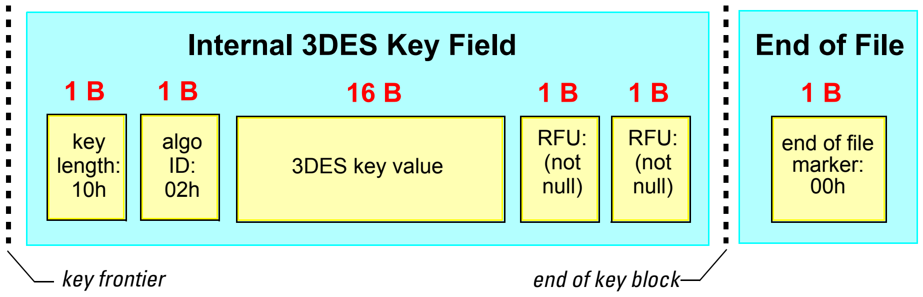
- *For information about an internal key file's input parameters, see the table on page 26.*
- *For examples of key file contents, see the table on page 27.*

Illustrations of Internal Key Data

Single-Length DES Key — The following illustration shows the format of a single-length DES key field in an internal key file.



Double-Length 3DES Key — The following illustration shows the format of a double-length 3DES key field in an internal key file



Initializing or Updating an Internal Key File — *If you set `00h` as the value for any of the RFU bytes in an internal key file, the associated key is blocked. You must set the RFU bytes to a value other than null.*

Internal Key File Access Condition Settings

The following table shows recommended settings for an $EF_{Key\ Int}$. *(For more information about access conditions, see "Access Rights and Security" starting on page 61.)*

Byte	Commands Affected	Hex Value	Setting	Meaning
B1	`Increase/Decrease`	`00`	—	*Disabled: Not applicable*
B2 MSN	`Read Binary`	`F`	NEV	*Never allowed*
B2 LSN	`Update Binary`	`4`	AUT	*Requires AUT (See page 81)*
B3 MSN	`Read Binary Enciphered`	`F`	NEV	*Never allowed*
B3 LSN	`Update Binary Enciphered`	`4`	AUT	*Requires AUT*
B4 MSN	`Rehabilitate`	`4`	AUT	*Requires AUT*
B4 LSN	`Invalidate`	`4`	AUT	*Requires AUT*

RSA Key Files

The private ($EF_{RSA\text{-}PRI}$) key file and public ($EF_{RSA\text{-}PUB}$) key file each contain half of an RSA key pair. A private key file has the reserved file ID 0012. A public key file's reserved file ID is 1012.

A single private or public file can hold the private exponent or public modulus for multiple keys—to a maximum of 15 keys. There is no limit on the size of the key file, other than the limits imposed by the size of the parent DF and the amount of available EEPROM on the card. A single file can hold keys of varying lengths.

32K+SS V1
32K+e-gate

Public keys can be generated in three different formats, specified using the P1 parameter of the Generate RSA Keys command. See `Generate RSA Keys` command (described on page 127) for information about using the P1 parameter to specify the public key format.

These are the three available public key formats:

- All public components are stored in the public key file.
- Only the public modulus and exponent are stored in the public key file.
- Only the public exponent is stored in the public key file.

All three formats may co-exist within the same public key file.

The RSA key type refers to the size of the public modulus *N*. The best choice of an RSA key type depends on security needs, balanced against the increased time it takes to perform operations with larger keys.

Key block lengths for private key files corresponding to each RSA key type are shown in the following table.

RSA Key Type	**Private Key Block Length**
512-bit	163 bytes
768-bit	243 bytes
1024-bit	323 bytes
2048-bit	643 bytes

Key block lengths for public key files corresponding to each RSA key type and public key format are shown in the following table.

RSA Key Type	**Public Key Block Length with all components**	**Public Key Block Length with public module and exponent only**[1]	**Public Key Block Length with public exponent only**[1]
512-bit	167 bytes	71 bytes	7 bytes
768-bit	247 bytes	103 byte	7 bytes
1024-bit	327 bytes	135 bytes	7 bytes
2048-bit	647 bytes	263 bytes	7 bytes

1 Available for 32K+SS V1 and 32K+e-gate cards only.

Uses for RSA Keys

Digital Signatures for Secure Email or Stored Data

You can use RSA keys to digitally sign data you want to send or store in a secure manner, or to receive encrypted or signed data.

To execute RSA signature commands successfully, you must initialize both the private key and corresponding public key.

- You can generate and automatically store RSA key pairs on the card by calling the `Generate RSA Keys` command.
- To generate a signature with a 1024-bit or smaller key, call the `RSA Signature (Internal Auth)` command. To compute a 2048-bit signature for a Cryptoflex card, call the `RSA Signature Intermediate` and `RSA Signature Last` commands.

Digital Signatures for Internal Authentication

You can use an RSA key signature for internal authentication (but not external authentication). This operation involves these steps:

1. The host calls a `Verify CHV` command to establish access rights for the `RSA Signature (Internal Auth)` command.
2. The host send an `RSA Signature (Internal Auth)` command that instructs the card to sign the challenge. As input data, the host sends a challenge (a random value that is 64, 96, or 128 bytes long).

   The card signs the challenge with a specified on-card private key.
3. The host calls a `Get Response` command to retrieve the signed challenge.

   The card returns the signature.
4. The host performs a check to verify that the signed challenge from the card matches a signed challenge the host creates by using the corresponding public key.

   If the check shows that the data strings match, the host regards the card as trustworthy.

Working with Multiple RSA Key Pairs

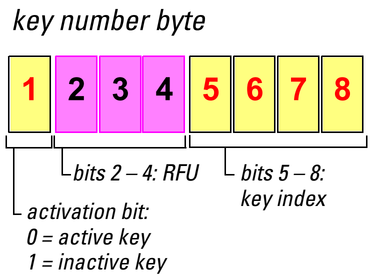
An RSA key file can contain a maximum of 15 keys. You may want to have multiple RSA key pairs on a card for a number of purposes, such as:

- Signing and receiving encrypted messages
- Signing work and personal messages
- Signing messages associated with different work roles
- Storing keys under escrow to guard against loss, while you use a non-escrowed key pair on a daily basis

Unlike other key types, RSA keys do not have domains. The card does not search for RSA keys anywhere but in the local directory.

Key Numbers in Public and Private Key Files

The key number byte (byte 3) of public and private key files is formatted as shown in the following illustration.



Bit(s)	Description
1	Activation bit: • 0 – Activates the key. • 1 – Invalidates any key value that follows, so the key cannot be used by any command. To reactivate an invalidated key, you must overwrite the key field by calling a `Generate RSA Keys` command.
2 – 4	RFU = 000
5 – 8	Key index: A unique value of 0001–1111 binary. As you can see from the preceding illustration, if you concatenate the key number byte value with an activation bit of 0 the key is active, the number = `01–0F`h (1–15 decimal). If the activation bit = 1 the key is inactive, and the number = `81–8F`h.

512-Bit Public and Private Key File Formats

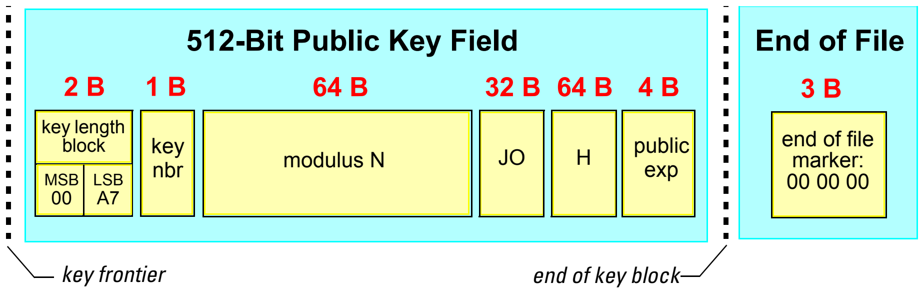
The following tables show the data string components for each 512-bit key stored in Cryptoflex card public and private key files. If a file contains multiple keys, the data string continues in the same pattern. (The data is stored LSB first.) Key files have 16-byte input parameter strings.

512-Bit Public Key File Format with all public components

Byte(s)	Description	Length
1	Length of the key block, MSB: `00`h	1 B
2	Length of the key block, LSB: `A7`h (167 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4–67	Public modulus (*N*)	64 B
68–99	*J0:* Montgomery constant	32 B
100–163	*H:* Montgomery constant	64 B
164–167	Public exponent (*e*)	4 B
...	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	...
final 3 B	*The final 3 bytes of the file must have the value `00 00 00`h to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

Public keys in this format include the Montgomery constants J0 and H, which must be contiguous.

The following illustration shows the format of a 512-bit public key file with all public components.

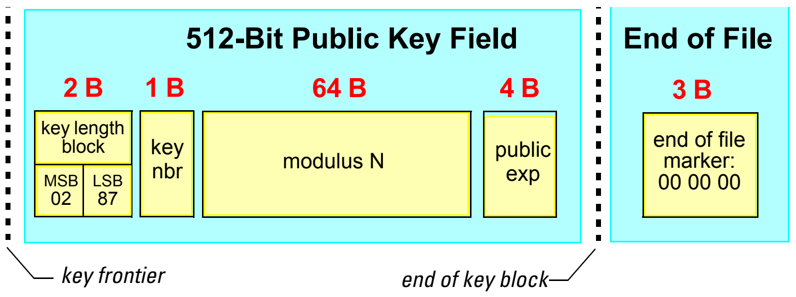


512-Bit Public Key File Format with public modulus and exponent only[1]

Byte(s)	Description	Length
1	Length of the key block, MSB: `00`h	1 B
2	Length of the key block, LSB: `A7`h (167 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4–67	Public modulus (*N*)	64 B
68-71	Public exponent (*e*)	4 B
...	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	...
final 3 B	*The final 3 bytes of the file must have the value `00 00 00`h to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

1. Available for 32K+SS V1 and 32K+e-gate cards only.

The following illustration shows the format of a 512-bit public key file with public modulus and exponent only.

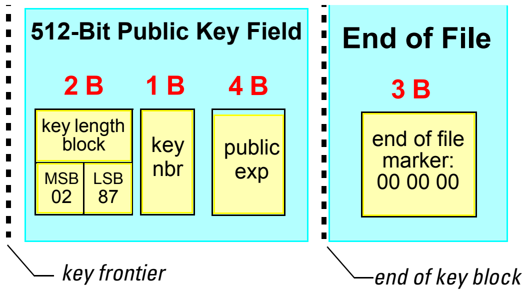


512-Bit Public Key File Format with public exponent only[1]

Byte(s)	Description	Length
1	Length of the key block, MSB: `00`h	1 B
2	Length of the key block, LSB: `A7`h (167 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4-7	Public exponent (*e*)	4 B
…	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	…
final 3 B	*The final 3 bytes of the file must have the value* `00 00 00`*h to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

1. Available for 32K+SS V1 and 32K+e-gate cards only.

The following illustration shows the format of a 512-byte public key file with public exponent only.



Public Key File Access Condition Settings

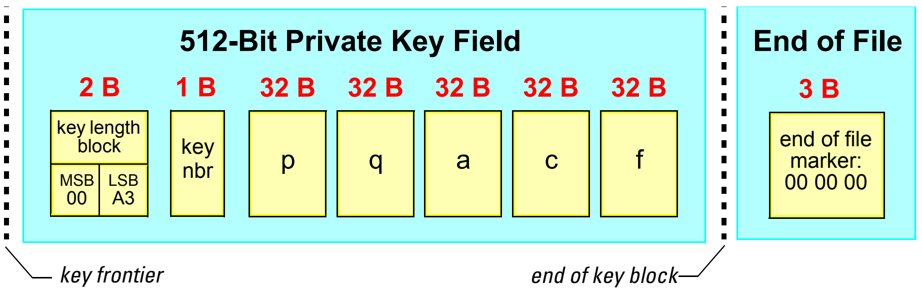
The following table shows recommended settings for $EF_{RSA-PUB}$ files. *(For more information about access conditions, see "Access Rights and Security" starting on page 61.)*

Nibble	Commands Affected	Setting	Hex Value	Meaning
1	`Read Binary`	ALW	0	*Always allowed*
2	`Update Binary`	AUT	4	*Requires AUT*
3	`Read Binary Enciphered`	ALW	0	*Always allowed*
4	`Update Binary Enciphered`	AUT	4	*Requires AUT*
5	`Rehabilitate`	AUT	4	*Requires AUT*
6	`Invalidate`	AUT	4	*Requires AUT*

512-Bit Private Key File Format

Byte(s)	Description	Length
1	Length of the key block, MSB: `00`h	1 B
2	Length of the key block, LSB: `A3`h (163 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4–35	Secret factor of the public modulus *P*	32 B
36–67	Secret factor of the public modulus *Q*	32 B
68–99	Inverse of the factor P ($a = Q^{-1} \bmod P$)	32 B
100–131	Private subexponent ($c = Ks \bmod (P - 1)$)	32 B
132–163	Private subexponent ($f = Ks \bmod (Q - 1)$	32 B
…	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	…
final 3 B	*The final 3 bytes of the file must have the value `00 00 00`h to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

The following illustration shows the format of a 512-bit private key file.



Private Key File Access Condition Settings

The following table shows recommended settings for $EF_{RSA\text{-}PRI}$ files. *(For more information about access conditions, see "Access Rights and Security" starting on page 61.)*

Nibble	Commands Affected	Setting	Hex Value	Meaning
1	`Read Binary`	NEV	`F`	*Never allowed*
2	`Update Binary`	AUT	`4`	*Requires AUT*
3	`Read Binary Enciphered`	NEV	`F`	*Never allowed*
4	`Update Binary Enciphered`	AUT	`4`	*Requires AUT*
5	`Rehabilitate`	AUT	`4`	*Requires AUT*
6	`Invalidate`	AUT	`4`	*Requires AUT*

768-Bit Public and Private Key File Formats

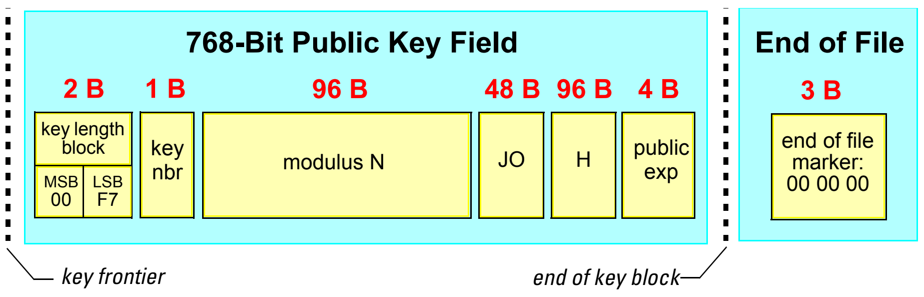
The following tables show the data string components for each 768-bit key stored in Cryptoflex card public and private key files. If a file contains multiple keys, the data string continues in the same pattern. (The data is stored LSB first.) Key files have 16-byte input parameter strings.

768-Bit Public Key File Format with all public components

Byte(s)	Description	Length
1	Length of the key block, MSB: `00`h	1 B
2	Length of the key block, LSB: `F7`h (247 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4–99	Public modulus (*N*)	96 B
100–147	*J0*: Montgomery constant	48 B
148–243	*H*: Montgomery constant	96 B
244–247	Public exponent (*e*)	4 B
...	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	...
final 3 B	*The final 3 bytes of the file must have the value* `00 00 00`*h to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

NOTE *Public keys in this format use the Montgomery constants J0 and H, which must be contiguous.*

The following illustration shows the format of a 768-bit public key file with all public components.

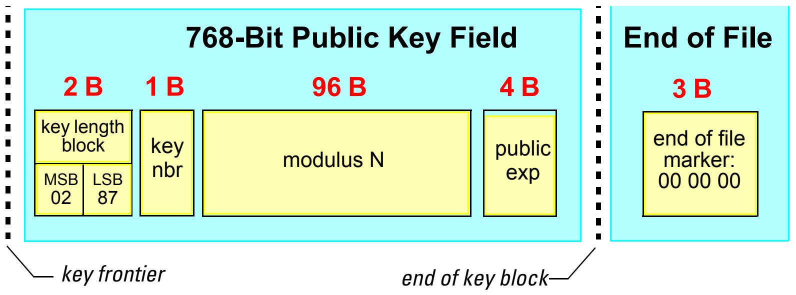


768-Bit Public Key File Format with public modulus and exponent only[1]

Byte(s)	Description	Length
1	Length of the key block, MSB: `00`h	1 B
2	Length of the key block, LSB: `F7`h (247 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4–99	Public modulus (*N*)	96 B
100-103	Public exponent (*e*)	4 B
…	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	…
final 3 B	*The final 3 bytes of the file must have the value* `00 00 00`*h to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

1. Available for 32K+SS V1 and 32K+e-gate cards only.

The following illustration shows the format of a 768-bit public key file with public modulus and exponent only.

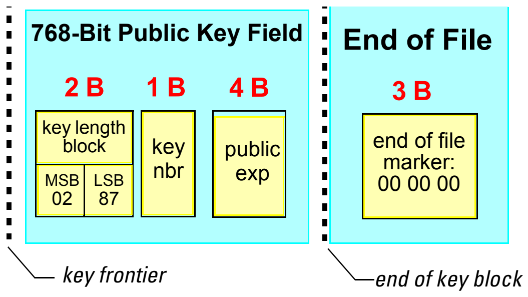


768-Bit Public Key File Format with public exponent only[1]

Byte(s)	Description	Length
1	Length of the key block, MSB: `00`h	1 B
2	Length of the key block, LSB: `F7`h (247 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4-7	Public exponent (*e*)	4 B
…	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	…
final 3 B	*The final 3 bytes of the file must have the value `00 00 00`h to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

1. Available for 32K+SS V1 and 32K+e-gate cards only.

The following illustration shows the format of a 768-bit public key file with public exponent only.



Public Key File Access Condition Settings

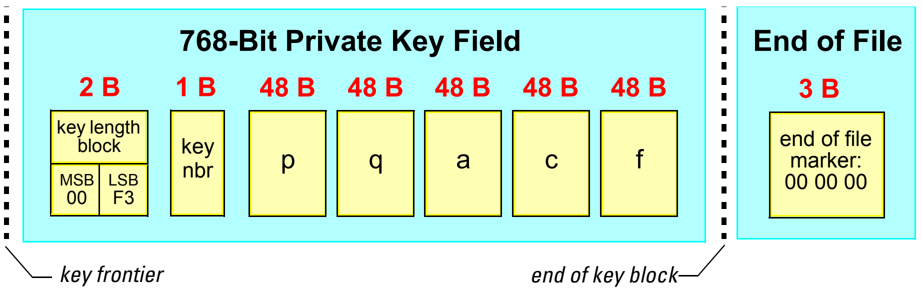
The following table shows recommended settings for $EF_{RSA\text{-}PUB}$ files. *(For more information about access conditions, see "Access Rights and Security" starting on page 61.)*

Nibble	Commands Affected	Setting	Hex Value	Meaning
1	`Read Binary`	ALW	0	*Always allowed*
2	`Update Binary`	AUT	4	*Requires AUT*
3	`Read Binary Enciphered`	ALW	0	*Always allowed*
4	`Update Binary Enciphered`	AUT	4	*Requires AUT*
5	`Rehabilitate`	AUT	4	*Requires AUT*
6	`Invalidate`	AUT	4	*Requires AUT*

768-Bit Private Key File Format

Byte(s)	Description	Length
1	Length of the key block, MSB: `00`h	1 B
2	Length of the key block, LSB: `F3`h (243 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4–51	Secret factor of the public modulus *P*	48 B
52–99	Secret factor of the public modulus *Q*	48 B
100–147	Inverse of the factor P ($a = Q^{-1}$ *mod* P)	48 B
148–195	Private subexponent ($c = Ks$ *mod* $[P - 1]$)	48 B
196–243	Private subexponent ($f = Ks$ *mod* $[Q - 1]$)	48 B
...	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	...
final 3 B	*The final 3 bytes of the file must have the value `00 00 00`h to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

The following illustration shows the format of a 768-bit private key file.



Private Key File Access Condition Settings

The following table shows recommended settings for EF$_{\text{RSA-PRI}}$ files. *(For more information about access conditions, see "Access Rights and Security" starting on page 61.)*

Nibble	Commands Affected	Setting	Hex Value	Meaning
1	`Read Binary`	NEV	`F`	*Never allowed*
2	`Update Binary`	AUT	`4`	*Requires AUT*
3	`Read Binary Enciphered`	NEV	`F`	*Never allowed*
4	`Update Binary Enciphered`	AUT	`4`	*Requires AUT*
5	`Rehabilitate`	AUT	`4`	*Requires AUT*
6	`Invalidate`	AUT	`4`	*Requires AUT*

1024-Bit Public and Private Key File Formats

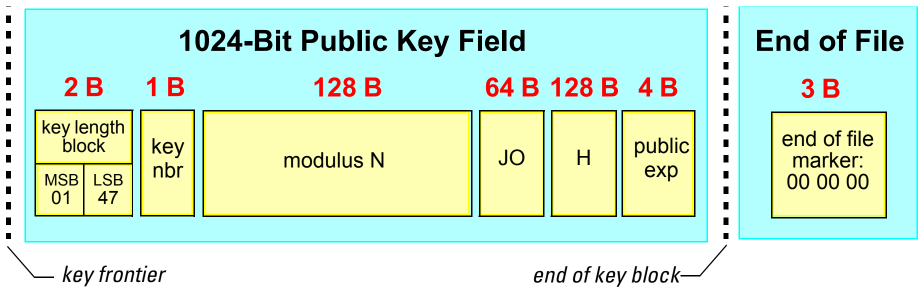
The following tables show the data string components for each 1024-bit key stored in Cryptoflex card's public and private key files. If a file contains multiple keys, the data string continues in the same pattern. (The data is stored LSB first.) Key files have 16-byte input parameter strings.

1024-Bit Public Key File Format with all public components

Byte(s)	Description	Length
1	Length of the key block, MSB: `01h`	1 B
2	Length of the key block, LSB: `47h` (`0147h`= 327 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4–131	Public modulus (*N*)	128 B
132–195	*J0:* Montgomery constant	64 B
196–323	*H:* Montgomery constant	128 B
324–327	Public exponent (*e*)	4 B
...	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	...
final 3 B	*The final 3 bytes of the file must have the value `00 00 00h` to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

NOTE *Public keys in this format use the Montgomery constants J0 and H, which must be contiguous.*

The following illustration shows the format of a 1024-bit public key file with all public components.

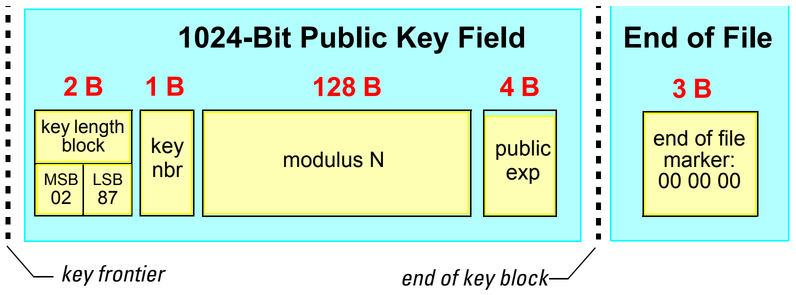


1024-Bit Public Key File Format with public modulus and exponent only[1]

Byte(s)	Description	Length
1	Length of the key block, MSB: `01h`	1 B
2	Length of the key block, LSB: `47h` (`0147h`= 327 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4–131	Public modulus (*N*)	128 B
132-135	Public exponent (*e*)	4 B
...	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	...
final 3 B	*The final 3 bytes of the file must have the value `00 00 00`h to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

1. Available for 32K+SS V1 and 32K+e-gate cards only.

The following illustration shows the format of a 1024-bit key file with public modulus and exponent only.

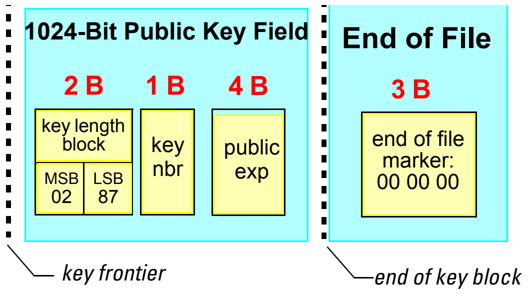


1024-Bit Public Key File Format with public exponent only[1]

Byte(s)	Description	Length
1	Length of the key block, MSB: `01h`	1 B
2	Length of the key block, LSB: `47h` (`0147h`= 327 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4-7	Public exponent (*e*)	4 B
...	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	...
final 3 B	*The final 3 bytes of the file must have the value `00 00 00h` to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

1. Available for 32K+SS V1 and 32K+e-gate cards only.

The following illustration shows a 1024-bit public key file with public exponent only.



Public Key File Access Condition Settings

The following table shows recommended settings for $EF_{RSA\text{-}PUB}$ files. *(For more information about access conditions, see "Access Rights and Security" starting on page 61.)*

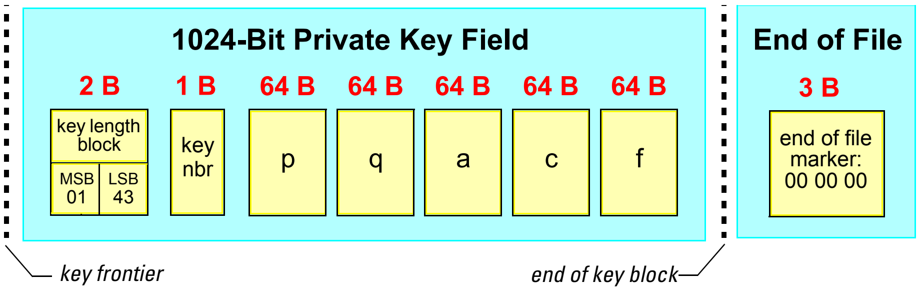
Nibble	Commands Affected	Setting	Hex Value	Meaning
1	`Read Binary`	ALW	`0`	*Always allowed*
2	`Update Binary`	AUT	`4`	*Requires AUT*
3	`Read Binary Enciphered`	ALW	`0`	*Always allowed*
4	`Update Binary Enciphered`	AUT	`4`	*Requires AUT*
5	`Rehabilitate`	AUT	`4`	*Requires AUT*
6	`Invalidate`	AUT	`4`	*Requires AUT*

1024-Bit Private Key File Format

Byte(s)	Description	Length
1	Length of the key block, MSB: `01h`	1 B
2	Length of the key block, LSB: `43h` (`0143h` = 323 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4–67	Secret factor of the public modulus P	64 B

Byte(s)	Description	Length
68–131	Secret factor of the public modulus Q	64 B
132–195	Inverse of the factor P ($a = Q^{-1} \mathit{mod}\ P$)	64 B
196–259	Private subexponent ($c = Ks\ \mathit{mod}\ (P - 1)$)	64 B
260–323	Private subexponent ($f = Ks\ \mathit{mod}\ (Q - 1)$)	64 B
...	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	...
final 3 B	*The final 3 bytes of the file must have the value* `00 00 00h` *to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

The following illustration shows the format of a 1024-bit private key file.



Private Key File Access Condition Settings

The following table shows recommended settings for $EF_{RSA\text{-}PRI}$ files. *(For more information about access conditions, see “Access Rights and Security” starting on page 61.)*

Nibble	Commands Affected	Setting	Hex Value	Meaning
1	`Read Binary`	NEV	`F`	*Never allowed*
2	`Update Binary`	AUT	`4`	*Requires AUT*
3	`Read Binary Enciphered`	NEV	`F`	*Never allowed*
4	`Update Binary Enciphered`	AUT	`4`	*Requires AUT*

Nibble	Commands Affected	Setting	Hex Value	Meaning
5	`Rehabilitate`	AUT	`4`	*Requires AUT*
6	`Invalidate`	AUT	`4`	*Requires AUT*

2048-Bit Public and Private Key File Formats

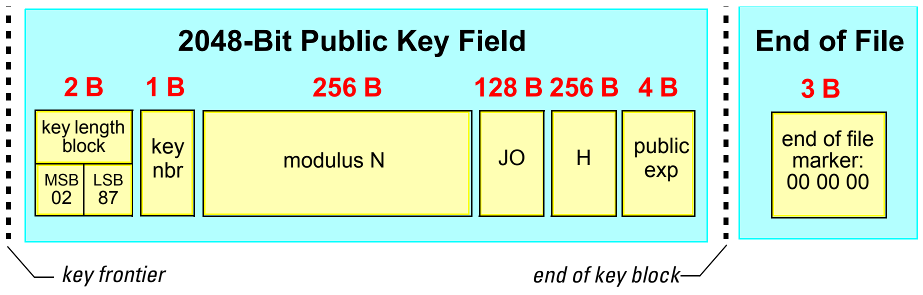
The following tables show the data string components for each 2048-bit key stored in Cryptoflex card public and private key files. If a file contains multiple keys, the data string continues in the same pattern. (The data is stored LSB first.) Key files have 16-byte input parameter strings.

2048-Bit Public Key File Format with all public components

Byte(s)	Description	Length
1	Length of the key block, MSB: `02`h	1 B
2	Length of the key block, LSB: `87`h (`0287`h = 647 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4–259	Public modulus (*N*)	256 B
260–387	*J0:* Montgomery constant	128 B
388–643	*H:* Montgomery constant	256 B
644–647	Public exponent (*e*)	4 B
...	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	...
final 3 B	*The final 3 bytes of the file must have the value `00 00 00`h to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

NOTE *Public keys in this format use the Montgomery constants J0 and H, which must be contiguous.*

The following illustration shows the format of a 2048-bit public key file with all public components.

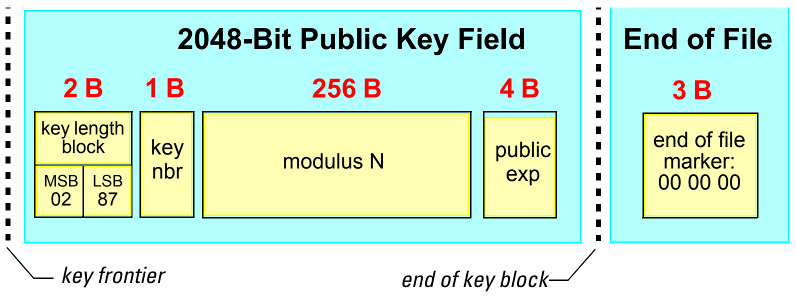


2048-Bit Public Key File Format with public modulus and exponent only[1]

Byte(s)	Description	Length
1	Length of the key block, MSB: `02`h	1 B
2	Length of the key block, LSB: `87`h (`0287`h = 647 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4–259	Public modulus (*N*)	256 B
260-263	Public exponent (*e*)	4 B
...	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	...
final 3 B	*The final 3 bytes of the file must have the value `00 00 00`h to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

1. Available for 32K+SS V1 and 32K+e-gate cards only.

The following illustration shows the format of a 2048-bit public key file with public modulus and exponent only.

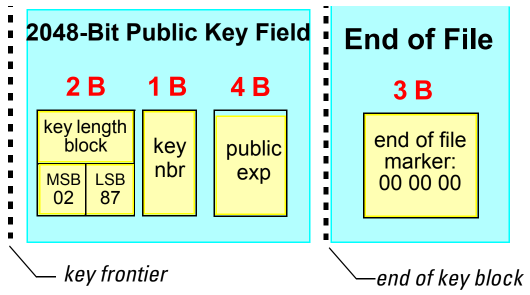


2048-Bit Public Key File Format with public exponent only[1]

Byte(s)	Description	Length
1	Length of the key block, MSB: `02h`	1 B
2	Length of the key block, LSB: `87h` (`0287h` = 647 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4-7	Public exponent (*e*)	4 B
...	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	...
final 3 B	*The final 3 bytes of the file must have the value* `00 00 00h` *to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

1. Available for 32K+SS V1 and 32K+e-gate cards only.

The following illustration shows the format of a 2048-bit public key file with public exponent only.



Public Key File Access Condition Settings

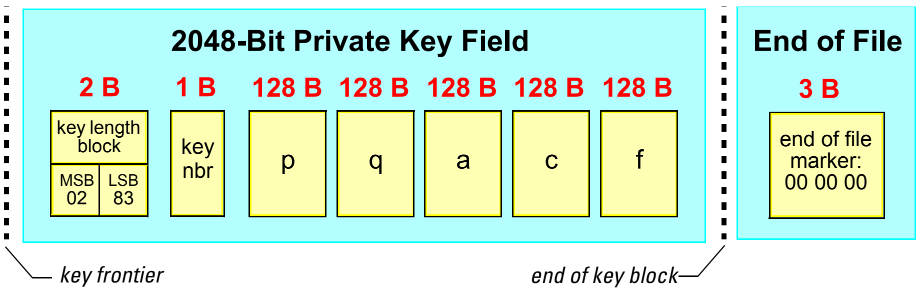
The following table shows recommended settings for EF$_{\text{RSA-PUB}}$ files. *(For more information about access conditions, see "Access Rights and Security" starting on page 61.)*

Nibble	Commands Affected	Setting	Hex Value	Meaning
1	`Read Binary`	ALW	0	*Always allowed*
2	`Update Binary`	AUT	4	*Requires AUT*
3	`Read Binary Enciphered`	ALW	0	*Always allowed*
4	`Update Binary Enciphered`	AUT	4	*Requires AUT*
5	`Rehabilitate`	AUT	4	*Requires AUT*
6	`Invalidate`	AUT	4	*Requires AUT*

2048-Bit Private Key File Format

Byte(s)	Description	Length
1	Length of the key block, MSB: `02`h	1 B
2	Length of the key block, LSB: `83`h (`0283`h = 643 bytes)	1 B
3	Key number, formatted as shown on page 37	1 B
4–131	Secret factor of the public modulus *P*	128 B
132–259	Secret factor of the public modulus *Q*	128 B
260–387	Inverse of the factor P ($a = Q^{-1} \bmod P$)	128 B
388–515	Private subexponent ($c = Ks \bmod (P - 1)$)	128 B
516–643	Private subexponent ($f = Ks \bmod (Q - 1)$)	128 B
...	*If additional keys are included, the format repeats at this point, beginning with the length bytes.*	...
final 3 B	*The final 3 bytes of the file must have the value `00 00 00`h to mark the end of the key file, where bytes 1 – 2 = the length of the following key block and byte 3 = the key number. (The key number value 00h is reserved for this use.)*	*3 B*

The following illustration shows the format of a 2048-bit private key file.



3

Access Rights and Security

Introduction

Access to most smart card files is subject to security clearance. The user obtains security clearance by logging in an access condition (AC). You set AC requirements on a file to control the use of certain commands on the file. For example, you can use access rights to control who adds files to a directory or changes file data. You specify the AC level and key for protected commands when you create the file, and this information is stored in the file's input parameter string.

This section covers the following topics:

Setting Access Rights on Card Operations

The master file on a new Cryptoflex card has default AC settings, which subsidiary files inherit unless you set overriding ACs when you create the subsidiary files. In this case, the file may have some locally defined ACs and may inherit some ACs from its parent directory.

AC s determine whether or not an action is allowed for a file—or, if the file is a directory, may also restrict actions on the files in the directory. If an action is allowed, the AC settings determine the security requirements (or security level).

To satisfy a security requirement, the user may have to log in an AC at some point before the command is issued. To log in an AC, users may verify their knowledge of a personal identification number (PIN) or may perform a cryptographic operation with a key that proves their identity. For added security, the command may also have to be digitally signed. You can also set an AC that enables all users to perform an action on the file, or you can forbid the action in any circumstances.

The default master file on a new card has predefined AC values (described on page 68). If you add a file to the card, you specify the ACs for it. To specify ACs, you set values in a command matrix—a 6-nibble string in the `Create File` input data. Each nibble holds a hexadecimal value that specifies which (if any) AC must be satisfied in order to execute the nibble's associated command or commands. Each type of file has its own command matrix (as described on page 64). You specify key numbers to use for any ACs you set—in a matching 6-nibble matrix of input data.

Access Condition Values

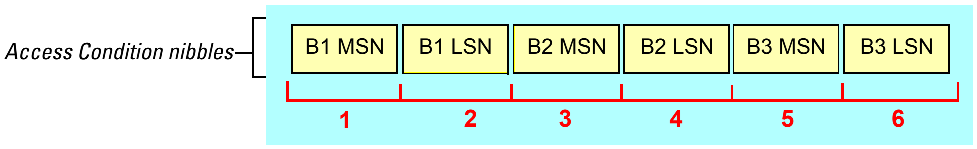
The following table summarizes the ACs that apply to Cryptoflex files. For a description of each access condition, see the page noted.

Access Conditions

Hex Value	Access Condition	Description of Restriction
`0`	**ALW** (page 71)	Always: It is always possible to perform the operation.
`1`	**CHV1** (page 72)	CHV1 verification must have been established (in a `Verify CHV` command) and remain in force.
`2`	**CHV2** (page 73)	CHV2 verification must have been established (in a `Verify CHV` command) and remain in force.
`3`	**PRO** (page 73)	Protected mode: The command must be signed by a verified PRO authentication digital signature.
`4`	**AUT** (page 81)	AUT must have been established (in a `Verify Key` or `External Authenticate Using DES` command) and remain in force.
`5`	*RFU*	Reserved for future use.
`6`	**CHV1 and PRO** (page 82)	CHV1 verification must have been established and remain in force, and the command must be signed by a verified PRO authentication digital signature.
`7`	**CHV2 and PRO** (page 82)	CHV2 verification must have been established and remain in force, and the command must be signed by a verified PRO authentication digital signature.
`8`	**CHV1 and AUT** (page 83)	Both CHV1 and AUT must have been established and remain in force.
`9`	**CHV2 and AUT** (page 83)	Both CHV2 and AUT must have been established and remain in force.
`A`	*RFU*	Reserved for future use.
`B`	*RFU*	Reserved for future use.
`C`	*RFU*	Reserved for future use.
`D`	*RFU*	Reserved for future use.
`E`	*RFU*	Reserved for future use.
`F`	**NEV** (page 84)	Never: The operation is never possible.

Commands That Are Subject to Access Conditions

This topic shows the commands associated with each nibble in the command matrix (illustrated below). As you can see, each file type has its own command matrix.



The following table shows the command matrix and corresponding commands, shown by file type.

Byte	Master File	DF (Directory	Transparent EF	Linear EF	Cyclic EF
B1 MSN	• `Dir Next`	• `Dir Next`	• `Read Binary`	• `Read Record` • `Read Record EMV` • `Seek`	• `Read Record`
B1 LSN	*RFU*	*RFU*	• `Update Binary`	• `Update Record`	• `Decrease`[1]
B2 MSN	• `Create File`	• `Delete File`	• `Read Binary Enciphered`	*RFU*	• `Increase`[1]
B2 LSN	• `Delete File`	• `Create File`	• `Update Binary Enciphered`	• `Create Record`	*RFU*
B3 MSN	• `Rehabilitate`	*RFU*	• `Rehabilitate`	• `Rehabilitate`	• `Rehabilitate`
B3 LSN	• `Invalidate`	*RFU*	• `Invalidate`	• `Invalidate`	• `Invalidate`

1 The Decrease and Increase commands are available only if you enable then when you create the cyclic EF (in byte 8 of the input data for the Create File command) The byte that precedes the command matrix holds the Decrease/increase setting.

Access Condition Summary

The following table shows the types of ACs you can set for each Cryptoflex command. ACs are categorized by file type or are noted in the Global column (if the AC setting is not file-specific).

Command	Global	MF	DF	Trans. EF	Cyclic EF	Linear EF
`Change CHV`	*none*					
`Create File`		*AUT*	*any AC*			
`Create Record`						*any AC*
`Decrease`					*any AC*	
`Delete File`		*AUT*	*any AC*			
`DES Block`	*CHV1*					
`DES Block Init`	*CHV1*					
`Dir Next`		*AUT*	*any AC*			
`External Auth Using DES`	*none*					
`Generate DES Key`	*CHV1*					
`Generate RSA Keys`	*CHV1*					
`Get AC Keys`	*CHV1*					
`Get Challenge`	*none*					
`Get Response`	*none*					
`Increase`					*any AC*	
`Internal Auth Using DES`	*CHV1*					
`Invalidate`		*AUT*		*any AC*	*any AC*	*any AC*
`Logout AC`	*none*					
`Read Binary`				*any but PRO*		
`Read Binary Enciphered`				*any but PRO*		
`Read Record`					*any but PRO*	*any but PRO*
`Read Record EMV`	*none*					
`Rehabilitate`		*AUT*		*any AC*	*any AC*	*any AC*
`RSA Signature (Int Auth)`	*CHV1*					

Command	Global	MF	DF	Trans. EF	Cyclic EF	Linear EF
`RSA Signature Intermediate`	*CHV1*					
`RSA Signature Last`	*CHV1*					
`Seek`						*any but PRO*
`Select`	*none*					
`Select EMV`	*none*					
`SHA-1 Intermediate`	*none*					
`SHA-1 Last`	*none*					
`Unblock CHV`	*none*					
`Update Binary`				*any AC*		
`Update Binary Enciphered`				*any but PRO*		
`Update Record`					*any AC*	*any AC*
`Verify CHV`	*none*					
`Verify Key`	*none*					

Setting Access Conditions and Key Numbers

Set ACs in bytes 9–11 of the input data for the `Create File` command. These 6 nibbles set security levels for the matrix of commands described on page 64. Set AC key numbers in bytes 14–16 of the input data for the `Create File` command. These 6 nibbles specify the key numbers required to satisfy each AC you set in the byte 9–11 command matrix.

NOTE *You cannot modify ACs after you create the file. To change a file's ACs, you must replace the file. You cannot modify the master file's ACs, unless you order cards that have been pre-personalized with a custom master file.*

Retrieving Access Condition Settings and Key Numbers

To Retrieve AC Settings:

- Call a `Get Response` command after you select a DF or EF, and examine bytes 8–11 of the response data.
- Call one or more `Dir Next` commands. Bytes 7–9 of the response data contain the AC settings for each file whose information you retrieve.

AC Key Settings: To retrieve the AC key settings, select a file, then call the `Get AC Keys` command. The card returns the key numbers required for the ACs set on the file.

Persistence of Access Rights

Once you satisfy an AC in a given context, that AC's access rights persist until one of the following events occurs:

- The card session ends (for example, power is reset or the card is removed from the reader).
- You enter a different key domain, either:
  - A lower-level directory that contains a CHV or key file of the type specified in the AC.
  - A higher-level directory that has a different relevant CHV or key file. *(For more information about relevant keys and key domains, see page 20.)*
- The AC key is deleted.
- The AC key is blocked by a series of unsuccessful verification attempts. (Note that a failed verification attempt always decrements the attempt counter, even if the key was verified before the failed attempt.)
- The AC key file becomes invalidated.
- The AC is logged out (with a `Logout AC` command).

Key Domains

If a command requires the use of an external key, internal key, or CHV key, the card uses a key from the *relevant* key file—the key file that protects the currently selected file. The card uses the key number specified in the AC key number matrix of the currently selected file (or the parent directory, if its AC applies). This number specifies the key the card checks or uses for encryption or decryption.

NOTE *For more information about relevant keys, see page 20.*

Access Condition Inheritance

A file is affected by its own AC properties and the AC properties of its parent directory. For example, the AC that enables you to delete an elementary file is inherited from the file's parent directory.

If you set ACs for lower-level directories that are different from upper-level directory settings, the local settings predominate. You can use this feature to set up different levels of security for different parts of the card's resources.

Default Access Condition Settings for the Master File

The default AC string for the master file is `4F4444`h, as shown in the following table.

Nibble 1 *Dir Next*	**Nibble 2** *RFU*	**Nibble 3** *Create File*	**Nibble 4** *Delete File*	**Nibble 5** *RFU*	**Nibble 6** *RFU*
AUT (`4`h)	NEV (`F`h)	AUT (`4`h)	AUT (`4`h)	AUT (`4`h)	AUT (`4`h)

AUT (nibbles 1, 3, 4, 5, 6) — The host application must be authenticated before a user can perform any of these actions:

- Retrieve data about the contents of the MF
- Add files to the MF
- Delete files from the MF

Keys Required for Master File Operations

The keys required for satisfying the MF ACs are specified in this string: `1`*x*`1111`. To satisfy the AUT AC for the `Dir Next`, `Create File` or `Delete File` commands, call the `Verify Key` or `External Authenticate Using DES` command. Use the key 1 value in the external key file located in the master file. (The card ignores the value specified in the *x* position.)

Examples of Access Conditions

AC Settings for a Directory

This example shows AC settings for a directory. The AC string in the example is `4F44FF`h, as shown in the following table.

Nibble 1 *Dir Next*	**Nibble 2** *RFU*	**Nibble 3** *Delete File*	**Nibble 4** *Create File*	**Nibble 5** *RFU*	**Nibble 6** *RFU*
AUT (`4`h)	NEV (`F`h)	AUT (`4`h)	AUT (`4`h)	NEV (`F`h)	NEV (`F`h)

AUT (nibbles 1, 3, 4) — The host application must be authenticated before a user can perform these actions:

- Examine the contents of the directory
- Add a file to the directory
- Delete a file from the directory

The keys required for satisfying the ACs set on the example external key file are specified in the string `2`*x*`22`*xx*. To satisfy the AUT AC for the `Dir Next`, `Create File` and `Delete File` commands, call the `Verify Key` or `External Authenticate Using DES` command. Use the key value for key 2 of the relevant external key file). (The card ignores the values specified in the *x* positions.)

AC Settings for an External Key File

This example shows the AC setting for an external key file, a transparent EF with the reserved ID of 0011. The AC string in the example is `FFF444`h, as shown in the following table.

Nibble 1 *Read Binary*	**Nibble 2** *Update Binary*	**Nibble 3** *Read Binary Enciphered*	**Nibble 4** *Update Binary Enciphered*	**Nibble 5** *Rehabilitate*	**Nibble 6** *Invalidate*
NEV (`F`h)	NEV (`F`h)	NEV (`F`h)	AUT (`4`h)	AUT (`4`h)	AUT (`4`h)

- *NEV (nibbles 1–3)* — No one can perform these actions:
  - Update key values with plaintext
  - Retrieve EF contents in enciphered form
- *AUT (nibbles 4–6)* — The host application must be authenticated before a user can perform these actions:
  - Update the EF values with enciphered data
  - Invalidate the key file
  - Rehabilitate the key file

The keys required for satisfying the ACs set on the example external key file are specified in the string *xxx*`111`. To satisfy the AUT AC for the `Update Binary Enciphered`, `Rehabilitate`, and `Invalidate` commands, call the `Verify Key` or `External Authenticate Using DES` command. Use the value for key 1 in the external key file. (The card ignores the values specified in the *x* positions.)

AC Settings for an Internal Key File

The following table shows recommended settings for an $EF_{Key\ Int}$.

Byte	Commands Affected	Hex Value	Setting	Meaning
B1	`Increase/Decrease`	`00`	—	*Disabled: Not applicable.*
B2 *MSN*	`Read Binary`	`F`	NEV	*Never allowed.*
B2 *LSN*	`Update Binary`	`4`	AUT	*Requires AUT. (See page 81)*
B3 *MSN*	`Read Binary Enciphered`	`F`	NEV	*Never allowed.*
B3 *LSN*	`Update Binary Enciphered`	`4`	AUT	*Requires AUT.*
B4 *MSN*	`Rehabilitate`	`4`	AUT	*Requires AUT.*
B4 *LSN*	`Invalidate`	`4`	AUT	*Requires AUT.*

ALW Access Condition

AC / Descriptive name – ALW / Always

Hex value – `0`

ALW specifies that the action is *always* possible. No restrictions are placed on the command.

CHV1 Access Condition

AC / Descriptive name – CHV1 / CardHolder Verified 1

Hex value – `1`

You can execute the command only after the relevant CHV1 key has been verified. The verification can occur immediately before the command is issued or earlier in the card session. The AC persists until an event terminates it (as described on page 67 and in the warning that follows).

Use a `Verify CHV` command to verify the credentials of the cardholder. (CHV1 is typically the card user.) The card is satisfied that the cardholder is legitimate if it receives a response that contains the correct PIN—a PIN value that matches the one stored in the relevant EF_{CHV1}.

CHV1 PINs are stored on the card in one or more EF_{CHV1} files. The card uses the relevant EF_{CHV1} for the currently selected file. The relevant EF_{CHV1} is the local one, or—if no local EF_{CHV1} exists—the relevant EF_{CHV1} is inherited.

A counter monitors the cardholder's attempts to enter a PIN. If the `Verify CHV` command fails, the counter decrements the available attempts. If the counter reaches the null value, you cannot make any more verification attempts.

You lose access permission if you submit a command that does not include the relevant PIN values. You are particularly likely to access in this way if you:

- *Move to a DF that contains a different EF_{CHV1} file.*
- *Move to a DF that inherits an EF_{CHV1} file's PIN values from a higher level, even though the new DF contains no EF_{CHV1} file itself.*
- *Create or update an EF_{CHV1} so that the previous ACs are no longer valid.*

If you execute a command that has a CHV1 AC, be aware of which EF_{CHV1} applies to the target file, and use the relevant PIN.

NOTES

- *Some commands require the CHV1 AC to be satisfied. These commands are shown in the table on page 65.*
- *For more information about relevant key files, see page 68.*
- *For information about EF_{CHV1} files, see page 21.*

CHV2 Access Condition

AC / Descriptive name – CHV2 / CardHolder Verified 2

Hex value – `2`

You can execute the command only after the relevant CHV2 key has been verified. The verification can occur immediately before the command is issued or earlier in the card session. The AC persists until an event terminates it (as described on page 67).

The card uses a `Verify CHV` command to verify the credentials of the CHV2 user (typically the card administrator). The card is satisfied that the administrator is legitimate if it receives a response that contains the correct PIN—a PIN value that matches the one stored in the relevant EF_{CHV2}.

NOTES

- *For more information, see the CHV1 description in the previous topic.*
- *For information about EF_{CHV2} files, see page 21.*

PRO Access Condition

AC / Descriptive name – PRO / Protected

Hex value – `3`

The PRO access condition is a protected command mode, in which the command is signed. The card supports the PRO access condition for commands that can change the contents or file status of sensitive files.

PRO authentication uses encryption to verify the identity of the command's originator, produces a cryptogram (a digital signature) of the command instruction, and appends the signature to the command. The card verifies the signature before it executes the protected command. Unlike the AUT AC, a PRO operation expires after authorizing a single command execution.

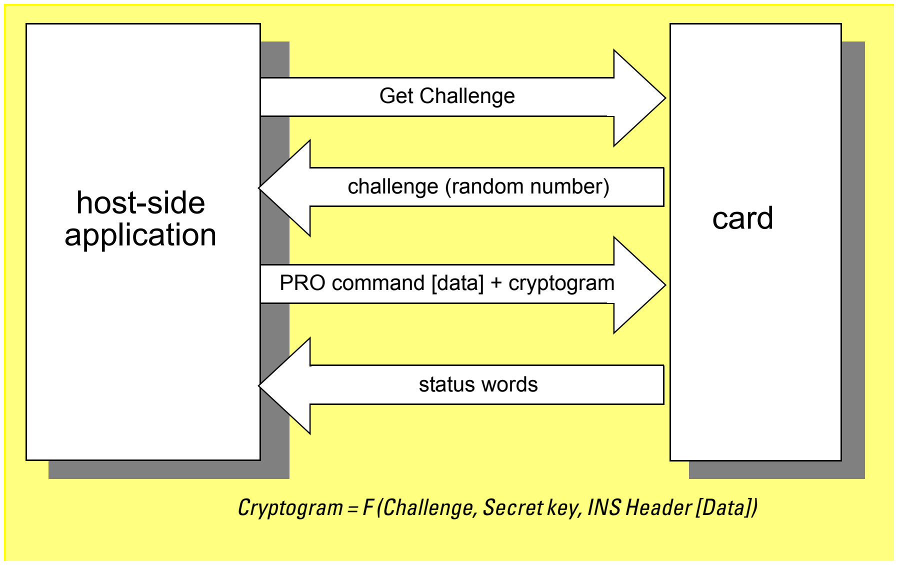
To execute a command with a PRO AC, use the key (whose number is specified in the target file's input parameter string) in the relevant $EF_{Key\ Ext}$.

The following table shows the commands that support the PRO AC.

Command	DF	Transparent EF	Cyclic EF	Linear EF
`Create File`	√			
`Create Record`				√
`Decrease`			√	
`Delete File`	√			
`Dir Next`	√			
`Increase`			√	
`Invalidate`		√	√	√
`Rehabilitate`		√	√	√
`Update Binary`		√		
`Update Record`			√	√

How PRO Authentication Works

The following illustration shows the events in a PRO authentication.


host-side application
card
Get Challenge
challenge (random number)
PRO command [data] + cryptogram
status words
Cryptogram = F (Challenge, Secret key, INS Header [Data])

PRO Authentication Steps

PRO authentication involves the following steps:

1 Calculate the series of input blocks to send:

Combine the components shown in the following table, then split the resulting string into 8-byte blocks. (The maximum number of blocks is 30.)

Command Format: Lengths of Components

Header					Input Data			
CLA	***INS***	***P1***	***P2***	***Lc***	***LOUD***	***message***	***fill block***	***cryptogram***
1 B	1 B	1 B	1 B	1 B	1 B	LOUD bytes	LOFB bytes	6 B

Command Components Converted to Input Data for PRO Calculation

INS	***P1***	***P2***	***LOUD***	***message***	***fill block***
1 B	1 B	1 B	1 B	LOUD bytes	LOFB bytes

To convert the command components into PRO input data, follow these guidelines:

- **Class byte** — Omit the class byte.
- **LOUD byte** — Use the LOUD byte in place of the Lc byte. The maximum amount of LOUD is 240 bytes, which would require 30 DES encryption operations.
- **LOFB byte** — Add LOFB as a 0–7 byte padding string as needed to make the total amount of calculation input data evenly divisible by 8. (LOFB + LOUD + 4 = 8 – 240 bytes octet.)

2. Get a challenge from the card by calling a `Get Challenge` command. (Using a challenge ensures that the cryptogram is unique.)
3. Use the 8-byte challenge and the first 8 bytes from the block in an *exclusive or* (XOR) operation to obtain an 8-byte result. (An illustration of this process appears on page 78.)
4. Choose the appropriate key and use it to encrypt the XOR result.

   Use either a DES or double-length 3DES key in the relevant external key file. The key number is the one you specified for the current command when you created the protected file. (See the command matrix in bytes 14–16 for `Create File` or bytes 12–14 of the response data from a `Dir Next` command). The algorithm ID associated with the key in the external key file identifies whether the key is DES or 3DES.
5. Use the DES output and the next 8 bytes of the input block to calculate the next XOR result.
6. Repeat the previous 2 steps until you have processed all of the 8-byte strings and produced a final 8-byte cryptogram.
7. Truncate the least significant 2 bytes of the result to create a 6-byte cryptogram.

8 Send the command with the 6-byte cryptogram as a trailing string in the following format:

CLA	INS	P1	P2	Lc	Data
1B	1B	1B	1B	1B	LOUD length (1B) + the message, or LOUD bytes (≤ 240 B)+ LOFB bytes (0–7 B) + PRO cryptogram (6 B)

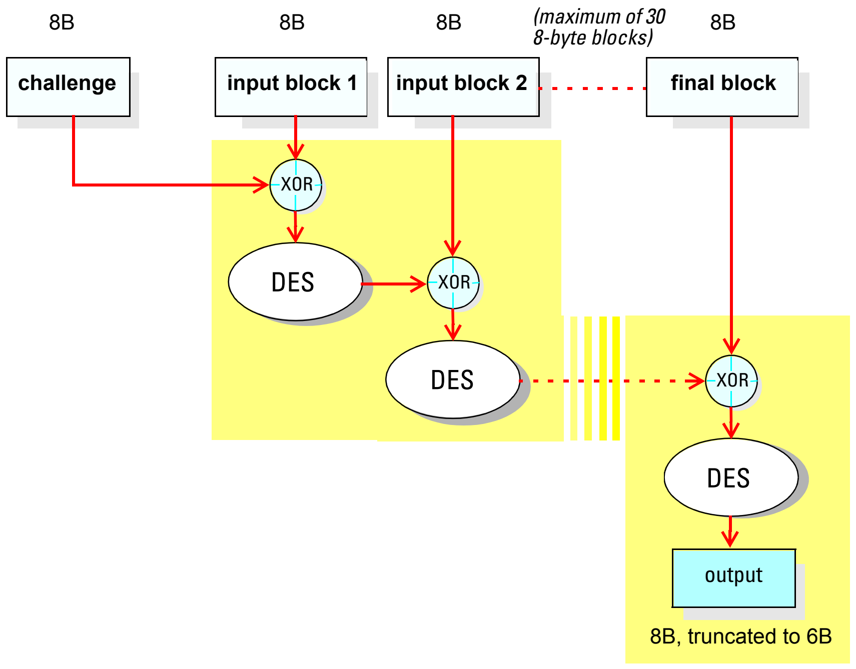
Lc =	Combined length of the data and the PRO cryptogram. The maximum Lc value is 246 bytes (`F0`h)

The card uses the same authentication process to produce a 6-byte cryptogram. If the card's cryptogram matches the one the host sent, the card executes the command.

If the cryptograms do not match, or an error occurs in the DES calculation, the card aborts the command. An unsuccessful authentication attempt decrements the key's attempt counter. If authentication succeeds, the counter is reset to its maximum value.

PRO Calculation Mechanism

The following illustration shows the DES operations in PRO cryptogram creation.



Example of PRO Authentication

The example in this topic shows how to create a file in PRO mode. The example file has the following characteristics:

- Type of file = Transparent EF
- File ID = EF00
- Size of the file = 16 bytes
- ACs set on the file = None

The relevant key used for PRO mode is an external key in the parent DF, which has the value AA AA AA AA AA AA AA AA.

If you did not use PRO mode, you would create the example file by issuing the `Create File` command formatted as shown in the following table.

Command	CLA	INS	P1	P2	Lc	Data	SW
`Create File`	`F0`	`E0`	`00`	`00`	`10`	`FF FF 00 10 EF 00 01 00 00 00 00 01 03 00 00 00`	`9000`

To issue the command in PRO mode, complete these four steps:

Step 1: Call a Get Challenge Command

Begin by issuing a `Get Challenge` command.

Command	CLA	INS	P1	P2	Le	Return Data	SW
`Get Challenge`	`C0`	`84`	`00`	`00`	`08`	`66 1F A3 E7 9D 8F 93 33`	`9000`

Step 2: Use the Challenge to Calculate the PRO Cryptogram

Compute the cryptogram to include with the `Create File` command, by using the format shown in the following table and described on page 75.

Header				Input Data	
INS	**P1**	**P2**	**LOUD**	**Message**	**Fill Block**
E0	00	00	10	FF FF 00 10 EF 00 01 00 00 00 00 01 03 00 00 00	A5 A5 A5 A5

NOTES

- *The fill block values are chosen by random.*
- *As required, the total length of all the bytes shown in the table (including the fill block) is in octet format (evenly divisible by 8).*

Step 3: Use the PRO Calculation Mechanism to Produce the Cryptogram

Use the calculation mechanism shown on page 78 to calculate the 8-byte PRO cryptogram. The resulting cryptogram has the following value:

```
92 88 11 25 1F 2B 6C D6
```

Truncate the cryptogram to 6 bytes—to produce this value:

```
92 88 11 25 1F 2B
```

Step 4: Send the Create File Command in PRO Mode

Call a PRO mode `Create File` command by using the following format.

Header					Input Data				SW
CLA	**INS**	**P1**	**P2**	**Lc**	**LOUD**	**Message**	**Fill Block**	**Cryptogram**	
F0	E0	00	00	1B	10	FF FF 00 10 EF 00 01 00 00 00 00 01 03 00 00 00	A5 A5 A5 A5	FF FF 00 10 EF 00	9000

AUT Access Condition

AC / Descriptive name – AUT / Authenticate

Hex value – `4`

You can execute the command only after the specified key in the relevant external key file is presented correctly in an `External Authenticate` or `Verify Key` call. The verification can occur immediately before you issue the command or earlier in the card session. Once established, the AUT AC persists for the life of the card session or until an event occurs that terminates it (as described on page 67).

The AUT access condition requires the host application to authenticate itself to the card. The process begins with a `Get Challenge` command. The host encrypts and returns the card's challenge. The card then uses the specified key and algorithm ID in the relevant external key file to verify the host cryptogram. If the verification succeeds, the AUT AC is satisfied.

Each failed authentication attempt decrements the key attempt counter. If the authentication is successful, the attempt counter is reset to its pre-set attempt value. If the counter is decremented until it reaches the null value, the key is blocked. The user assigned to the blocked key (for example, the cardholder) can no longer perform any actions that require the key. Only an administrator with the appropriate access rights can unblock the key—by calling an `Update Binary` command and overwriting the blocked key value and counter for remaining attempts.

If you execute a command that has an AUT AC, be aware of which EF$_{Key\ Ext}$ applies to the target file, and use the relevant key.

CHV1 and PRO Access Condition

AC / Descriptive name – CHV1 and PRO / CardHolder Verified 1 and Protected

Hex value – 6

This access condition requires both CHV1 and PRO authentication:

- You must verify CHV1 before you issue the command. (CHV1 is typically the cardholder's PIN). *(For more information about CHV1, see page 72.)*
- You must append a verified PRO authentication digital signature to the command. *(For more information about PRO authentication, see page 73.)*

If you execute a command that has a CHV1 and PRO AC, be aware of which $EF_{Key\ Ext}$ and EF_{CHV1} apply to the target file, and use the relevant key and PIN.

CHV2 and PRO Access Condition

AC / Descriptive name – CHV2 and PRO / CardHolder Verified 2 and Protected

Hex value – 7

This access condition requires both CHV2 and PRO authentication:

- You must verify CHV2 before you issue the command. (CHV2 is typically the card administrator's PIN). *(For more information about CHV2, see page 73.)*
- You must append a verified PRO authentication digital signature to the command. *(For more information about PRO authentication, see page 73.)*

If you execute a command that has a CHV2 and PRO AC, be aware of which $EF_{Key\ Ext}$ and EF_{CHV2} apply to the target file, and use the relevant key and PIN.

CHV1 and AUT Access Condition

AC / Descriptive name – CHV1 and AUT / CardHolder Verified 1 and Authenticated

Hex value – 8

This access condition requires both CHV1 and AUT authentication:

- You must verify CHV1 before you issue the command. (CHV1 is typically the cardholder's PIN). *(For more information about CHV1, see page 72.)*
- You must satisfy the AUT access condition before you issue the command: Send a successful `Verify Key` or `External Authenticate Using DES` command to verify that the host-side application is authentic. Once the AUT AC is satisfied, you can issue AUT-protected commands repeatedly.

If you execute a command that has a CHV1 and AUT AC, be aware of which $EF_{Key\ Ext}$ and EF_{CHV1} apply to the target file, and use the relevant key and PIN.

CHV2 and AUT Access Condition

AC / Descriptive name – CHV2 and AUT / CardHolder Verified 2 and Authenticated

Hex value – 9

This access condition requires both CHV2 and AUT authentication:

- You must verify CHV2 before you issue the command. (CHV2 is typically the card administrator's PIN). *(For more information about CHV2, see page 73.)*
- You must satisfy the AUT AC before you issue the command: Send a successful `Verify Key` or `External Authenticate Using DES` command to verify that the host-side application is authentic. Once the AUT AC is satisfied, you can issue AUT-protected commands repeatedly.

If you execute a command that has a CHV1 and AUT AC, be aware of which $EF_{Key\ Ext}$ and EF_{CHV2} apply to the target file, and use the relevant key and PIN.

NEVer Access Condition

AC / Descriptive name – NEV / Never

Hex value – `F`

The action is never possible.

Overview of Cryptographic Security

Encryption and Decryption

In symmetric key encryption, the sender and recipient have the same DES or 3DES key. The sender uses the key to encrypt data, which is sent to the recipient as a cryptogram. The recipient may use the DES or 3DES key to decrypt the cryptogram or to perform a parallel process on the original data and compare the results.

In asymmetric key encryption, different keys are used for encryption and decryption. When the sender wants to send a secure message to the recipient, the sender looks up the recipient’s public key in a directory and uses it to encrypt the message. The recipient uses the private key to decrypt the encrypted message and read it. Anyone can send an encrypted message to the recipient, but only the recipient has the private key to decrypt the message.

Digital Signatures

You can also use symmetric cryptography to digitally sign data to verify that the data has not been altered.

To digitally sign a message or data, the data does not have to be encrypted. You create a digital signature by dividing the data into blocks and computing a hash. You encrypt the digest with an RSA private key. The encrypted hash is the digital signature, which you attach to the data.

To create digital signatures and hashes, you can use these card commands:

- `SHA-1 Intermediate` and `SHA-1 Last` *(hash data)*
- `RSA Signature (Internal Auth)` *(encrypt data with a 1024-bit or weaker key)*
- `RSA Signature Intermediate` *(use with an `RSA Signature Last` command to encrypt data with a 2048-bit or weaker key)*
- `RSA Signature Last` *(use with `RSA Signature Intermediate` to encrypt data with a 2048-bit or weaker key)*

Authentication

Authentication is the process of establishing the identity of the card, the host application or both parties. You can use three types of authentication:

- **Internal authentication** — The card proves its identity to the host.
- **External authentication** — The host proves its identity to the card. External authentication is often required before the card will execute sensitive commands.
- **Mutual authentication** — The process of establishing mutual trust between a terminal and a card before transactions begin. Mutual authentication consists of internal and external authentication.

To perform an external authentication, call the `Get Challenge` command, then the `External Authenticate Using DES` or `Verify Key` command. To perform internal authentication, call the `Internal Authenticate Using DES` command.

Key Strength

The strength of a key indicates how difficult it is to break. The strength of a given key type is typically related to its length.

Single-length DES keys are 64 bits long, and double-length 3DES keys are 128 bits long. DES keys actually use only 56 of the 64 bits, and double-length 3DES keys use 112 of the 128 bits.

On Cryptoflex cards, RSA key formats are designated as 512-bit, 768-bit, 1024-bit, and 2048-bit. The key format designation is taken from the length of the RSA public key's public modulus.

The longer the key, the more secure it is likely to be, and the longer the key takes to generate or use for encryption and decryption. The current recommended maximum RSA key format is 1024-bit, which takes one second or less to execute.

Key Padding

A number of commands require a specific input data length or format, such as octet data input (data you can divide evenly into 8-byte blocks). If the input data does not meet these requirements, you must add padding bytes at the end of the data string to standardize its length. For more information, see specific command descriptions starting on page 87.

Certificates

RSA keys are typically certified by a third-party certification authority (CA). Cryptoflex cards provide transparent elementary files to store certificates on the card.

4

Cryptoflex Card Commands

The Cryptoflex card operating system has two types of commands — commands for managing files and commands for security operations.

File Management Commands

Use the file management commands to create, delete, organize, modify, and search elementary files (EFs) and directories (or dedicated files, DFs). You can perform the following tasks:

- `Change CHV` – Change the PIN value in the relevant CHV key file.
- `Create File` – Create a new EF or DF under the currently selected DF.
- `Create Record` – Write a new record at the end of a linear EF.
- `Decrease` – Decrease the value stored in a cyclic EF record.
- `Delete File` – Delete a file under the currently selected directory.
- `Dir Next` – Retrieve input parameter string data for the files in a directory—file by file and in order of creation.
- `External Authenticate Using DES` – Establish the host application's right to interact with the card.
- `Get Response` – Retrieve data the previous command generated or captured, such as file data, record data, or a cryptogram.

NOTE *See command description for limitations when used with the 32K+e-gate card or the Cryptoflex 32K card version 1.*

- `Increase` – Increase the value stored in a cyclic EF record.

- `Invalidate` – Make an EF inaccessible. (See `Rehabilitate`.)
- `Read Binary` – Retrieve data as plaintext from a transparent EF. (Compare with `Read Binary Enciphered`.)
- `Read Record` – Retrieve record data from a linear or cyclic EF.
- `Read Record EMV` – With an EMV application selected, read record data in a EMV linear EF.

NOTE *The Read Record EMV command is not available for the 32K+e-gate card or the Cryptoflex 32K card version 1.*

- `Rehabilitate` – Reactivate an invalidated file. (See `Invalidate`.)
- `Seek` – Search for occurrences of a specified string in linear EF records.
- `Select` – Select an EF or DF by its file ID. Many commands are context-specific, and you frequently use the `Select` command to navigate through the card's file system before you call other commands.
- `Select EMV` – Instruct the card to send the commands that follow to an EMV application.

NOTE *The Select EMV command is not available for the 32K+e-gate card or the Cryptoflex 32K card version 1.*

- `Unblock CHV` – Regain access to card files protected by a blocked CHV (that is, a CHV blocked when a user repeatedly enters the PIN incorrectly).
- `Update Binary` – Send plaintext data to the card to update data in a transparent EF. (Compare with `Update Binary Enciphered`.)
- `Update Record` – Write new data into a linear or cyclic EF record.
- `Verify CHV` – Establish the current card user's access condition (AC) by verifying the user's PIN.
- `Verify Key` – Establish the host application user's AC by verifying a key stored on the card (such as the application authorization key, or AAK).

Cryptographic Commands

Cryptographic commands execute operations to conduct secure transactions, such as electronic signatures, user and card authentication, key generation, and hashing. Use the cryptographic commands to perform the following tasks:

- `DES Block Init` – Encrypt or decrypt a sole or initial block of data by using a Data Encryption Standard (DES) or double-length 3DES key in cipher block chaining (CBC) mode. (Follow with one or more `DES Block` calls if the total amount of data exceeds 232 bytes.)
- `DES Block` – Complete the encryption or decryption started by a `DES Block Init` command if the total amount of data exceeds 232 bytes.
- `Generate DES Key` – Create and store a DES or double-length 3DES key. (Both operations take place on the card.) You can use this command to generate keys for internal authentication and other DES operations.
- `Generate RSA Keys` – Create and automatically store a private and public RSA key on the card.

  32K+SS V1 32K+e-gate only

  Three public key formats are available when you use the Generate RSA Keys command.
- `Get AC Keys` – Retrieve the key number needed to satisfy the access conditions (ACs) for AUT-protected or PRO mode commands in a given context.
- `Get Challenge` – Ask the card for a challenge (a random number to encrypt) to use for operations such as external authentication.
- `Internal Authenticate Using DES` – Establish whether the card sis authorized to interact with the host application.
- `Logout AC` – Selectively revoke one or more of the currently logged-in ACs. Use this command to avoid accumulating user access rights without resetting the card.
- `Read Binary Enciphered` – Encrypt and retrieve data from a transparent EF.
- `RSA Signature (Internal Auth)` – Create an RSA digital signature on the card by using a 1024-bit or smaller RSA key. The computation result is *not* physically stored in EEPROM. An internal RSA signature verification is performed when the `RSA Signature (Internal Auth)` command is run to ensure the coherence of the signature.

- `RSA Signature Intermediate` – Perform the first or intermediate step (before `RSA Signature Last`) to store an RSA digital signature on the card by using a 2048-bit RSA key.
- `RSA Signature Last` – Perform the final or only step to store an RSA digital signature on the card by using a 2048-bit RSA key. (To use a 2048-bit key, you must use this command with `RSA Signature Intermediate`.) An internal RSA signature verification is performed when the `RSA Signature Last` command is run to ensure the coherence of the signature.
- `SHA-1 Intermediate` – Begin or continue a Secure Hash Algorithm (SHA-1) operation to create a hash digest of data.
- `SHA-1 Last` – Conduct SHA-1 hashing on a final (or only) data block of 64 bytes or less. (To hash more than 64 bytes of data, use this command with `SHA-1 Intermediate`.)
- `Update Binary Enciphered` – Send DES-encrypted data to the card to overwrite values in a transparent EF.

Cryptoflex Card Command Summary

The table below summarizes information about the Cryptoflex card's operating system commands. (Table number values are always shown in hexadecimal format unless otherwise noted.)

Cryptoflex Card Operating System Commands

Command	Cla	Ins	P1	P2	Lc	Le	S / R	AC
`Change CHV`	`F0`	`24`	`00`	*CHV nbr*	`10`		S	*NA*
`Create File`	`F0`	`E0`	*init*	*NR*	*lgth + X* [1]		S	*MF=AUT/per DF*
`Create Record`	`C0`	`E2`	`00`	`00`	*lgth + X* [1]		S	*per linear EF*
`Decrease`	`F0`	`30`	`00`	`00`	`03` + *X* [1]		S/R	*per cyclic EF*
`Delete File`	`F0`	`E4`	`00`	`00`	`02` + *X* [1]		S	*MF=AUT/per DF*
`DES Block`	`F0`	`58`	*mode*	*key nbr*	`08–E8`		S/R	*CHV1*
`DES Block Init`	`F0`	`56`	*mode*	*key nbr*	`08–E8`		S/R	*CHV1*
`Dir Next`	`F0`	`A8`	`00`	`00`	*lgth + X* [1]		R	*MF=AUT/per DF*
`External Authenticate Using DES`	`C0`	`82`	`00`	`00`	`07`		S	*NA*
`Generate DES Key`	`F0`	`50`	`00`	*key nbr*	`00`		S	*CHV1*
`Generate RSA Keys`[2]	`F0`	`46`	*key nbr*	`00/40/60/80`	`04`		S	*CHV1*
`Get AC Keys`	`F0`	`C4`	`00`	`00`	`03`		S	*CHV1*
`Get Challenge`	`C0`	`84`	`00`	`00`		*lgth*	R	*NA*
`Get Response`[3] *(`C0` CLA)*	`C0`	`C0`	`00`	`00`		*lgth*	R	*NA*
(`F0` CLA)	`F0`	`C0`	`00`	`00`		*lgth*	R	*NA*
(`00` CLA)	`00`	`C0`	`00`	`00`		*lgth*	R	*NA*
`Increase`	`F0`	`32`	`00`	`00`	`03` + *X* [1]		S/R	*per cyclic EF*
`Internal Authenticate Using DES`	`C0`	`88`	`00`	*key nbr*	`08`		S/R	*CHV1*
`Invalidate`	`F0`	`04`	`00`	`00`	`00` + *X* [1]		S	*MF=AUT/per EF*
`Logout AC`	`F0`	`22`	*AC*	`00`	`00`		S	*NA*
`Read Binary`	`C0`	`B0`	*off MSB*	*offset LSB*	*lgth*		R	*per EF, no PRO*
`Read Binary Enciphered`	`04`	`B0`	*off MSB*	*offset LSB*	`08–E8`		R	*per EF, no PRO*
`Read Record`	`C0`	`B2`	*rec nbr*	*mode*		*lgth*	R	*per EF, no PRO*

Command	**Cla**	**Ins**	**P1**	**P2**	**Lc**	**Le**	**S / R**	**AC**
`Read Record EMV`[4]	C0	B2	*rec nbr*	*ref ctrl*		*lgth*	R	*NA*
`Rehabilitate`	F0	44	00	00	00 + X^1		S	*MF=AUT/per EF*
`RSA Signature (Internal Auth)`	C0	88	00	*key nbr*	40/60/80		S/R	*CHV1*
`RSA Signature Intermediate`	10	88	00	*key nbr*	*lgth*		S	*CHV1*
`RSA Signature Last`	00	88	00	*key nbr*	*lgth*		S/R	*CHV1*
`Seek`	F0	A2	*offset*	*mode*	*lgth*		S	*per EF, no PRO*
`Select`	C0	A4	00	00	02		S/R	*NA*
`Select EMV`[4]	00	A4	04	00	05–10		S/R	*NA*
`SHA-1 Intermediate`	14	40	00	00	40		S	*NA*
`SHA-1 Intermediate (ISO-2)`	10	40	00	00	40		S	*NA*
`SHA-1 Last`	00	40	00	00	08–40		S/R	*NA*
`SHA-1 Last (ISO-3)`	04	40	00	00	08–40		S/R	*NA*
`Unblock CHV`	F0	2C	00	*CHV #*	10		S	*NA*
`Update Binary`	C0	D6	*off MSB*	*offset LSB*	*lgth* + X^1		S	*per EF*
`Update Binary Enciphered`	04	D6	*off MSB*	*offset LSB*	08–E8		S	*per EF*
`Update Record`	C0	DC	*rec nbr*	*mode*	*lgth* + X^1		S	*per EF*
`Verify CHV`	C0	20	00	*CHV nbr*	08		S	*NA*
`Verify Key`	F0	2A	00	*key nbr*	08/0F *lgth*		S	*NA*

1 For commands with protected access, *X* is the length of the cryptogram if the access condition is PRO. If no PRO AC applies, *X* = 0.

2 Enhanced when using the 32K+SS V1 or 32K+e-gate card.

3 Limitations on command when used with the 32K+e-gate card or the Cryptoflex 32K card version 1.

4 Command not available for the 32K+e-gate card or the Cryptoflex 32K card version 1.

NOTE *For information about the S/R column values, see the following topics:*

- —: "Case 1: No Input or Output," on page 234
- R: "Case 2: Receive Mode," on page 235
- S: "Case 3: Send Mode," on page 236
- S/R: "Case 4: Send/Receive Mode," on page 237 *(To receive the available data for an S/R command, send a follow-up* `Get Response` *command.)*

Change CHV

Use the `Change CHV` command to change a CHV key (PIN) in the relevant CHV file and verify the corresponding user AC. The relevant CHV file is the one that applies to the current context—the CHV file that protects the currently selected EF or DF. A card can have more than one relevant CHV file—for example, one for a card administrator and another for a cardholder. You can use the `Change CHV` command to change the value for either CHV file's PIN.

S/R: Send *(Case 3)*

AC – Not applicable

To execute this command successfully, the currently selected file must have a relevant CHV key that is not blocked and whose CHV file is not invalidated.

PINs are 8 bytes long, and can include numbers and any standard ASCII characters. (Use hexadecimal values for each number and character.) For a shorter PIN, add padding bytes after the PIN value. To change the PIN, include old and new PIN numbers together as a 16-byte data string—old key first.

If you change the PIN successfully, the card grants you access to the protected files and resets the CHV attempt counter to the default pre-set value.

If you submit an incorrect value for the current PIN, the card decrements the attempt counter. If all the attempts are exhausted (the counter reaches null), the CHV key is blocked. Commands protected by a blocked CHV key are inaccessible. Use the `Unblock CHV` command to unblock a CHV key. You may also be able to verify a different (unblocked) AC and call an `Update Binary` command to overwrite the remaining attempt counter and PIN value in the CHV file.

Command Format Navigate to the directory that contains the CHV file you want to update and call the `Change CHV` file in the following format.

Command	CLA	INS	P1	P2	Lc
`Change CHV`	`F0`	`24`	`00`	*CHV nbr*	`10`

Parameters

Name	Length	Value / Meaning
P1	1 B	`00h`
P2	1 B	CHV type: • `01h` = CHV1 PIN (in the relevant EF$_{CHV1}$, file ID 0000), or • `02h` = CHV2 PIN (in the relevant EF$_{CHV1}$, file ID 0100).
Lc	1 B	`10h` – Length of the input key data (old and new PIN values). PINs are 8 bytes long. Always enter `10h`, regardless of PIN length. (For a shorter PIN, apply padding.)
Input Data	16 B	Bytes 1–8 = Current PIN value, and Bytes 9–16 = Updated PIN value.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6300`	The current PIN value you entered as input data is incorrect.
`6581`	Memory-related problem: The EEPROM may have failed.
`6710`	Incorrect value entered for Lc. Enter `10h`.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	Either no relevant CHV file (of the type specified in P2) exists, or the file does not contain a CHV key.
`6983`	The CHV key is blocked; no more attempts are possible.
`6B00`	Unsupported values entered for P1, P2, or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The specified PIN is updated.

See Also:

- `Unblock CHV` command on page 188
- "Cardholder Verification Files (CHV1 and CHV2)," on page 21

Create File

Use the `Create File` command to create a new file under the currently selected directory. You can use this command to create either a new directory file (DF) or elementary file (EF).

S/R: Send *(Case 3)*

AC – To create a file under the master file (MF), you must first satisfy the AUT access condition. To create a file under a DF other than the MF, you must first satisfy the AC for the selected DF. (To find out which key number is required to satisfy an AUT or PRO AC, call the `Get AC Keys` command, described on page 133. To find out which AC level is required, call a `Select` command followed by a `Get Response` command.)

To create a file, first select the directory that will contain the file. Specify the new file's input parameter string and body structure in the command data block. You can create a file successfully only if the parent directory has enough space to hold the new file and its input parameter string. (For format descriptions of the various file types, see "File Types," on page 4.)

If you create a DF it automatically becomes the currently selected directory. If you create an EF it becomes the currently selected file.

Do not use a reserved file ID unless the file you create is appropriate for that ID. The reserved file IDs are: 0000, 0001, 0002, 0011, 0012, 0100, 1012, 2F01, 3F00, 3FFF, FFFF, FFxx, and xxFF (where xx is a placeholder for any value). If you misuse a reserved ID, you may encounter unexpected card behavior, or your card may become fatally deadlocked.

SchlumbergerSema recommends that you limit the number of directory levels to make it easier to navigate to all the files on the card. Two levels of directories will accommodate most applications.

Command Format

Select the directory that you want to contain the new file and call the `Create File` command with the following format.

Command	CLA	INS	P1	P2	Lc
`Create File`	`F0`	`E0`	*init*	*NR*	*lgth* + *X*

Parameters

Name	Length	Value / Meaning
P1	1 B	File body initialization: • `00`h — Initialize the file body with null values. • `FF`h — Do not initialize the file body. *(Note that the `Create File` command initializes a cyclic EF body to `00`h by default, regardless of the P1 setting.)*
P2	1 B	Number of records to be created: `00`h — Creates no records. `01–FF`h — Specifies a number of records to create (1–255). *DF or transparent EF:* Enter `00`h (no records). *Cyclic EF:* Specify the total number of records, which (with the record headers) must fill the available file space completely. *Fixed-length linear EF:* Specify records to fill all or part of the available file space. If the specified records fill only part of the file space, you may be able to add more records later. (For more information, see the `Create Record` command on page 105.) *Variable-length linear EF:* Specify one or more records (of the length specified in byte 17). You have the option to leave space at the end of the file to add records of other lengths later (by calling the `Create Record` command). The maximum number of records you can create is 255.
Lc	1 B	Length of the file structure data to send as input: *DF, transparent EF, or variable-length EF* = `10`*h* + *X*: No record envelope size. (*X* =Cryptogram length if AC = PRO, or 0 if no PRO AC.) *Fixed-length EF or cyclic EF* = `11`*h* + *X*: Record envelope matches records specified at file creation. Records can be no more than 255 bytes long. (*X* =Cryptogram length if AC = PRO, or 0 if no PRO AC.)

Name	Length	Value / Meaning
Input Data	*lgth* + *X*	Data for the file structure + the cryptogram (if AC = PRO) (See the Input Data table that follows.)
SW1, SW2	2 B	Status word bytes the card returns.

Input Data

Byte(s)	Data Description	Length
1–2	RFU (enter `FF FF`)	2 B
3–4	File size (See page 98.)	2 B
5–6	File ID	2 B
7	File type: • *Transparent EF* = `01`h • *Fixed-length linear EF* = `02`h • *Variable-length linear EF* = `04`h • *Cyclic EF* = `06`h • *Dedicated file* = `38`h	1 B
8–11	Access conditions (See page 101.)	4 B
12	File validation status: • `01`h = activated • `00`h = invalidated	1 B
13	Length of the input data from byte 14 to EOF: • *Transparent EF* = `03`h • *Fixed-length linear EF* = `04`h • *Variable-length linear EF* = `03`h • *Cyclic EF* = `04`h • *Dedicated file* = `03`h	1 B
14–16	Key numbers in the relevant external key file to use for granting access to file commands	3 B

Byte(s)	Data Description	Length
17	Record length: • *Transparent EF* = Omit • *Fixed-length linear EF* = Record length • *Variable-length linear EF* = Omit • *Cyclic EF* = Record length • *Dedicated file* = Omit	1 B
18+	For a PRO command, add the cryptogram.	x B

Setting the File Size

Bytes 3–4

Set the file size to accommodate the file's input parameter string and data container. Remember that you cannot change the file size later. When you set the file size, take these factors into consideration:

Dedicated file (directory) — Set the directory size to equal or exceed the amount of space needed for:

- The input parameter strings for the files in the directory (16 bytes for EFs and 24 bytes for DFs)
- All the files (DFs and EFs) the directory will contain

Transparent EF — Set the file size to equal or exceed the amount of space needed for the data container, which is an undivided envelope that must be large enough to accommodate the data you plan to store in it

Cyclic EF — The file size you specify must exactly equal the number of records multiplied by the record length. The card adds space for record headers automatically. Do not include the 4-byte record headers in the calculation.

Fixed-length linear EF — It is best to specify all record envelopes at file creation. Make the file large enough to accommodate the records. The card adds space for the 4-byte record headers automatically, so you do not need to include record headers in your calculation.

Variable-length linear EF — Set the file size to equal or exceed the amount of space needed for all the record envelopes — the ones you specify at file creation and the ones you will add later. The card adds space for the 4-byte record headers automatically, so you do not need to include record headers in your calculation.

NOTE *Record files (linear or cyclic EFs) can contain a maximum of 255 records, each of which is limited to a maximum length of 255 bytes.*

Setting Access Conditions

This topic describes how to set a new file's access conditions (ACs), activation status, and key numbers needed to satisfy the ACs.

Setting Access Conditions for a Dedicated File

If you create a new DF, use the following specifications for bytes 8 through 11 to set the file's ACs, activation status, and AC key numbers.

Byte 8

For a DF, byte 8 is reserved for future use (RFU). Any value you enter is ignored.

Bytes 9–11

Bytes 9 through 11 specify ACs for the DF. This 6-nibble command matrix holds the hexadecimal values (described on page 100) that determine which (if any) ACs must be satisfied in order to execute the corresponding commands on files under the new DF. The following table shows the command matrix.

DF Commands Subject to Access Conditions

Byte	Most Significant Nibble (MSN)	Least Significant Nibble (LSN)
9	`Dir Next`	RFU
10	`Delete File`	`Create File`
11	RFU	RFU

Use the values in the following table to set ACs for either DFs or EFs. (For more information about access conditions, see "Setting Access Rights on Card Operations," on page 62.)

Hexadecimal Values for Setting Access Conditions

Value	AC	Description of Restriction
`0h`	ALW	Always: It is always possible to perform the operation.
`1h`	CHV1	CHV1 verification must have been established (in a `Verify CHV` command) and remain in force.
`2h`	CHV2	CHV2 verification must have been established (in a `Verify CHV` command) and remain in force.
`3h`	PRO	Protected mode: The command must be signed by a verified PRO authentication digital signature.
`4h`	AUT	AUT must have been established (in a `Verify Key` or `External Authenticate Using DES` command) and remain in force.
`5h`	RFU	Reserved for future use.
`6h`	CHV1 & PRO	CHV1 verification must have been established and remain in force, and the command must be signed by a verified PRO authentication digital signature.
`7h`	CHV2 & PRO	CHV2 verification must have been established and remain in force, and the command must be signed by a verified PRO authentication digital signature.
`8h`	CHV1 & AUT	Both CHV1 and AUT must have been established and remain in force.
`9h`	CHV2 & AUT	Both CHV2 and AUT must have been established and remain in force.
`Ah`	RFU	Reserved for future use.
`Bh`	RFU	Reserved for future use.
`Ch`	RFU	Reserved for future use.
`Dh`	RFU	Reserved for future use.
`Eh`	RFU	Reserved for future use.
`Fh`	NEV	Never: The operation is never possible.

Byte 12

The LSN of byte 12 indicates the file status. A value of 0 means the file is invalidated. A value of 1 means the file is activated and available. Unlike an EF, you cannot change the status of a DF after you create it.

Bytes 14–16

For each command you specified as protected by a AUT or PRO AC in bytes 9 through 11, enter a key number value in bytes 14 through 16. The key number specifies the key in the relevant external key file that must be used to satisfy the AUT or PRO AC. The byte 14–16 matrix matches the byte 9–11 matrix.

Key Numbers for the DF Commands Subject to Access Conditions

Byte	Most Significant Nibble (MSN)	Least Significant Nibble (LSN)
14	`Dir Next`	RFU
15	`Delete File`	`Create File`
16	RFU	RFU

Setting Access Conditions for an Elementary File

If you create a new EF, use the following specifications for bytes 8–11 to set the file's ACs, activation status, and AC key numbers.

Byte 8: Defining the Availability of the Increase and Decrease Commands

Use byte 8 to enable or restrict the use of the `Increase` and `Decrease` commands for the EF, as shown in the following table. (Note that bit 8 is the most significant nibble, MSN of the byte.) Only the values of bits 8 (`Decrease`) and bit 7 (`Increase`) affect command availability. The values of bits 1–6 are ignored.

Command Restrictions in Byte 8

8	7	6	5	4	3	2	1		Value	Command Restrictions
0	0	x	x	x	x	x	x	=	`00–3Fh`	No `Decrease` or `Increase` permitted
0	1	x	x	x	x	x	x	=	`40–7Fh`	`Decrease` not permitted
1	0	x	x	x	x	x	x	=	`80–BFh`	`Increase` not permitted
1	1	x	x	x	x	x	x	=	`C0–FFh`	No restrictions

Bytes 9–11: Defining an EF's Access Conditions

Bytes 9 through 11 specify ACs for the new EF. The command matrix in these six nibbles varies according to the type of EF you create. If you want to execute one of these commands on the file later, you must satisfy the AC specified in the command's corresponding nibble. (For information about the hexadecimal values you use to set ACs, see the table on page 100.)

Transparent EF: Commands Subject to Access Conditions

Byte	MSN (holds AC data for these commands)	LSN (holds AC data for these commands)
9	`Read Binary`	`Update Binary`
10	`Read Binary Enciphered`	`Update Binary Enciphered`
11	`Rehabilitate`	`Invalidate`

Cyclic EF: Commands Subject to Access Conditions

Byte	MSN (holds AC data for these commands)	LSN (holds AC data for these commands)
9	`Read Record`	`Decrease`[1]
10	`Increase`[1]	RFU
11	`Rehabilitate`	`Invalidate`

1 Available only if enabled in byte 8, as described on page 101.

Linear EF: Commands Subject to Access Conditions

Byte	MSN (holds AC data for these commands)	LSN (holds AC data for these commands)
9	`Seek`, `Read Record`	`Update Record`
10	RFU	`Create Record`
11	`Rehabilitate`	`Invalidate`

Byte 12: Specifying the File Validation Status

The LSN of byte 12 specifies the file validation status. A value of `0`h invalidates the file. A value of `1`h means the file is activated and available. An application typically cannot execute any command other than `Select`, `Delete File`, or `Rehabilitate` on an invalidated file.

Bytes 14–16: Specifying the Key Numbers for Satisfying Access Conditions

For each command you specified as protected by an AC in bytes 9 through 11, enter a key number value in bytes 14 through 16. For a PRO or AUT AC, the key number refers to the relevant external key file. The nibbles of bytes 14–16 comprise a command matrix that matches the byte 9–11 matrix, as shown in the following table.

NOTE *Numbers of keys in the external key file are zero-based. (For example, key 0 occupies key slot 1.)*

Transparent EF: Key Numbers for Commands Subject to Access Conditions

Byte	MSN (holds AC data for these commands)	LSN (holds AC data for these commands)
14	`Read Binary`	`Update Binary`
15	`Read Binary Enciphered`	`Update Binary Enciphered`
16	`Rehabilitate`	`Invalidate`

Cyclic EF: Key Numbers for Commands Subject to Access Conditions

Byte	MSN (holds AC data for these commands)	LSN (holds AC data for these commands)
14	`Read Record`	`Decrease`[1]
15	`Increase`[1]	RFU
16	`Rehabilitate`	`Invalidate`

1 Available only if enabled in byte 8, as described on page 101.

Linear EF: Key Numbers for Commands Subject to Access Conditions

Byte	MSN (holds AC data for these commands)	LSN (holds AC data for these commands)
14	`Seek, Read Record`	`Update Record`
15	RFU	`Create Record`
16	`Rehabilitate`	`Invalidate`

Status Words Returned

Hex Value	Meaning
`6283`	The file cannot be created because the parent DF is invalidated.
`6300`	PRO authentication failed because the cryptogram is wrong.
`6581`	Memory-related problem: The EEPROM may have failed.
`67`*xx*	The value entered for Lc does not match the data block or is an unsupported length. Enter the value that appears in place of *xx*.
`6CYYh` `(YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	The AC cannot be satisfied because either the cryptographic key, CHV key, or key file is missing.
`6982`	The required AC was not satisfied because the key is invalid or no key was presented.
`6985`	The AUT AC was not satisfied because the host did not send a `Get Challenge` command to the card.
`6A80`	One of these problems occurred: • A file with the specified ID already exists in the parent DF. • The currently selected file type is inconsistent with the command. To create a DF, first select the most recently created DF. To create an EF, you must first select the parent DF. • The record length is null.
`6A83`	The current DF already contains the maximum number of files (255).
`6A84`	Insufficient EEPROM space is available.
`6B00`	Incorrect values were entered for P1, P2, or both. For example, the amount of EEPROM for the specified records exceeds the file size (linear EF), or does not exactly match the file size (cyclic EF.)
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The card created the specified file.

See Also:

- `Dir Next` command on page 118
- `Get AC Keys` command on page 133
- `Select` command on page 177
- "Commands That Are Subject to Access Conditions," on page 64

Create Record

Use the `Create Record` command to write a new record at the logical end of a fixed-length or a variable-length linear EF.

S/R: Send *(Case 3)*

AC – To create a record in a linear EF, you must first satisfy the access condition specified for the `Create Record` command in that file's input parameter string. (To find out which key number is required to satisfy an AUT or PRO AC, call the `Get AC Keys` command, described on page 133.)

You can add a record if sufficient space remains in the file, and if the selected file contains less than the maximum number of records (255). If enough space is available, the new record can be a maximum of 255 bytes long. The new record becomes the currently selected record.

For a fixed-length linear EF, the size of the record envelope is specified when the file is created. New records can be the original length or can be shorter. If you add a shorter record, the card fills the unused part of the record envelope with null values.

Command Format

Select a linear EF to contain the new record and call the `Create Record` command in the following format.

Command	CLA	INS	P1	P2	Lc
`Create Record`	`C0`	`E2`	`00`	`00`	*lgth* + *X*

Parameters

Name	Length	Value / Meaning
P1	1 B	`00`h
P2	1 B	`00`h
Lc	1 B	Length of the data input data: The data to write (*lgth*) + cryptogram (if the PRO mode applies). The *lgth* value equals the record length.
Input Data	*lgth* + *X*	The new record contents + the cryptogram (if AC = PRO). *lgth* = Length of the data (the record length). *X* =Cryptogram length if AC = PRO, or 0 if no PRO AC.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6283`	The currently selected EF is invalidated.
`6300`	PRO authentication failed because the cryptogram is wrong.
`6581`	Memory-related problem: The EEPROM may have failed.
`6700`	The Lc value is null.
`67`*xx*	The Lc value is too long. The correct Lc entry appears in place of *xx*.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	The AC cannot be satisfied because either the cryptographic key, CHV key, or key file is missing.
`6982`	The required AC was not satisfied because the key is invalid or no key was presented.
`6983`	The key required for PRO authentication is blocked.
`6985`	The AUT AC was not satisfied because the host did not send a `Get Challenge` command to the card.
`6986`	A DF is currently selected. (Select a linear EF.)
`6A80`	A cyclic or transparent EF is currently selected. (Select a linear EF.)
`6A83`	The selected file is full: there are no more available record spaces. (You can use the `Update Record` command to overwrite an existing record.)
`6A84`	Insufficient EEPROM space is available to write the new record in the currently selected EF.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The card created and selected the new record.

See Also:

- `Read Record` command on page 157
- `Update Record` command on page 195
- `Seek` command on page 175
- “Commands That Are Subject to Access Conditions,” on page 64

Decrease

Use the `Decrease` command to decrease the value stored in the currently selected cyclic EF. For example, you can use the `Decrease` command to debit an electronic purse.

S/R: Send/Receive *(Case 4)*

AC – To use this command successfully, you must first satisfy the access condition specified for the `Decrease` command in the selected cyclic EF's input parameter string. (To find out which key number is required to satisfy an AUT or PRO AC, call the `Get AC Keys` command, described on page 133.)

When the `Decrease` command executes, the card reads the value in the file's most recently written record, decrements the specified amount, and stores the result in the file's oldest record (record #1). The updated record becomes the currently selected record. You can call a `Get Response` command to retrieve the updated value and the amount of the decrement.

To execute the `Decrease` command successfully, these conditions apply:

- The record must contain a positive value.
- The record value must be equal to or greater than the amount of the decrement. (For example, the command fails if all bytes are set to `00`h.)
- The record must be 3–252 bytes long (`03`–`FC`h).

Command Format

Select the cyclic EF whose record value you want to decrease and call the `Decrease` command in the following format.

Command	CLA	INS	P1	P2	Lc
`Decrease`	`F0`	`30`	`00`	`00`	`03` + *X*

Parameters

Name	Length	Value / Meaning
P1	1 B	`00`h
P2	1 B	`00`h
Lc	1 B	Length of the input data + *X*. (*X* =Cryptogram length if AC = PRO, or 0 if no PRO AC.)
Input Data	3 + *X* B	Value to be deducted + cryptogram, if the PRO AC applies.
SW1, SW2	2 B	Status word bytes the card returns.

Response Data Available

If you follow the `Decrease` command immediately with a `Get Response` command, the card returns the data shown in the following table. Note that you can retrieve all or part of the record data. In either case, the first byte of the returned data is always the first byte in the record.

Bytes	Description of Data	Length
1–*lgth*	The decreased value stored in the record.	*lgth* B
(*lgth* + 1) – (*lgth* + 3)	The deducted value.	3 B

Status Words Returned

Hex Value	Meaning
`61`*xx*	*xx* bytes of response data are available for return by `Get Response`.
`6283`	The currently selected EF is invalidated.
`6300`	PRO authentication failed because the cryptogram is wrong, or the data is not the correct length for PRO mode.
`6581`	Memory-related problem: The EEPROM may have failed.
`6703`	The Lc value (value to be deducted) does not match the record size or does not match the amount of input data. Enter `03`h.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	The AC cannot be satisfied because either the cryptographic key, CHV key, or key file is missing.
`6983`	The key required for PRO authentication is blocked.
`6985`	The PRO AC cannot be satisfied because the host application did not send a `Get Challenge` command to the card.
`6986`	A DF is currently selected. Select a cyclic EF.
`6A80`	A linear or transparent EF is currently selected. Select a cyclic EF.
`6A83`	The record length out of range: It is less than 3 bytes (`03`h) or greater than 252 bytes (`FC`h).
`6B00`	Incorrect values were entered for P1, P2, or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.

Hex Value	Meaning
`6F00`	Technical problem without a specified diagnostic.
`9850`	The minimum value has been reached, or the remaining value is less than the amount of the decrement. The card cannot perform the decrement.

See Also:

- `Increase` command on page 143
- `Update Record` command on page 195
- `Read Record` command on page 157
- `Create Record` command on page 105
- `Get Response` command on page 138
- "Commands That Are Subject to Access Conditions," on page 64

Delete File

Use the `Delete File` command to delete a file under the current sekected directory. You can delete an EF or an empty DF. You can use this command to reclaim space on the card, but only if you use the Last In/First Out (LIFO) mechanism described below. Once you delete a file, the value of the EEPROM space the file occupied is reset to null values.

S/R: Send *(Case 3)*

AC – To delete a file under the master file (MF), first satisfy the AUT access condition. To delete a file under a subsidiary DF, satisfy that DF's AC. (For instructions about retrieving a file's AUT AC key number information, see `Get AC Keys` on page 133.)

- *Do not delete the external key file (the file with ID 0011), located under the master file. If you delete this file or its AAK, you will be unable to satisfy the access condition for the root directory of the card system. You will permanently lock yourself out of the card.*
- *Delete EFs and DFs in the exact reverse order from the order in which they were created (in LIFO—Last In, First Out—order). Within a DF, delete the most recently created EF before you delete older EFs. You can select its parent directory and delete the DF. If a DF contains both EFs and other DFs, make sure you delete these files in LIFO order.*

  To determine the order of file creation, you can build a memory map by using the `Dir Next` command.

NOTE *Unlike most commands, you can execute a `Delete File` command regardless of the target file's validation status. You can delete an invalid EF or an empty invalid DF. You cannot delete an EF in an invalid DF.*

Command Format

Select the DF that contains the EF or empty DF you want to delete and call the `Delete File` command in the following format.

Command	CLA	INS	P1	P2	Lc
`Delete File`	`F0`	`E4`	`00`	`00`	`02` + *X*

Parameters

Name	Length	Value / Meaning
P1	1 B	`00h`
P2	1 B	`00h`
Lc	1 B	Length of input data + cryptogram (if PRO mode applies).

Name	**Length**	**Value / Meaning**
Input Data	2 + *X* B	Bytes 1–2 = File ID, and Bytes 3+ = The cryptogram, if the AC is PRO.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	**Meaning**
`6283`	The currently selected DF is invalidated. You cannot invalidate an EF contained in an invalidated DF.
`6300`	PRO authentication failed because the cryptogram is wrong.
`6581`	Memory-related problem: The EEPROM may have failed.
`67`*xx*	The value entered for Lc is unsupported or does not match the amount of data included. The correct Lc value appears in place of *xx*.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	The AC cannot be satisfied because either the cryptographic key, CHV key, or key file is missing.
`6982`	The required AC was not satisfied because the key is invalid or no key was presented.
`6983`	The key required for PRO authentication is blocked.
`6985`	The PRO AC cannot be satisfied because the host application did not send a `Get Challenge` command to the card.
`6A80`	No DR/MF selected as current.
`6A82`	The specified file ID was not found.
`6B00`	Invalid values entered for P1 (mode), P2 (key number), or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The card deleted the file.

See Also: `Dir Next` command on page 118

DES Block

Use the `DES Block` command after a `DES Block Init` command if you need to encrypt or decrypt more than 232 bytes of data. The `DES Block Init` command handles the first data string of up to 232 bytes. To process more data, issue a `DES Block` command. Each `DES Block` command enables you to encrypt or decrypt an additional string of octet-format data that is no more than 232 bytes in length.

S/R: Send/Receive *(Case 4)*

AC – The access condition is CHV1. Before you can issue a successful `DES Block` command in a given context, you must establish appropriate access rights by executing a successful `Verify` CHV command (as described on page 198).

You must call a single `DES Block Init` command before you call the first `DES Block` command. The `DES Block` command fails unless it immediately follows a successful `DES Block Init` or `DES Block` operation.

As with the `DES Block Init` command, you use `DES Block` to perform a cipher block chaining (CBC) DES or 3DES operation. The card uses the initialization vector (IV) that was generated by the previous command and stored in RAM.

Use P1 to set the command mode to encryption or decryption, and P2 to specify the key number. The key you specify is either a single-length DES or double-length 3DES key in the relevant internal key file.

Retrieving Available Return Data – To retrieve the ciphertext or the plaintext that results from the operation, follow with a `Get Response` command. The data you can retrieve is the same length as the input data.

Command Format

Command	CLA	INS	P1	P2	Lc
`DES Block`	`F0`	`58`	*mode*	*key nbr*	*lgth*

Parameters

Name	Length	Value / Meaning
P1	1 B	Encryption/decryption mode: • `00h` = Encrypt • `01h` = Decrypt
P2	1 B	Number of the DES or 3DES key to use for encryption or decryption, located in the relevant internal key file. Enter a value from `00h` (for key 0) to `0Fh` (for key 15).
Lc	1 B	Length of the input data, a value from `08–E8h` in octet format (8–232 bytes long, evenly divisible into 8-byte blocks).
Input Data	*lgth*	Data to be encrypted or decrypted: Any octet amount from 8–232 bytes long.
SW1, SW2	2 B	Status words returned by the card

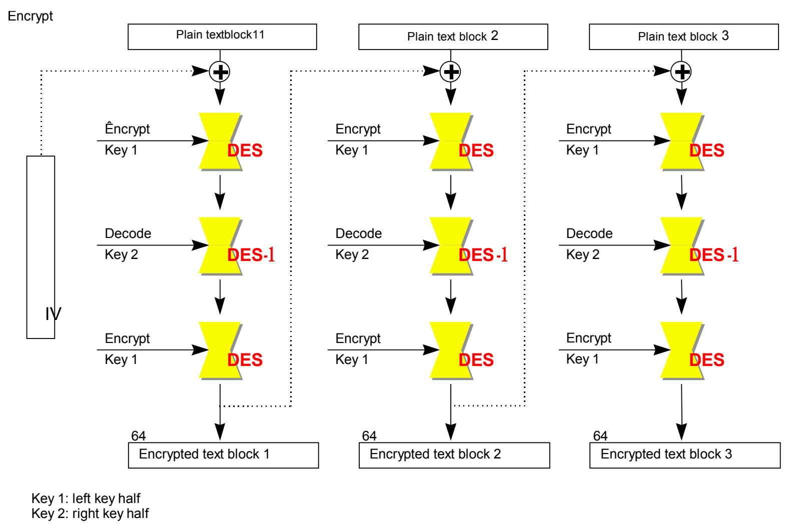
Response Data Available

If you follow a final `DES Block` command immediately with a `Get Response` command, the card returns ciphertext or plaintext produced by the final CBC DES or 3DES operation.

Status Words Returned

Hex Value	Meaning
`61`*xx*	Command succeeded, and *xx* bytes of ciphertext or plaintext are available for return by a `Get Response` command.
`6700`	The length value specified for Lc does not match the length of the input data, or exceeds the maximum supported length.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	Either the key slot specified in P2 does not exist, or no relevant internal key file exists.
`6982`	The required CHV1 AC was not satisfied.
`6985`	No DES Block Init done before Algo Id not authorized for the key used.
`6B00`	Invalid values entered for P1 (mode), P2 (key number), or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.

How the CBC Algorithm Works



CBC Encryption Process with a Double-Length 3DES Key

If the command mode is set to encryption:

1 The `DES Block Init` command initializes with the initialization vector (IV) value set to `00 00 00 00 00 00 00 00`h.

2 The card reads the first 8-byte block of data, then exclusive ORs (XORs) these bytes with the IV value.

This operation produces an 8-byte block of XORed plaintext.

3 The card uses the specified DES or 3DES key to encrypt this data block.

This operation produces an 8-byte block of ciphertext, which is stored in the IV.

4 The card XORs the ciphertext with the next 8-byte block of plaintext.

This operation produces a new 8-byte block of XORed plaintext.

5 The card encrypts the new plaintext block.

The card stores the resulting 8-byte block of ciphertext in the IV.

6 Steps 4 and 5 repeat until the whole file is encrypted.

The command returns the status `61`*xx* to indicate that the data is encrypted, and to inform you that the resulting cryptogram's length is *xx* bytes.

If the command mode is set to decryption:

1 The `DES Block Init` command initializes with the IV set to `00 00 00 00 00 00 00 00`h. (For a `DES Block` command, the IV is initialized with the 8-byte block of encrypted data produced by the previous `DES Block` or `DES Block Init` command.

2 The card reads the first 8 bytes of data and stores them as the next IV.

3 The card uses the specified DES or 3DES key to decrypt the first 8 bytes of ciphertext, and XORs these bytes with the current IV.

4 The next IV becomes the current IV. The card decrypts the next 8 bytes of ciphertext, and XORs these bytes with the current IV.

The current encrypted data becomes the next IV.

5 Step 4 repeats until the entire file is decrypted.

The command returns the status `61`*xx* to indicate that the data is decrypted, and to inform you that the resulting plaintext is *xx* bytes long.

See Also:

- `DES Block Init` command on page 116
- `Get Response` command on page 138

DES Block Init

Use the `DES Block Init` command to encrypt or decrypt blocks of data with the Data Encryption Standard (DES) or Triple Data Encryption Standard (3DES) algorithm in cipher block chaining (CBC) mode. You use the `DES Block Init` command for the initial or only data block, which can be any octet length between 8 and 232 bytes. If you need to process additional data, follow with one or more `DES Block` commands.

S/R: Send/Receive *(Case 4)*

AC – The access condition is CHV1. Before you can issue a successful `DES Block Init` command in a given context, you must establish appropriate access rights by executing a successful `Verify CHV` command (described on page 198).

If you use the `DES Block Init` command for encryption, the cipher is used to encrypt plaintext in 8-byte blocks. (The data must be divisible into 8-bytes components.) The first data block is XORed with an IV, then encrypted. The resulting ciphertext is XORed with the next plaintext block. The process continues, concatenating a series of data blocks with the XOR operation so that later blocks are dependent on earlier ones. When you decrypt the block chain, its built-in dependence reveals whether any changes were made to the original message. (For more information about CBC encryption and decryption, see “How the CBC Algorithm Works,” on page 114.)

The IV for the `DES Block Init` command is set to null values. At the end of the encryption process, the card stores the final IV in RAM and uses it in any subsequent call to the `DES Block` command.

Use P1 to set the command mode to encryption or decryption, and P2 to specify the key number. The key you specify is either a single-length DES or double-length 3DES key in the relevant internal key file.

Retrieving Available Return Data – To retrieve the ciphertext or the plaintext that results from the operation, follow with a `Get Response` command. The data you can retrieve is the same length as the input data.

Command Format

Command	CLA	INS	P1	P2	Lc
`DES Block Init`	`F0`	`56`	*mode*	*key nbr*	*lgth*

Parameters

Name	Length	Value / Meaning
P1	1 B	Encryption/decryption mode: • `00h` = Encrypt • `01h` = Decrypt
P2	1 B	Number of the DES or 3DES key to use for encryption or decryption, located in the relevant internal key file. Enter a value from `00h` (for key 0) to `0Fh` (for key 15).
Lc	1 B	Length of the input data, a value from `08–E8h` in octet format (8–232 bytes long, evenly divisible into 8-byte blocks).
Input Data	*lgth*	Data to be encrypted or decrypted: Any octet amount from 8–232 bytes long.
SW1, SW2	2 B	Status words returned by the card

Response Data Available

If you follow a sole `DES Block Init` command immediately with a `Get Response` command, the card returns the ciphertext or plaintext produced. Retrieve the data only if no `DES Block` calls are needed.

Status Words Returned

Hex Value	Meaning
`61`*xx*	Command succeeded, and *xx* bytes of ciphertext or plaintext are available for return by a `Get Response` command.
`6700`	The length value specified for Lc does not match the length of the data entered or exceeds the maximum supported length.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	Either the key slot specified in P2 does not exist, or no relevant internal key file exists.
`6982`	The required CHV1 AC was not satisfied.
`6985`	Algo Id not authorized for the key used.
`6B00`	Invalid values entered for P1 (mode), P2 (key number), or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.

See Also:

- `DES Block` command on page 112
- `Get Response` command on page 138

Dir Next

Use the `Dir Next` command to find out which files are in the currently selected directory and retrieve information about the files. The response data for each call to the command describes a single file. You can use this command to build a memory map that tells you the order in which the files were created.

S/R: Send/Receive *(Case 4)*

AC – If you want to display information about files under the master file (MF), you must first satisfy the AUT access condition. If you want to display information about files under a DF other than the MF, you must first satisfy the access condition for the selected DF.

The first `Dir Next` command returns information about the first file in the current directory. Each additional command returns information about the next file in the sequence. Once you reach the end of the file sequence, a call to the `Dir Next` command returns the status word `6A82` (no more files found).

File Size Information – The information the `Dir Next` command returns is similar to the information you retrieve by calling a `Get Response` command after you select a file, except for the format of the file size.

- A `Get Response` command returns the size of the file body.
- A `Dir Next` command returns file size information (FSI), which includes the input parameter string with the file size. This value is rounded up (if necessary) to be modulo 4 (evenly divisible by 4). If rounding occurs, an `8` appears as the MSN of the first FSI byte.

  Examples: A file with a 12-byte body + 16-byte input parameter string = 28 bytes for an FSI of `001C`h. A file with a 14-byte body + 16-byte input parameter string = 30 bytes, which is rounded to 32 bytes for an FSI of `8020`h.

NOTE *AC key number information is not included in the data returned by the `Dir Next` command. To find out which key is used to satisfy an AUT or PRO AC, use the `Get AC Keys` command, described on page 133.*

Command Format

Select the directory whose contents you want to read and call the `Dir Next` command in the following format.

Command	CLA	INS	P1	P2	Le
`Dir Next`	`F0`	`A8`	`00`	`00`	*lgth*

Parameters

Name	Length	Value / Meaning
P1	1 B	`00`h
P2	1 B	`00`h
Le	1 B	Length of data to retrieve (a value between `01–16`h). Enter `16`h to get all the data, or enter a lower value to get partial data. (For example, enter `04`h to get the first 4 bytes.)
Output Data	1–16 B	The file information the card returns, as detailed in the following Output Data table.
SW1, SW2	2 B	Status word bytes the card returns.

Output Data

Byte(s)	Description
1–2	File size information (FSI) = File body + input parameter string + rounding (if needed) to equal a value that is evenly divisible by 4 (modulo 4). If rounding occurs, an 8 appears in the MSN of the first byte, as described on page 118.
3–4	File identifier
5	File type: • `01`h = Transparent EF • `02`h = Fixed-length linear EF • `04`h = Variable-length linear EF • `06`h = Cyclic EF • `38`h = Dedicated file

Byte(s)	Description
6	Instruction byte, which shows whether the `Increase` and `Decrease` commands on cyclic EFs are enabled or restricted: • `00–3F`h = No `Decrease` or `Increase` permitted • `40–7F`h = `Decrease` not permitted • `80–BF`h = `Increase` not permitted • `C0–FF`h = No restrictions *For more information, see the file creation input data for cyclic EFs (byte 8, described on page 101).*
7–9	Access conditions (ACs) set on the file *For more information, see the AC file creation input data (DF bytes 8–11, described on page 99 or EF bytes 9–11, described on page 101).*
10	File status: • `01`h = Activated • `00`h = Invalidated
11	RFU
12–14	Key numbers (in the relevant external key file), used to fulfill ACs set for particular commands *For more information, see the AC key file creation input data (bytes 14–16, described on page 101 for DFs and on page 103 for EFs.*
15	File type-dependent information: • *Directory* – Number of subdirectories (DFs) • *Linear or cyclic EF* – Record length • *Transparent EF* – RFU (returns `00`h)
16	File type-dependent information: • *Directory* – Number of EFs • *Linear or cyclic EF* – Number of records • *Transparent EF* – RFU (returns `00`h)

Status Words Returned

Hex Value	Meaning
`6710`	The value specified for Le is unsupported. Enter `10`h to retrieve all data.
`6CYYh` `(YY=P3)`	**32K+e-gate USB mode only**: wLength of GetData request does not correspond to P3.
`6982`	The required AC was not satisfied: To display information about the MF, you must first satisfy the AUT condition. To display information about another DF, you must satisfy that DF's AC for the `Dir Next` command.
`6A80`	An EF is currently selected. Select a DF.
`6A82`	The specified file ID was not found or information has already been displayed for all files in the current DF.
`6B00`	Incorrect values entered for P1, P2, or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The card returned the file information.

See Also: "Commands That Are Subject to Access Conditions," on page 64

External Authenticate Using DES

Use the `External Authenticate Using DES` command to establish access rights for the host application to interact with the card and to satisfy an AUT access condition (AC).

S/R: Send *(Case 3)*

AC – Not applicable

To execute an `External Authenticate Using DES` command successfully, the card must contain a relevant external key file, and the specified key slot must contain a valid key that is not blocked.

NOTE *If security is not a concern, you can also satisfy the AUT AC by calling the `Verify Key` command. (Unlike the `External Authenticate Using DES` command, the `Verify Key` command transmits the key to the card in plaintext.)*

How External Authentication Works — The following steps show how external authentication works.

1. The host sends the card a `Get Challenge` command with `08`h as the Le value.

   The card returns an 8-byte plaintext challenge.

2. The host application encrypts the challenge with a DES or 3DES key and truncates the cryptogram to 6 bytes. The host sends the card an `External Authenticate Using DES` command, which includes the cryptogram and the number of the decryption key (located in the relevant external key file).

3. The card decrypts the cryptogram with the specified key, using the algorithm specified in the key's algorithm ID.

   If the result matches the original challenge, the card grants the AUT AC to the host. The AC persists until one of the events described on page 67 occurs.

   If the verification fails, the key's attempt counter is decremented. If the attempt counter reaches the null value, the key is blocked.

Command Format

Command	CLA	INS	P1	P2	Lc
`External Authenticate Using DES`	`C0`	`82`	`00`	`00`	`07`

Parameters

Name	Length	Value / Meaning
P1	1 B	`00`h
P2	1 B	`00`h
Lc	1 B	`07`h — Length of input data (1-byte key number + 6-byte truncated cryptogram).
Input Data	7 B	Byte 1 — `00–0F`h: Number of the key slot in the relevant external key file to use for decryption (key 0–15), and Bytes 2–7 — The truncated cryptogram.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6300`	Wrong cryptogram entered: The card used the specified key to decrypt the ciphertext and the result does not match the original challenge.
`6707`	Value entered for Lc is incorrect. (Enter `07`.)
`6CYYh` `(YY=P3)`	**32K+e-gate USB mode only**: wLength of GetData request does not correspond to P3.
`6981`	Key specified in input data does not exist: Either the key or relevant external key file is missing.
`6983`	Key specified in input data is blocked.
`6985`	Host did not send a successful `Get Challenge` command to the card immediately before issuing the `External Authenticate Using DES` command.
`6B00`	Incorrect values entered for P1, P2, or both. (Enter `00`h for P1 and P2.)
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	Command succeeded: The host application has established the AUT AC and can interact with the card.

See Also:

- `Verify Key` command on page 200
- `Internal Authenticate Using DES` command on page 146

Generate DES Key

Use the `Generate DES Key` command to instruct the card to generate a DES or 3DES key and store it in on the card in the relevant internal key file.

S/R: — *(Case 3)*

AC – The access condition is CHV1. Before you can issue a successful `Generate DES Key` command in a given context, you must establish appropriate access rights by executing a successful `Verify CHV` command (described on page 198).

NOTE *You can use this command to generate keys for an internal key file, but not for an external key file.*

Before you can generate a key with this command, the specified key slot must already exist in the relevant internal key file. The key slot can be empty or can contain an existing key that will be overwritten. (The operation does not affect the maximum attempt setting or the counter for remaining attempts.)

The following table shows the format of the content for each single-length DES or double-length 3DES key slot in the internal key file:

Byte	**Description**
1	RFU (key slot header) = 1 B
2	Length of the key = 1 B
3	Key number, which indicates key's algorithm ID = 1 B
4–11 *or* 4–19	Key value in LSB-first format or empty bytes ready to receive the key value. (8 B for a single-length DES key or 16 B for a double-length 3DES key.)
12 *or* 20	Maximum number of validation attempts allowed for the key = 1 B
13 *or* 21	Number of validation attempts that currently remain for the key = 1 B

The key length and algorithm ID for DES or 3DES keys are stored in the internal key file. (Note that you cannot use this command to create verification keys in the external key file.)

Command Format

Navigate to the directory that holds the target internal key file, and call the `Generate DES Key` command in the following format.

Command	CLA	INS	P1	P2	Lc
`Generate DES Key`	`F0`	`50`	`00`	*key nbr*	`00`

Parameters

Name	Length	Value / Meaning
P1	1 B	`00h`
P2	1 B	Number of the key slot for storing the new key in the relevant internal key file, a value from `00h` (key 0) to `0Fh` (key 15).
Lc	1 B	`00h` (No input data is sent with the command.)
SW1, SW2	2 B	Status words returned by the card

Status Words Returned

Hex Value	Meaning
`6581`	Memory-related problem: The EEPROM may have failed.
`6700`	Incorrect value entered for Lc. (Enter `00h`.)
`6981`	Either the key slot specified in P2 does not exist, or there is no relevant internal key file on the card.
`6982`	The required CHV1 AC was not satisfied.
`6B00`	Incorrect or unsupported values entered for P1, P2, or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The specified DES key was generated and stored in the card's relevant internal key file.

Key Checking

When DES key generation begins, the internal random number generator (RNG) produces a key value. The key is tested for weakness. If the key matches any of the weak or semiweak keys described in this topic, it is discarded. A new key is generated and tested. This process continues until a suitable key is found.

Weak Keys — The Cryptoflex card rejects any key that matches one of the four DES weak keys. For these keys: *E(K,E(K,M)) = M*. That is, if you encrypt a message twice with one of these keys, you get the original message back. The weak keys are:

```
0101 0101 0101 0101
FEFE FEFE FEFE FEFE
1F1F 1F1F 1F1F 1F1F
E0E0 E0E0 E0E0 E0E0
```

Semiweak Keys — The Cryptoflex card also rejects any of the twelve keys that appear in the six key pairs considered semiweak for DES use. If one of these keys is used with its pair, *(couples K, K'): E(K,E(K',M)) = M*. That is, if you encrypt a message with one semiweak key, and then decrypt it with the key's complement, you get the original message back. The semiweak key pairs are shown in the following table.

K value	K′ value
01FE 01FE 01FE 01FE	FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1	E01F E01F F10E F10E
01E0 01E0 01F1 01F1	E001 E001 F101 F101
1FFE 1FFE 0EFE 0EFE	FE1F FE1F FE0E FE0E
011F 011F 010E 010E	1F01 1F01 0E01 0E01
E0FE E0FE F1FE F1FE	FEE0 FEE0 FEF1 FEF1

For faster performance, the card checks on a superset of the 16 weak and semiweak keys. The Cryptoflex card does verify that two different keys are used for 3DES operations.

NOTE *Generating a new DES key changes only the key value. It does not affect the number of verification attempts currently logged by the key counter or the Max Attempt value.*

See Also: `Generate RSA Keys` command on page 127

Generate RSA Keys

Use the `Generate RSA Keys` command to create a pair of public and private RSA keys and store them in private and public key files. RSA keys can be in any of these formats: 512-, 768-, 1024-, or 2048-bit. You can call the command in a mode that enables you to retrieve the public key modulus with a follow-up `Get Response` command.

32K+SS V1 32K+e-gate only

Public keys can be generated in three different formats:

- All public components are stored in the public key file.
- Only the public modulus and exponent are stored in the public key file.
- Only the public exponent is stored in the public key file.

S/R: Send *(Case 3)*

AC – The access condition is CHV1. Before you can issue a successful `Generate RSA Keys` command in a given context, you must establish appropriate access rights by successfully executing `Verify CHV` (page 198).

Before you call a `Generate RSA Keys` command, complete these tasks:

- If the local directory does not already contain a private key file ($EF_{RSA\text{-}PRI}$ with file ID 0012) and public key file ($EF_{RSA\text{-}PUB}$ with file ID 1012), create these two files to hold the key pair. (For more information about the file formats, see “RSA Key Files,” on page 34.)
- Execute `Verify CHV` for the CHV file relevant to the directory that contains the RSA key files. If no relevant CHV file exists, create one.

Depending on the public key format, Montgomery constants (*JO* and *H*), which are used to compute *N*, might also be included in the public key file. See “RSA Key Files,” on page 34 for information about which public key formats store the Montgomery constants (*JO* and *H*) in the public key file.

Assigning a Key Number to the Private and Public Key

Use P1 to specify the key number of the private and public key. When you submit the command, the card checks to see if a key already exists that matches the specified number, and one of these actions results:

- If the key number is available, the key pair is added to the key files on-the-fly: you do not have to prepare containers for new key pairs before you create them.

- If you specify the number of an existing key, the card performs a key length check. If the new existing keys are the same length, the existing key pair values are overwritten. If the lengths do not match, the command returns the status word `6A83`, and the original key pair values are unchanged.

The card stores keys in the RSA key files in the order of creation, not consecutively by number. The hexadecimal key number you enter for P1 is derived from an 8-bit binary value.

Record the numbers you assign to each private key—you cannot retrieve this information later.

32K+SS V1
32K+e-gate
only

Choosing the Public Key Format

Also use P1 to specify the public key format: either store all public components in the public key file, store the public modulus and exponent only, or store the public exponent only. The supported values are listed in the Extended P1 Parameters table on page 130.

Choosing the RSA Format

The P2 value determines the key's RSA format — 512-bit, 768-bit, 1024-bit, or 2048-bit. The format sets the length of the private key's prime factors and the public key's modulus. Supported values are in the Parameters table, page 129.

Specifying the Public Exponent

The Lc parameter specifies the length of the public key exponent, which is always a 4-byte value with a maximum value of `FF FF FF FF`h (4,294,967,295). The public exponent also must satisfy these criteria: its value must be greater than or equal 3 and be an odd, prime number in LSB-first format. The more bits in the public exponent that equal "1," the slower the RSA computation will be. The number `01 00 01` is a popular choice for the public exponent: it has good mathematical properties, and the cryptoprocessor has optimized routines for it. You enter the public key exponent in the input data block LSB-first.

Key Replacement

If the target key slot contains a key when you call a successful `Generate RSA Keys` command, the key is overwritten. If, however, the key length check shows that the new and original keys are different lengths, the command fails, but leaves the original key undisturbed. In any other case, the key may be invalid. To correct this problem, you must call a successful `Generate RSA Keys` command.

NOTE *When the card generates RSA keys, it calculates Chinese Remainder Theorem (CRT) information to give faster calculation of modular exponentiation.*

Command Format

Command	CLA	INS	P1	P2	Lc
`Generate RSA Keys`	`F0`	`46`	*see Parameters table*	`40` / `60` / `80` / `00`	`04`

Parameters

Name	Length	Value / Meaning
P1	1 B	***For 16K+SS V1 cards:*** Key number, which specifies a new key slot created on-the-fly, or an existing key slot whose value will be overwritten. RSA key files support a maximum of 15 keys, which can be numbered in any order. Use either: • Storage only mode — Enter `00h-0Eh`, or • Retrieval mode — Enter `80h-8Eh`. Retrieval mode enables you to retrieve the public key modulus by calling a follow-up `Get Response` command. If you specify the value of P1 as `00`h, the key number is 1. (The value 0 is reserved as a marker for the end of the file.) ***For 32K+SS V1 and 32K+e-gate cards, see the Extended P1 Parameters table.***
P2	1 B	Length of the public modulus: • `40`h = 64 bytes (512-bit RSA key), • `60`h = 96 bytes (768-bit RSA key), • `80`h = 128 bytes (1024-bit RSA key), or • `00`h = 256 bytes (2048-bit RSA key).
Lc	1 B	`04`h = Length of the public exponent (e_p).
Input Data	4 B	Value of the public exponent (in LSB-first format).
SW1, SW2	2 B	Status word bytes the card returns.

Extended P1 Parameters

32K+SS V1 32K+e-gate only

Parameter	Effect on the Public Components	Stored in Public Key File
0000*xxxx*=0Xh	All components of public key generated in the public key file.	Public modulus, J0, H, and public exponent
1000*xxxx*=8Xh	All components of public key generated in the public key file and the public modulus is ready to be output.	Public modulus, J0, H, and public exponent
0100*xxxx*=4Xh	Only public module and exponent generated in the public key file.	Public modulus and public exponent
1100*xxxx*=CXh	Only public module and exponent generated in the public key file and the public modulus is ready to be output.	Public modulus and public exponent
1010*xxxx*=AXh	Only public exponent in the public key file and the public modulus is ready to be output.	Public exponent
1001*xxxx*=9Xh	Public modulus is ready to be output	Nothing

where xxxx=X denotes the key number to be generated.

RSA key files support a maximum of 15 keys, which can be numbered in any order. Use either:

- Storage only mode — Enter `00h-0Eh`, or
- Retrieval mode — Enter `80h-8Eh`.

Retrieval mode enables you to retrieve the public key modulus by calling a follow-up `Get Response` command.

If you specify the value of P1 as `00h`, the key number is 1. (The value 0 is reserved as a marker for the end of the file.)

MSB of P1=0, private and public keys will be stored in the corresponding files;

MSB of P1=1, the public modulus will be output by an immediately following GET RESPONSE.

NOTE *When an RSA key is to be generated with P1=9Xh, the public exponent given by the command in the data field must be the same as the one stored in EEPROM during pre-personalization. (The default value is 01000100 in*

LSB first format.) If the public exponent is not the one stored in EEPROM, an error status word `6FOO` *is sent and the private key is not valid anymore. A successful RSA key generation must be performed iwth this key index to be able to use this key for RSA signature.*

Retrieving the Public Key

You can retrieve the public key value (modulus *N)* in either of these ways:

- Select the public key file and call a `Read Binary` command.
- Specify P1 as a value between 80h-80Eh, and follow the `Generate RSA Keys` command immediately with a `Get Response` command. The card returns the public key modulus *(N)*. The length of available data depends on the RSA key strength, as described below:
  - *512-bit key* = 64-byte public key modulus (`40`h)
  - *768-bit key* = 96-byte public key modulus (`60`h)
  - *1024-bit key* = 128-byte public key modulus (`80`h)
  - *2048-bit key* = 256-byte public key modulus (`100`h, expressed as `00`h)

Status Words Returned

Hex Value	Meaning
`6100`	The command (with a P1 value of `00-0E`h) succeeded: the 2048-bit RSA key pair has been created and stored. Call a `Get Response` command to retrieve the 256-byte public key value (modulus *N*).
`6140`	The command (with a P1 value of `00-0E`h) succeeded: the 512-bit RSA key pair has been created and stored. Call a `Get Response` command to retrieve the 64-byte public key value (modulus *N*).
`6160`	The command (with a P1 value of `00-0E`h) succeeded: the 768-bit RSA key pair has been created and stored. Call a `Get Response` command to retrieve the 96-byte public key value (modulus *N*).
`6180`	The command (with a P1 value of `00-0E`h) succeeded: the 1024-bit RSA key pair has been created and stored. Call a `Get Response` command to retrieve the 128-byte public key value (modulus *N*).
`6581`	Memory-related problem: The EEPROM may have failed.

Hex Value	Meaning
6700	Length specified for the public exponent does not match the data entered or exceeds the maximum supported length. Any key that was in the target slot is now invalid.
6CYYh (YY=P3)	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
6981	The modulus length specified in P2 is inconsistent with the key length. Any key that was in the target slot is now invalid.
6982	The required CHV1 AC was not satisfied or an unsupported value is specified as the public exponent.
6A80	Private/public key file is not a transparent EF.
6A82	The private key file, public key file, or both were not found.
6A83	The private key file, public key file, or both are not large enough to hold the specified keys.
6B00	Unsupported values were entered for P1 (key number), P2 (modulus length), or both.
6F00	Technical problem without a specified diagnostic. Try calling the command again.
9000	The command (with a P1 value of 00-0Eh) succeeded: The specified RSA key pair has been created and stored in the private and public key files. (If the command's P1 value is 80-8Eh, the card returns status word 6100, 6140, 6160, or 6180 instead of 9000.)

See Also: `Generate DES Key` command on page 124

Get AC Keys

Use the `Get AC Keys` command to retrieve the key numbers that apply to commands protected by AUT or PRO access conditions (ACs).

S/R: Receive *(Case 2)*

AC – The access condition is CHV1. Before you can issue a successful `Get AC Keys` command in a given context, you must establish appropriate access rights by executing a successful `Verify CHV` command (described on page 198).

When you create a file, you include a set of 6 nibbles command matrix in two locations (bytes 9–11 and bytes 14–16). Several of the nibbles in the matrix hold information about the ACs for particular commands. The nibbles specified for bytes 9–11 contain information about which ACs are set for the commands. The nibbles in bytes 14–16 specify which key number (if any) is needed to satisfy each AC. To satisfy a PRO or AUT AC, you use the corresponding key in the relevant external key file.

NOTE *Key number and key slot values are not identical. For keys in the external key file, key numbers are zero-based. (That is, key one occupies key slot 0.) Key numbers are not related to slot numbers.*

To find out which commands are protected by an AUT or PRO AC in the current file, use the `Get Response` command and examine bytes 9–11 of the file input parameter information the card returns.

For example, let's say you select a DF and issue a `Get Response` command to examine the file input parameter data. In this example, the `Create File` nibble in bytes 9–11 has a value of `4`h *(AUT)*, `8`h *(CHV1 and AUT)*, or `9`h *(CHV2 and AUT)*, so you know you must supply a key from the relevant external key file in order to create a new file in that directory. To find out which key is required, you call the `Get AC Keys` command and examine the nibble set returned. The `Create File` nibble has the value `00`h, the value for key number 1. This means you must authenticate the AC for key 1 in the relevant external key file.

Key Numbers Used to Protect Commands

The following tables show which commands correspond to nibble positions for a currently selected DF or EF.

Dedicated File

Byte Returned	MSN Contains Key Number for:	LSN Contains Key Number for:
1	`Dir Next`	RFU
2	`Delete File`	`Create File`
3	RFU	RFU

Transparent Elementary File

Byte Returned	MSN Contains Key Number for:	LSN Contains Key Number for:
1	`Read Binary`	`Update Binary`
2	`Read Binary Enciphered`	`Update Binary Enciphered`
3	`Rehabilitate`	`Invalidate`

Cyclic Elementary File

Byte Returned	MSN Contains Key Number for:	LSN Contains Key Number for:
1	`Read Record`	`Decrease/Update Record`[1]
2	`Increase`[1]	RFU
3	`Rehabilitate`	`Invalidate`

1 The Decrease and Increase commands' availability depends on the values set in byte 8, as described on page 101.

Linear Elementary File

Byte Returned	MSN Contains Key Number for:	LSN Contains Key Number for:
1	`Seek, Read Record`	`Update Record`
2	RFU	`Create Record`
3	`Rehabilitate`	`Invalidate`

Command Format

Command	CLA	INS	P1	P2	Le
`Get AC Keys`	`F0`	`C4`	`00`	`00`	`03`

Parameters

Name	Length	Value / Meaning
P1	1 B	`00h`
P2	1 B	`00h`
Le	1 B	`03h`: Length of data to be returned.
Output Data	3 B	Key numbers the card returns.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6700`	Incorrect Le value entered. Enter `03h` for the Le parameter.
`6CYYh` `(YY=P3)`	**32K+e-gate USB mode only**: wLength of GetData request does not correspond to P3.
`6982`	The required CHV1 AC was not satisfied.
`6B00`	Incorrect values entered for P1, P2, or both. (Enter `00h` for P1 and P2.)
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The card returned AC information for the currently selected file.

See Also:

- "Setting Access Rights on Card Operations," on page 62
- `Dir Next` command on page 118
- `Get Response` command on page 138

Get Challenge

Use the `Get Challenge` command to ask the card to return a challenge (a random alphanumeric string) and begin external authentication.

S/R: Receive *(Case 2)*

AC – Not applicable

Uses for Challenges — In external authentication, the host application encrypts the challenge it receives from the card and sends it back to prove possession of the appropriate key (a key that matches the card's key). The card may also occasionally need to generate a challenge for other purposes—for example, to use as a session key or to create padding. The card uses a challenge once, then deletes it.

Challenge Length — The length of the challenge varies according to the needs of the cryptographic commands that use the challenge. For a Cryptoflex card, a challenge can be a maximum of 128 bytes long. To use the challenge with an `External Authenticate Using DES` command, the challenge must be 8 bytes long.

Command Format

Command	CLA	INS	P1	P2	Le
`Get Challenge`	`C0`	`84`	`00`	`00`	*output lgth*

Parameters

Name	Length	Value / Meaning
P1	1 B	`00`h
P2	1 B	`00`h
Le	1 B	Length of the challenge: A value from `1` to `80`h. Enter a value of `08`h if an `External Authenticate Using DES` command follows.
Output Data	*lgth* B	The challenge string the card returns.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6700`	The value entered for the Le parameter is unsupported. (That is, the Le value = `00`h or a value greater than `80`h.)
`6CYYh` `(YY=P3)`	**32K+e-gate USB mode only**: wLength of GetData request does not correspond to P3.
`6B00`	Incorrect values entered for P1, P2, or both. (Enter `00`h for P1 and P2.)
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The card has returned a challenge to the host application.

See Also:

- `External Authenticate Using DES` command on page 122

Get Response

Use the `Get Response` command to retrieve data that was calculated or captured by the previous command.

S/R: Receive *(Case 2)*

AC – Not applicable

You can send a `Get Response` command immediately after any command that returns the status word `61`*xx*. The CLA byte value can either match the CLA byte for the preceding command or equal `C0`h (as described on page 139.)

The following table contains examples of commands that you can follow with `Get Response`. (For details about the data returned, see the command descriptions.)

Command	CLA	Retrievable Data
`Decrease`	`F0`	New record value stored in a cyclic EF, along with the amount of the decrease
`DES Block Init`	`F0`	The ciphertext or plaintext from encryption or decryption of a single data block with DES / 3DES
`DES Block`	`F0`	The ciphertext or plaintext from encryption or decryption of a series of data blocks with DES / 3DES
`Generate RSA Keys`	`F0`	Public key modulus (64, 96, 128, or 256 bytes long), available only if P1 = `80–8E`h
`Increase`	`F0`	New record value stored in a cyclic EF, along with the amount of the increase
`Internal Authenticate Using DES`	`C0`	Truncated 6-byte or full 8-byte cryptogram the card produces (with a DES key in the internal key file) to authenticate itself to the host application
`RSA Signature (Internal Auth)`	`C0`	RSA signature (64, 96, or 128 bytes in length)
`RSA Signature Last`	`00`	RSA signature (64, 96, 128, or 256 bytes in length)
`Select`	`C0`	EF or DF information
`Select EMV`[1]	`00`	ADF or PSE information about the currently selected EMV application
`SHA-1 Last`	`00`	Hash digest (20-byte) produced by SHA-1 operation

1 Command not available for the 32K+e-gate card or the Cryptoflex 32K card version 1.

Command Format

Immediately after you call a command that generates or locates data to retrieve, call a `Get Response` command in the following format.

Command	CLA	INS	P1	P2	Le
`Get Response`:					
after `00` class command	`00`	`C0`	`00`	`00`	*lgth*
after `C0` class or any command	`C0`	`C0`	`00`	`00`	*lgth*
after `F0` class command	`F0`	`C0`	`00`	`00`	*lgth*

Class Byte Options

The class (CLA) byte value can either match the class byte for the preceding command or (in most cases) equal `C0`h. This flexibility gives you two options:

- Use the preceding class value to comply with standards that require matching class values, such as PC/SC, or
- Use the `C0` class value to maintain backward compatibility with earlier Cryptoflex cards. For example, use the `C0` class to develop programs that run on both Cryptoflex 8K V2.1 cards and Cryptoflex 16K cards.

NOTE *A `Get Response` command that follows a `Select EMV` command must have a class value of `00`h. The card does not require matching class values for any other commands.*

Parameters

Name	Length	Value / Meaning
P1	1 B	`00h`
P2	1 B	`00h`
Le	1 B	Length of the data the card will return, which is specific to the preceding command. (See the following table.)
Output Data	*lgth* B	Data the card returns (The information that is available depends on the previous command. See the command descriptions.)
SW1, SW2	2 B	Status word bytes the card returns.

Supported Le Values

Previous Command	**Le Value**
`Decrease`	`01–FF`h – All or part of cyclic record value (1–252 B), starting with the first byte of the record + amount of decrease (3 B).
`DES Block` *or* `DES Block Init`	`08-E8`h – Encrypted or decrypted data. (Available data = 8–232 bytes of octet data.)
`Generate RSA Key` *(if P1 =* `80–8E`*h)*	Public key modulus *(N)*, whose length is key-dependent: • *512-bit key* = `01–40`h: All or part of 64 bytes, • *768-bit key* = `01–60`h: All or part of 96 bytes, • *1024-bit key* = `01–80`h: All or part of 128 bytes, or • *2048-bit key* = `01–00`h (`100`h is expressed as `00`h): All or part of 256 bytes.
`Increase`	`01–FF`h – All or part of cyclic record value (1–252 B), starting with the first byte of the record + amount of increase (3 B).
`Internal Authenticate Using DES`	• `06`h – Truncated DES cryptogram, or • `08`h – Full DES cryptogram.
`RSA Signature Last`	• `01–40`h – All or part of cryptogram for a 512-bit signature (`40`h returns all 64 bytes), • `01–60`h – All or part of cryptogram for a 768-bit signature (`60`h returns all 96 bytes), • `01–80`h – All or part of cryptogram for a 1024-bit signature (`80`h returns all 128 bytes), or • `01–00`h – All or part of cryptogram for a 2048-bit signature (`00`h returns all 256 bytes).
`Select`[1]	The type and amount of data available depends on the currently selected file type (and contents, in the case of a DF): • *DF selected* = `01–17`h: All or part of DF data available, including number of available bytes, file AID, ACs set on the DF, active/invalidated status, CHV1/CHV2 data (if available), and number of EFs and DFs. (See page 178.) • *EF selected* = `01–15`h: All or part of EF data available, including size of file body, file AID, file type, ACs set on the EF, active/invalidated status, and record length (if the EF is a fixed-length linear EF). (See page 179.)

Previous Command	**Le Value**
`Select EMV`[2]	EMV application data, whose length and data depends on the currently selected file: • *ADF selected* = `01–19`h: All or part of 14–25 bytes of TLV-formatted data, which includes the AID value (5–16 bytes) and indicates whether the card has multiple applications. (See page 182.) • *PSE selected* = `01–17`h: All or part of 23 bytes of TLV-formatted data, which includes the 14-byte AID value and the SFI of the directory elementary file. (See page 183.)
`SHA-1 Last`	`01–14`h – All or part of the 20-byte hash digest

1 If you call a Get Response command with a DF selected, the amount of available response data varies according to the presence of relevant CHV files. Bytes 21–23 of the response data are available only if the card contains a relevant EF_{CHV2}. Otherwise, the 20 bytes of data are available. If the card does not contain a relevant EF_{CHV1} file, some of the 20 bytes contain no useful data.

2 Command not available for the 32K+e-gate card or the Cryptoflex 32K card version 1.

Status Words Returned

Hex Value	**Meaning**
`67`*xx*	The Le value is unsupported or does not match the amount of data available. Enter the value that appears in place of *xx*.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of GetData request does not correspond to P3.
`6B00`	Incorrect values entered for P1, P2, or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded, and card returned the specified information.

See Also:

- `Decrease` command on page 107
- `DES Block Init` command on page 116
- `DES Block` command on page 112
- `Increase` command on page 143
- `Internal Authenticate Using DES` command on page 146
- `RSA Signature (Internal Auth)` command on page 164
- `RSA Signature Last` command on page 171
- `Select` command on page 177
- `Select EMV` command on page 181
- `SHA-1 Last` command on page 186

Increase

Use the `Increase` command to increase a number value stored in a cyclic EF record. For example, you could use the `Increase` command to add value to an electronic purse.

S/R: Send/Receive *(Case 4)*

AC – To increase a value in a particular EF, you must first satisfy the access condition specified for the `Increase` command in that file's input parameter string. (To find out which key number is required to satisfy an AUT or PRO AC, call the `Get AC Keys` command, described on page 133.)

When the `Increase` command executes, the card reads the value in the file's most recently written record, adds the specified amount, and stores the result in the file's oldest record (record #1). The updated record becomes the currently selected record. You can call a `Get Response` command to retrieve the updated value and the amount of the increase.

The following conditions apply to the use of the `Increase` command:

- The updated value must not exceed the maximum value for the record. (The maximum value is reached when the value of all record bytes = `FF`h.)
- The record length must be 3–252 bytes (`03`–`FC`h).

Command Format

Select the cyclic EF that contains the record value you want to increase and call the `Increase` command in the following format.

Command	CLA	INS	P1	P2	Lc
`Increase`	`F0`	`32`	`00`	`00`	`03` + *X*

Parameters

Name	Length	Value / Meaning
P1	1 B	`00`h
P2	1 B	`00`h
Lc	1 B	Length of the input data.
Input Data	3 + *X*B	The value to be added + the cryptogram (*X*), if the AC is PRO. If no PRO AC applies, *X* = 0.
SW1, SW2	2 B	Status word bytes the card returns.

Response Data Available

If you follow the `Increase` command immediately with a `Get Response` command, the card returns the data shown in the following table. Note that you can retrieve all or part of the record data. In either case, the data you retrieve always starts with the first byte in the record.

Bytes	Description of Data	Length
1–*lgth*	The increased value of the record	3–252 B
(*lgth* + 1) – (*lgth* + 3)	The value added	3 B

Status Words Returned

Hex Value	Meaning
`61`*xx*	Response data (in the amount of *xx* bytes) are available for return by a `Get Response` command.
`6283`	The currently selected EF is invalidated.
`6300`	PRO authentication failed because the cryptogram is wrong, or the data is not the correct length for PRO mode.
`6581`	Memory-related problem: The EEPROM may have failed.
`6703`	The Lc value (value to be deducted) does not match the record size or does not match the amount of input data. Enter `03`h.
`6CYYh` `(YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	The AC cannot be satisfied because either the cryptographic key, key, or key file is missing.
`6982`	Access Conditions not satisfied.
`6983`	The key required for PRO authentication is blocked.
`6985`	The PRO AC cannot be satisfied because the host application did not send a `Get Challenge` command to the card.
`6986`	A DF is currently selected. Select a cyclic EF.
`6A80`	A linear or transparent EF is currently selected. Select a cyclic EF.
`6B00`	Incorrect values entered for P1, P2, or both. (Enter `00`h for P1 and P2.)
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.

Hex Value	Meaning
`6F00`	Technical problem without a specified diagnostic.
`9850`	The maximum value has been reached, or the current value is too high to allow the specified increase. The card cannot perform the increase.

See Also:

- `Update Record` command on page 195
- `Decrease` command on page 107
- `Read Record` command on page 157
- `Get Response` command on page 138
- “Commands That Are Subject to Access Conditions,” on page 64

Internal Authenticate Using DES

Use the `Internal Authenticate Using DES` command to authenticate the card to the host application. Internal authentication is the first step in mutual authentication, which you use to ensure that host-to-card transactions are secure.

S/R: Send/Receive *(Case 4)*

AC – The access condition is CHV1. Before you can issue a successful `Internal Authenticate Using DES` command in a given context, you must establish appropriate access rights by executing a successful `Verify CHV` command (described on page 198).

Key Used — The DES key used for internal authentication is stored in an internal key file. The key is never passed between the card and the host application—only the plaintext challenge and its cryptogram are exchanged. You specify a key number, and the card uses the corresponding DES or 3DES key in electronic code book (EBC) mode.

Overview of Operation — Like the `External Authenticate Using DES` command, the `Internal Authenticate Using DES` command asks for proof that the card and the host share the same DES key. In external authentication, the host proves that it possesses a key stored on the card. In internal authentication, the card proves that it holds a key the host possesses. Internal authentication consists of these steps:

1. The host uses a `Verify CHV` command to establish access rights for internal authentication.
2. The host calls the `Internal Authenticate Using DES` command:

   The host sends the card a plaintext challenge and indicates which key to use from the relevant internal key file. Using the specified key, the card encrypts the challenge and returns a status word that indicates the length of the cryptogram that is available for retrieval.
3. The host sends a `Get Response` command to retrieve the cryptogram.

   The host decrypts the cryptogram. If the resulting string matches the original challenge, it proves that the card possesses the correct key and is trustworthy.

Command Format

Command	CLA	INS	P1	P2	Lc
`Internal Authenticate Using DES`	`C0`	`88`	`00`	*key nbr*	`08`

Parameters

Name	Length	Value / Meaning
P1	1 B	Internal key value to use, either: • `00`h = DES encryption key (first 8 bytes of DES or double-length 3DES key value), or • `01`h = DES decryption key (second 8 bytes of double-length 3DES key value).
P2	1 B	Key number: `00`–`0F`h (key 0–15). Number of the DES or 3DES key to use for internal authentication, located in the relevant internal key file.
Lc	1 B	`08`h: Length of the challenge host application sends the card.
Input Data	8 B	Plaintext challenge the host application sends the card.
SW1, SW2	2 B	Status word bytes the card returns.

Response Data Available

Bytes	Description of Data
1–6 *or* 1–8	Truncated (6-byte) or full (8-byte) DES cryptogram. SchlumbergerSema sets the cryptogram length during manufacturing, as appropriate for export.

Status Words Returned

Hex Value	Meaning
`6106`	Response data (in the amount of 6 bytes) are available for return by a `Get Response` command if the card is set for truncated 6-byte DES encryption.
`6108`	Response data (in the amount of 8 bytes) are available for return from a Full DES card by a `Get Response` command. A Full DES card is one that is pre-personalized for 8-byte DES encryption rather than truncated 6-byte DES encryption (as required for some export situations).
`6700`	Incorrect value entered for Lc.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	Either the key slot specified in P2 does not exist, no relevant internal key file exists, or the P2 value is greater than `0F`h.

Hex Value	Meaning
`6982`	The required CHV1 AC was not satisfied.
`6983`	The specified key is blocked because the key file was initialized or updated with unsupported values set for one or more of the RFU bytes.
`6985`	The key specified in the P2 parameter contains an algorithm ID and key value that are incompatible.
`6B00`	Unsupported values entered for P1, P2, or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.

See Also:

- `External Authenticate Using DES` command on page 122
- `Get Response` command on page 138
- “Digital Signatures for Internal Authentication,” on page 36

Invalidate

Use the `Invalidate` command to deactivate the currently selected elementary file. An invalidated file is unavailable to any command except `Select`, `Delete File`, or `Rehabilitate`. You typically invalidate a file to:

- Block access to a sensitive file when a security violation is suspected. The card administrator can make the file available again by calling the `Rehabilitate` command.
- Create a file during pre-personalization that is initially unavailable to the cardholder. The file becomes accessible if the user establishes the appropriate access rights.

S/R: Send *(Case 3)*

AC – To invalidate an elementary file (EF) located directly under the master file (MF), first satisfy the AUT access condition (AC). To invalidate an EF located under a DF other than the MF, first satisfy the AC (if any) set for the `Invalidate` command in the selected DF. (To find out which key number is required to satisfy an AUT AC, call the `Get AC Keys` command, described on page 133.)

NOTES

- *To find out whether a file is already invalidated, examine the status byte you retrieve by calling a `Dir Next` command or by selecting the file and calling a `Get Response` command.*
- *You cannot use the `Invalidate` command on a DF.*

Command Format

Select the EF you want to invalidate and call the `Invalidate` command in the following format.

Command	CLA	INS	P1	P2	Lc
`Invalidate`	`F0`	`04`	`00`	`00`	`00` + *X*

Parameters

Name	Length	Value / Meaning
P1	1 B	`00h`
P2	1 B	`00h`
Lc	1 B	Length of the input data.

Name	Length	Value / Meaning
Input Data	*X* B	The cryptogram (*X*), if the PRO AC applies. If no PRO AC applies, *X* = 0.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6283`	The currently selected EF is already invalidated.
`6300`	PRO authentication failed because the cryptogram is wrong.
`6581`	Memory-related problem: The EEPROM may have failed.
`67`*xx*	The value entered for Lc is unsupported or does not match the amount of data included. (This response applies to PRO mode commands.)
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	The AC cannot be satisfied because either the cryptographic key, CHV key, or key file is missing.
`6982`	The required AC was not satisfied because the key is invalid or no key was presented.
`6983`	The key required for PRO authentication is blocked.
`6985`	The PRO AC was not satisfied because the host did not send a `Get Challenge` command to the card.
`6986`	A DF is currently selected. Select an EF, then issue the command.
`6B00`	Incorrect values entered for P1, P2, or both. (Enter `00`h for P1 and P2.)
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: the selected EF is now invalidated.

See Also:

- `Rehabilitate` command on page 162
- "Commands That Are Subject to Access Conditions," on page 64

Logout AC

Use the `Logout AC` command to log out one or more previously logged-in access conditions (ACs) from the card.

S/R: Send *(Case 3)*

AC – Not applicable

The `Logout AC` command is designed for multi-application environments in which you have resources that are proprietary to some applications, but not to all. Let's say an application calls for the card user to verify a PIN (granting CHV2 AC rights), then calls the card administrator to establish AC rights. The application calls a `Logout AC` command to reset the user's AC status. As a result, the card administrator logs in without inheriting card user access rights. In this way, the application controls AC rights without resetting the card and interrupting the flow of events.

Command Format

Command	CLA	INS	P1	P2	Lc
`Logout AC`	`F0`	`22`	*access conditions*	`00`	`00`

Parameters

Name	Length	Value / Meaning
P1	1 B	AC or ACs to be reset: • `01`h = AUT (00000001 binary) • `02`h = CHV1 (0000010 binary) • `03`h = AUT + CHV1 (00000011 binary) • `04`h = CHV2 (00000100 binary) • `05`h = AUT + CHV2 (00000101 binary) • `06`h = CHV1 + CHV2 (00000110 binary) • `07`h = AUT + CHV1 + CHV2 (00000111 binary) Bit 0 controls the AUT AC, bit 1 controls CHV1, and bit 2 controls CHV2. Bits 3–7 are RFU. If you enter `00`h or a value greater than `07`h for P1, the command fails.
P2	1 B	`00`h (Must be null.)
Lc	1 B	`00`h (Must be null.)
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6700`	The value entered for Lc is incorrect. (Enter `00`h.)
`6B00`	The value entered for P1, P2, or both is incorrect.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The specified ACs are reset.

See Also:

- `External Authenticate Using DES` command on page 122
- `Verify CHV` command on page 198
- `Verify Key` command on page 200

Read Binary

Use the `Read Binary` command to read data in the currently selected transparent EF. (You cannot use this command with linear or cyclic EFs: Use the `Read Record` command to retrieve information from these types of files.)

S/R: Receive *(Case 2)*

AC – To read binary data in an EF, you must first satisfy the access condition specified for the `Read Binary` command in the currently selected EF's input parameter string. Note that the card does not support the PRO AC for the `Read Binary` command. (To find out which key number is required to satisfy an AUT AC, call the `Get AC Keys` command, described on page 133.)

You can read a maximum of 256 bytes of data by calling a `Read Binary` command. Specify the number of data bytes to retrieve and specify an offset to use as the starting point for retrieval. The frame of reference for the offset is big-endian and zero-based. In other words, the most significant byte (MSB) in the sequence is stored first, at the lowest storage address (`0000`). For example, to read bytes 12 through 15 in a file, enter `00`h for P1, `0B`h for P2, and `04`h for Le.

NOTE *You use this command only to read transparent elementary files. To determine whether a file is a transparent EF, select the file, then issue a `Get Response` command. Byte 7 of the response data for a transparent EF = `01`h.*

Command Format Select the transparent EF you want to read and call the `Read Binary` command in the following format.

Command	CLA	INS	P1	P2	Le
`Read Binary`	`C0`	`B0`	*offset MSB*	*offset LSB*	*lgth*

Parameters

Name	Length	Value / Meaning
P1	1 B	Most significant byte of offset for reading data
P2	1 B	Least significant byte of offset for reading data
Le	1 B	Length of the data to be returned (a value from `00` to `FF`h). Note: P3=00 returns 256 bytes.
Output Data	*lgth* B	Output data the card returns.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6283`	The currently selected EF is invalidated.
`6581`	Memory-related problem.
`6700`	The Le value is longer than the length of data from the defined offset to the end of the file.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of GetData request does not correspond to P3.
`6982`	The required AC was not satisfied because the key is invalid or no key was presented.
`6986`	A DF is currently selected. Select a transparent EF.
`6A80`	A linear or cyclic EF is currently selected. Select a transparent EF.
`6B00`	The specified offset is outside the boundaries of the EF.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The card returned the binary data from the transparent EF.

See Also:

- `Read Binary Enciphered` command on page 155
- `Read Record` command on page 157
- "Commands That Are Subject to Access Conditions," on page 64

Read Binary Enciphered

Use the `Read Binary Enciphered` command to encrypt and retrieve data from the currently selected transparent EF. For example, you can use this command to retrieve sensitive information over a network or the Internet. This command works only with transparent EFs, not with linear or cyclic EFs.

S/R: Receive *(Case 2)*

AC – To use this command, you must first satisfy the access condition (AC) specified for the `Read Binary Enciphered` command in the input parameter string of the currently selected EF. The card does not support the PRO AC for this command. (To find out which key number is required to satisfy an AUT AC, call the `Get AC Keys` command, described on page 133.)

Encryption Key — To encrypt the data, the card uses the key number specified in the input parameters of the currently selected file. The key number refers to a key in the relevant external key file. (The `Read Binary Enciphered` key is specified in the MSN of byte 10 in the `Create File` structure data. This key number applies to a DES or 3DES key in the relevant external key file.) To retrieve the key number, call a `Dir Next` command for the target EF and examine the MSN of byte 13.

Defining the Offset — Use P1 and P2 to specify the offset—the starting point of the data to be returned. The frame of reference for the offset is big-endian and zero-based. In other words, the most significant byte (MSB) in the sequence is stored first, at the lowest storage address (`0000`). For example, to read bytes 12 through 20 in a file, enter `00`h for P1, `0B`h for P2, and `09`h for Le.

Use Le to specify the number of data bytes. You can retrieve a maximum of 232 bytes of data, which must be evenly divisible into 8-byte blocks.

Command Format Select the transparent EF whose data you want to retrieve and call the `Read Binary Enciphered` command in the following format.

Command	CLA	INS	P1	P2	Le
`Read Binary Enciphered`	`04`	`B0`	*offset MSB*	*offset LSB*	*lgth*

Parameters

Name	Length	Value / Meaning
P1	1 B	MSB of offset that defines the retrieval starting point.
P2	1 B	LSB of offset that defines the retrieval starting point.
Le	1 B	Number of bytes to retrieve (`08–E8`h octet, or 8–232 bytes decimal).
Output Data	*lgth* B	Enciphered data the card returns (maximum = 248 bytes).
SW1, SW2	2 B	Status words returned by the card.

Status Words Returned

Hex Value	Meaning
`6283`	The currently selected EF is invalidated.
`6700`	The Le value is outside the range of retrievable data. Data not divisible by 8. Enter an octet value of `08–F8`h.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of GetData request does not correspond to P3.
`6981`	The AC cannot be satisfied because either the card contains no relevant external key file or the specified AC key is missing.
`6982`	The required AC was not satisfied because the key is invalid or no key was presented.
`6983`	Key blocked
`6986`	A DF is currently selected. Select a transparent EF.
`6A80`	A linear or cyclic EF is currently selected. Select a transparent EF.
`6B00`	Incorrect values entered for P1, P2, or both. (The specified offset is outside the boundaries of the EF.)
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command was successful: The card enciphered and returned the binary data from the transparent EF.

See Also:

- `Read Binary` command on page 153
- `Read Record` command on page 157
- "Commands That Are Subject to Access Conditions," on page 64

Read Record

Use the `Read Record` command to read data in one of the records in the currently selected linear or cyclic EF. You cannot use the `Read Record` command with transparent EFs—use `Read Binary` to retrieve data from this type of file.

S/R: Receive *(Case 2)*

AC – To issue a successful `Read Record` command, you must first satisfy the access condition (AC) specified for this command in the input parameter string of the currently selected EF. Note that the card does not support the PRO AC for this command. (To find out which key number is required to satisfy an AUT AC, call the `Get AC Keys` command, described on page 133.)

NOTE *You cannot read data in an invalidated file.*

Specifying a Record to Read

Use P2 and P1 to specify the record selection mode and record to read.

Linear EF — Specify a record in either of these ways:

- Use the absolute mode (P2 = `04`h) and choose a record by number (in P1), or
- Use the current mode (P2 = `00`–`03`h, P1 = `00`h) and choose the *first*, *last*, *next*, or *previous* record in the file.

  To update the first or last record in the file, set P2 to `00`h (current mode, first record) or `01`h (current mode, last record). You can then call the command again to update the *next* or *previous* record. If the first `Update Record` call is for the *next* record, the card updates the first record in the file. If the first `Update Record` call is for the *previous* record, the card updates the last record in the file.

Cyclic EF — Use the current mode (P2 = `00`–`03`h, P1 = `00`h) and choose the *first*, *last*, *next*, or *previous* record in the file, as described above for linear EFs.

Command Format

Select the linear or cyclic EF that contains the record you want read and call the `Read Record` command in the following format.

Command	CLA	INS	P1	P2	Le
`Read Record`	`C0`	`B2`	*rec nbr*	*mode*	*lgth*

Parameters

Name	Length	Value / Meaning
P1	1 B	Record number: • `00`h = No record number specified (Current mode) • `01-`*xx*h = Number of the linear record to read (Absolute mode, P2 = `04`h)
P2	1 B	Record selection mode: • `00`h = First record, current mode (P1 = `00`h). Retrieves first record in the file. • `01`h = Last record, current mode (P1 = `00`h) Retrieves last record in the file. • `02`h = Next record, current mode (P1 = `00`h) Retrieves the record after the currently selected one. If no record is selected, updates the first record in the file. • `03`h = Previous record, current mode (P1 = `00`h) Retrieves the record before the currently selected one. If no record is selected, updates the last record in the file. • `04`h = Absolute mode (if P1 is a non-null value), *or* Current record (if P1 = `00`h). (The current record is the record currently selected by the record pointer.) The record pointer does not change location.
Le	1 B	Length of the data string you want to read.
Output Data	*lgth* B	Record data the card returns to the host.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6281`	Data may be corrupt.
`6283`	The currently selected EF is invalidated.
`6581`	Memory-related problem.
`67`*xx*	The value entered for Le exceeds the length of the specified record. Enter the value that appears in place of *xx*.
`6981`	The AC cannot be satisfied because either the cryptographic key, CHV key, or key file is missing.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of GetData request does not correspond to P3.
`6982`	The required AC was not satisfied because the key is invalid or no key was presented.
`6986`	A DF is currently selected. Select a linear or cyclic EF.
`6A80`	A transparent EF is currently selected. Select a linear or cyclic EF.
`6A83`	The specified record is out of range or the record ID is not found. (For example, you may have already read all the records in the file.)
`6B00`	Incorrect values entered for P1, P2, or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command was successful, and the card returned the specified record data.

See Also:

- `Update Record` command on page 195
- `Seek` command on page 175
- "Commands That Are Subject to Access Conditions," on page 64

Read Record EMV

NOTE *The Read Record EMV command is not available for the 32K+e-gate card or the Cryptoflex 32K card version 1.*

If you select an EMV application, then call a `Read Record` command, the card interprets the command as a `Read Record EMV` command. You can use the `Read Record EMV` command to read data in one of the following types of linear EFs:

- Call the command with an application dedicated file (ADF) selected in order to retrieve the only record of the application elementary file (AEF) dedicated to the ADF.
- Call the command with a payment system environment (PSE) selected in order to retrieve the only record of the EMV Dir file (FID 3F04).

S/R: Receive *(Case 2)*

AC – Not applicable

The `Read Record EMV` command is similar to the `Read Record` command, except that you use the P1 parameter to identify the number of the record you want to read. In the P2 parameter, specify the file to read (by its short file identifier, or SFI) in bits 7–3, and set bits 2–0 to 100 binary.

Command Format

Call a `Select EMV` command to select the PSE or ADF of an EMV application, then call the `Read Record EMV` command in the following format.

Command	CLA	INS	P1	P2	Le
`Read Record EMV`	`C0`	`B2`	*rec num*	*ref control*	*lgth*

Parameters

Name	Length	Value / Meaning
P1	1 B	`01h` = Record number.
P2	1 B	Enter a hexadecimal value derived from an 8-bit binary number formatted as follows: • Bits 7–3 = Short file identifier (SFI), and • Bits 2–0 = 100 (Bit 2 = 1, bit 1 = 0, and bit 0 = 0).
Le	1 B	Length of the record data.

Name	**Length**	**Value / Meaning**
Output Data	*lgth* B	Record data returned by the card.
SW1, SW2	2 B	Status words returned by the card.

Status Words Returned

Hex Value	**Meaning**
`6A86`	Incorrect values entered for P1, P2, or both.
`6C`*xx*	The currently selected EF is invalidated.
`6985`	No EMV file is currently selected.
`6A82`	The currently selected file type is not supported for this command. Select an EMV application (PSE or ADF) and reissue the command.
`9000`	The command was successful: The card returned the specified record data.

See Also:

- `Select EMV` command on page 181
- "Commands That Are Subject to Access Conditions," on page 64

Rehabilitate

Use the `Rehabilitate` command to reactivate a currently selected EF that has been invalidated. Once the file is rehabilitated, you can execute commands on the file and its contents.

S/R: Send *(Case 3)*

AC – If you want to rehabilitate an elementary file (EF) located directly under the master file (MF), you must first satisfy the AUT access condition (AC). If you want to rehabilitate an EF located under a DF other than the MF, you must first satisfy the AC (if any) that is set for the `Rehabilitate` command in the selected DF. (To find out which key number is required to satisfy an AUT or PRO AC, call the `Get AC Keys` command, described on page 133.)

NOTES

- *To find out whether a file is invalidated, examine the status byte you retrieve by calling a `Dir Next` command or by selecting the file and calling a `Get Response` command.*
- *You cannot use the `Rehabilitate` command to change the validation status of a DF.*

Command Format

Select an invalidated EF and call the `Rehabilitate` command in the following format.

Command	CLA	INS	P1	P2	Lc
`Rehabilitate`	`F0`	`44`	`00`	`00`	`00` + *X*

Parameters

Name	Length	Value / Meaning
P1	1 B	`00`h
P2	1 B	`00`h
Lc	1 B	Length of the input data, `00`h, unless the AC is PRO. (*X* =Cryptogram length if AC = PRO, or 0 if no PRO AC.)
Input Data	*X* B	The cryptogram for a PRO command.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6283`	The currently selected EF is not invalidated.
`6300`	PRO authentication failed because the cryptogram is wrong.
`6581`	Memory-related problem: The EEPROM may have failed.
`67`*`xx`*	The value entered for Lc is unsupported or does not match the amount of data included. (This response is currently known to be valid only for PRO mode commands.)
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	The AC cannot be satisfied because either the cryptographic key, CHV key, or key file is missing.
`6982`	The required AC was not satisfied because the key is invalid or no key was presented.
`6983`	The key required for PRO authentication is blocked.
`6985`	The AUT AC was not satisfied because the host did not send a `Get Challenge` command to the card.
`6986`	No EF is selected. (You must select an invalid EF to perform this command.)
`6B00`	Incorrect values entered for P1, P2, or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The card rehabilitated the specified file.

See Also:

- `Invalidate` command on page 149
- "Commands That Are Subject to Access Conditions," on page 64

RSA Signature (Internal Auth)

Use the `RSA Signature (Internal Auth)` command to encrypt data as an RSA signature on the card. The computation result is *not* stored in EEPROM on the card. (The result is stored temporarily in RAM to be recoverd by a following `Get Response` instuction.) The following tables shows the amount of data you can encrypt and key sizes you can use. *(For information about using a 2048-bit key, see page 167.)*

	512-bit key	**768-bit key**	**1024-bit key**
Amount of Input Data/Length of Signature	64 B	96 B	128 B

S/R: Send/Receive *(Case 4)*

AC – The access condition is CHV1. Before you can issue a successful `RSA Signature` command, you must establish appropriate access rights by executing a successful `Verify CHV` command (described on page 198).

The card uses the specified private key to sign data (typically a hash digest) from the host application. The length of the modulus is 512, 768, or 1024 bits, and produces a cryptogram that is 64, 96, or 128 bytes long, respectively. Call the `Get Response` command immediately after the `RSA Signature` command to retrieve the cryptogram, which you use as a signature. The recipient uses a matching public key to verify the cryptogram.

- *The input string value must be smaller than the modulus of the RSA algorithm used for the signature, or the output will be incorrect. Apply padding to ensure that the input string is an appropriate length. You can use one of the padding types described in standards such as ISO 9796 or PKCS #1.*
- *A private key file (0012) and a public one (1012) must be locally available and have keys in place, or the* `RSA Signature` *command cannot succeed. If the card does not find this file and the specified key in the currently selected directory, it does not search for it. (Relevance does not apply to RSA key files.)*

An internal RSA signature verification is performed to ensure the coherence of the signature in order to avoid a potential attack on the private RSA key.

To perform this internal RSA signature verification, the public exponent is needed. The public exponent is found in the public key file at the index corresponding to the key number. If the public exponent cannot be found in the public key file, a default public exponent is used. The default public exponent is 10001 (01000100 in LSB first format).

This command is similar to `Internal Authenticate Using DES`. Use Lc to specify the key length and P2 to specify which key to use in the private key file. The card returns the status word `61`*xx* to indicate the number of bytes available for retrieval.

The public exponent is used in the RSA signature operation. *For more information about the public exponent, see page 128.*

Command Format

Select the directory that contains the RSA key you want to use for the encryption, then call the `RSA Signature (Internal Auth)` command as described in the following table.

Command	CLA	INS	P1	P2	Lc
`RSA Signature (Internal Auth)`	`C0`	`88`	`00`	*key nbr*	`40/60/80`

Parameters

Name	Length	Value / Meaning
P1	1 B	`00h`
P2	1 B	Number of key to use in the private key file: A value from `00–0E`h (key 1–15).
Lc	1 B	Length of cryptogram: `40`, `60`, or `80`h (64, 96, or 128 B).
Input Data	64 / 96 / 128 B	Hash or other data to be encrypted, in LSB-first format.
SW1, SW2	2 B	Status word bytes the card returns.

Response Data Available

If you follow the `RSA Signature (Internal Auth)` command immediately with a `Get Response` command, the card returns the data described in the following table.

Key Format	Bytes	Description of Data	Length
512-bit	1–64	Cryptogram for 512-bit RSA signature (LSB first)	64 B
768-bit	1–96	Cryptogram for 768-bit RSA signature (LSB first)	96 B
1024-bit	1–128	Cryptogram for 1024-bit RSA signature (LSB first)	128 B

Status Words Returned

Hex Value	Meaning
`6140`	The command succeeded. A 64-byte cryptogram is available for return by a `Get Response` command.
`6160`	The command succeeded. A 96-byte cryptogram is available for return by a `Get Response` command.
`6180`	The command succeeded. A 128-byte cryptogram is available for return by a `Get Response` command.
`6700`	Incorrect Lc value entered: Either the value specified is not `40`, `60`, or `80`h, or it does not match the length of the input data.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	Either the key specified in P2, or the private key file itself was not found.
`6982`	The required CHV1 AC was not satisfied.
`6983`	End of RSA Key File reached, key not found.
`6A80`	Private/public key file is not a transparent EF.
`6B00`	Unsupported P1 value specified. Enter `00`h for P1.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.

Commands Available for RSA Signatures

The Cryptoflex card supports several commands for processing RSA signatures. The following table shows the available options for different key sizes.

Key Size	Size of Signature / Input Data	Command Options
512-bit	64-byte signature/ data	• `RSA Signature (Internal Auth)`, • `RSA Signature Intermediate` *(1 or more)* + 1 `RSA Signature Last`, or • `RSA Signature Last`
768-bit	96-byte signature/data	
1024-bit	128-byte signature/data	
2048-bit	256-byte signature/data	• `RSA Signature Intermediate` *(1 or more)* + 1 `RSA Signature Last`

RSA Signature Processing

You must satisfy the CHV1 access condition (AC), and the applicable AC (if any). An RSA Chinese Remainder Theorem algorithm is used to generate a cryptogram from the data string, using the private key stored in the local private key file. The RSA computation is $C = X^S \bmod N$, where:

C	=	Cryptogram
X	=	Data to be signed, or encrypted data to decrypt
S	=	Private exponent of the private key file (KS_{PRI} in the $EF_{RSA\text{-}PRI}$)
N	=	Public modulus of the private key file ($EF_{RSA\text{-}PRI}$)

Retrieving the Cryptogram – Retrieve the resulting cryptogram by calling the `Get Response` command.

See Also:

- `Get Response` command on page 138
- `RSA Signature Intermediate` command on page 168
- `RSA Signature Last` command on page 171
- `Generate RSA Keys` command on page 127

RSA Signature Intermediate

Use the `RSA Signature Intermediate` command as a first or intermediate step to generate an RSA signature on the card. In combination with the `RSA Signature Last` command, you can use this command to generate a 2048-bit signature.

S/R: Send *(Case 3)*

AC – The access condition is CHV1. Before you can issue a successful `RSA Signature Intermediate` command in a given context, you must establish appropriate access rights by executing a successful `Verify CHV` command (described on page 198).

- *If the input string is the wrong length, apply padding to make it the appropriate length. You can use one of the padding types described in standards such as ISO 9796 or PKCS #1.*
- *A private key file (0012) and a public one (1012) must be locally available and have keys in place, or the `RSA Signature` command cannot succeed. If the card does not find this file and the specified key in the currently selected directory, it does not search for it. (Relevance does not apply to RSA key files.)*
- *You must follow an `RSA Signature Intermediate` command with another `RSA Signature Intermediate` command or an `RSA Signature Last` command.*

Encryption Key — Use P2 to specify a key in the local private key file to use for the encryption. To generate a 256-byte signature, you must use one or more `RSA Signature Intermediate` commands and a follow-up `RSA Signature Last` command. Use the same key for the `RSA Signature Intermediate` and `RSA Signature Last` command.

NOTE *To generate a signature with an RSA key that is 1024-bit or smaller, call an `RSA Signature (Internal Auth)` command (or an `RSA Signature Last` command by itself).*

Amount of Input Data — You can store 1–255 bytes of data on the card with a single `RSA Signature Intermediate` command. Follow with an `RSA Signature Last` command to send an additional block of data (0–255 bytes). The total amount of data sent by both types of commands must equal 64 bytes (512-bit key), 96 bytes (768-bit key), 128 bytes (1024-bit key), or 256 bytes (2048-bit key).

You can send multiple `RSA Signature Intermediate` commands if you like, but only one is needed with the `RSA Signature Last` command to store a full 256-byte signature.

Command Format

Select the directory that contains the RSA key you want to use for the encryption, then call the `RSA Signature Intermediate` command in the following format.

Command	CLA	INS	P1	P2	Lc
`RSA Signature Intermediate`	`10`	`88`	`00`	*key nbr*	*lgth*

Parameters

Name	Length	Value / Meaning
P1	1 B	`00h`
P2	1 B	`00-0E`h (key 1–15): Number of the key to use from the local private key file. If you call multiple `RSA Signature Intermediate` commands, this value must match the P2 value of the preceding command.
Lc	1 B	Length of the input data, which is limited by the key size used, as shown in the Input Data description that follows.
Input Data	0–255 B	Portion of data to be encrypted, entered LSB first. Total data length (per command or per sum of all `RSA Intermediate/ Last` commands) must equal: • *512-bit key* = 01–40h (1–64 bytes) data, • *768-bit key* = 01–60h (1–96 bytes) data, • *1024-bit key* = 01–80h (1–128 bytes) data, or • *2048-bit key* = 01–FFh (1–255 bytes) of data.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6700`	Incorrect Lc value entered: The specified value is unsupported or does not match the length of the input data.
`6CYYh (YY=P3)`	**USB Mode only**: wLength of SendData request does not correspond to P3
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.

Hex Value	Meaning
6981	Either the key specified in P2, or the private key file itself does not exist.
6982	The required CHV1 AC was not satisfied.
6B00	One or both of these errors apply: A value other than 00h is specified for P1, or the value for P2 (key number) does not match the P2 value in the preceding RSA Signature Intermediate command.
6D00	Unknown command instruction value entered.
6E00	Incorrect command class value entered.
6F00	Technical problem without a specified diagnostic.
9000	The command succeeded: The data block is stored on the card. The card is ready to receive an RSA Signature Last command or another RSA Signature Intermediate command.

See Also:

- RSA Signature Last command on page 171
- Table of RSA signature command options on page 167
- "RSA Signature Processing," on page 167
- RSA Signature (Internal Auth) command on page 164
- Generate RSA Keys command on page 127

RSA Signature Last

Use the `RSA Signature Last` command (alone or in combination with an `RSA Signature Intermediate` command) to generate an RSA signature on the card. Using the two commands together, you can generate a 2048-bit signature.

S/R: Send/Receive *(Case 4)*

AC – The access condition is CHV1. Before you can issue a successful `RSA Signature Last` command in a given context, you must establish appropriate access rights by executing a successful `Verify CHV` command (described on page 198).

- *A private key file (0012) and a public one (1012) must be locally available and have keys in place or the `RSA Signature Last` command cannot succeed. If the card does not find this file and the specified key in the currently selected directory, it will not search for it in upper directories. (Relevance does not apply to RSA key files.)*
- *The input string value must be smaller than the modulus of the RSA algorithm used for the signature, or the output will be incorrect. Apply padding to ensure that the input string is an appropriate length. Use one of the padding types described in standards such as ISO 9796 or PKCS #1.*

An internal RSA signature verification is performed to ensure the coherence of the signature in order to avoid a potential attack on the private RSA key.

To perform this internal RSA signature verification, the public exponent is needed. The public exponent is found in the public key file at the index corresponding to the key number. If the public exponent cannot be found in the public key file, a default public exponent is used. The default public exponent is 10001 (01000100 in LSB first format).

Encryption Key — Use P2 to specify a key in the local private key file to use for encryption. This must be the same key the preceding `RSA Signature Intermediate` command used.

NOTE *To generate a signature with an RSA key that is 1024-bit or smaller, call an `RSA Signature (Internal Auth)` command (or an `RSA Signature Last` command by itself).*

Public Exponent — The public exponent is used during the signature operation.

Amount of Input Data — You can store 0–255 bytes of data on the card with an `RSA Signature Last` command. If you send one or more `RSA Signature Intermediate` commands and an `RSA Signature Last` command, the total amount of data must equal 64 bytes (512-bit key), 96 bytes (768-bit key), 128 bytes (1024-bit key), or 256 bytes (2048-bit key). Once the card has stored the final data block, it performs the signature.

Retrieving the Signature – If the command is successful, the card returns the status `61`*xx* to indicate the number of bytes available for retrieval. Use the *xx* value in a `Get Response` command immediately afterward to retrieve the cryptogram.

Command Format

Use the following format to call this command—immediately after a final `RSA Signature Intermediate` command or in standalone form—after you navigate to the directory that contains the RSA keys you want to use for the operation.

Command	CLA	INS	P1	P2	Lc
`RSA Signature Last`	`00`	`88`	`00`	*key nbr*	*lgth*

Parameters

Name	Length	Value / Meaning
P1	1 B	`00h`
P2	1 B	`00-0E`h (key 1–15): Number of the key to use from the local private key file. This value must match the P2 value of the preceding `RSA Signature Intermediate` command.
Lc	1 B	Length of the input data, which is limited by the key size used, as shown in the Input Data description that follows.
Input Data	0–255 B	All or last block of data to be encrypted, entered LSB-first. Total data length (per command or per sum of all `RSA Intermediate` / `Last` commands) must equal: • *512-bit key* = 00–40h (0–64 bytes) data, • *768-bit key* = 00–60h (0–96 bytes) data, • *1024-bit key* = 00–80h (0–128 bytes) data, or • *2048-bit key* = 00–FFh (0–255 bytes) of data.
SW1, SW2	2 B	Status word bytes the card returns.

Response Data Available

If you follow the `RSA Signature Last` command immediately with a `Get Response` command, the card returns the data described in the following table.

Key Size	Bytes	Description of Data	Length
512-bit	1–64	Cryptogram for 512-bit RSA signature (LSB-first format)	64 B
768-bit	1–96	Cryptogram for 768-bit RSA signature (LSB-first format)	96 B
1024-bit	1–128	Cryptogram for 1024-bit RSA signature (LSB-first format)	128 B
2048-bit	1–256	Cryptogram for 2048-bit RSA signature (LSB-first format)	256 B

Status Words Returned

Hex Value	Meaning
`6100`	The command succeeded. A 256-byte (`100`h) cryptogram is available for return by a `Get Response` command.
`6140`	The command succeeded. A 64-byte cryptogram is available for return by a `Get Response` command.
`6160`	The command succeeded. A 96-byte cryptogram is available for return by a `Get Response` command.
`6180`	The command succeeded. A 128-byte cryptogram is available for return by a `Get Response` command.
`6700`	The specified Lc value incorrect: Either it is not `40`, `60`, `80`, `00`h, or it does not match the length of the input data.
`6CYYh` `(YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	Key not found: No local private key file or key number not specified.
`6982`	The required CHV1 AC was not satisfied.
`6983`	Key not found: The end of RSA key file was reached without finding the specified key.
`6A80`	Private/public key file is not a transparent EF.
`6B00`	One or more of these errors apply: • Unsupported value specified for P1. Enter `00`h. • Unsupported value specified for P2 (key number). • P2 value is different from the P2 value in the preceding `RSA Signature Intermediate` command.
`6D00`	Unknown command instruction value entered.

Hex Value	Meaning
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.

See Also:

- `RSA Signature Intermediate` command on page 168
- `RSA Signature (Internal Auth)` command on page 164
- Table of RSA signature command options on page 167
- "RSA Signature Processing," on page 167
- `Generate RSA Keys` command on page 127

Seek

Use the `Seek` command to search for a specified string in the records in the currently selected linear EF. You cannot use this command to search cyclic EFs.

S/R: Send *(Case 3)*

AC – To issue a successful `Seek` command, you must first satisfy the access condition (AC) specified for the command in the currently selected EF's input parameter string. The card does not support the PRO AC for the `Seek` command. (To find out which key number is required to satisfy an AUT AC, call the `Get AC Keys` command, described on page 133.)

The card searches each record starting from the offset you specify in P1. The card continues to search until it finds a match or reaches the end of the file. If the `Seek` operation succeeds, the record that contains the string becomes the currently selected record. You can follow with the `Read Record` command to retrieve the contents of the record.

If the `Seek` operation fails, the record pointer location is unchanged. If the pointer was undefined before the operation, it remains undefined.

Command Format

Select a linear EF and call the `Seek` command in the following format.

Command	CLA	INS	P1	P2	Lc
`Seek`	`F0`	`A2`	*offset*	*mode*	*lgth*

Parameters

Name	Length	Value / Meaning
P1	1 B	Offset to define the first byte to search in the record. The offset is zero-based. For example, specify an offset of `00`h to start the search with byte 1 of the record.
P2	1 B	Search mode: • `00`h = Start at the beginning of the file, or • `02`h = Start from the next record.
Lc	1 B	Length of the search string appended as input data.
Input Data	*lgth* B	Search string (bytes 1–*lgth*). The search string must be no larger than the record size.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6281`	Data may be corrupt.
`6283`	The currently selected EF is invalidated.
`6700`	The value entered for Lc does not match the search string length or is longer than the record length.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	The AC cannot be satisfied because either the cryptographic key, CHV key, or key file is missing.
`6982`	The required AC was not satisfied because the key is invalid or no key was presented.
`6986`	A DF is currently selected. Select a linear EF.
`6A80`	Either the pattern is not found, or a cyclic EF or transparent EF is currently selected. Select a linear EF.
`6B00`	One or both of these errors apply: The P1 value is incorrect (the specified offset is outside of the EF) or an unsupported value is specified for P2.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: A record that contains the search string was located and is now selected as the current record.

See Also:

- `Read Record` command on page 157
- “Commands That Are Subject to Access Conditions,” on page 64

Select

Use the `Select` command to select an EF or DF by supplying its file ID. Most commands are context-sensitive: You must establish the appropriate context by selecting a file or its parent before you can perform operations on the file.

S/R: Send/Receive *(Case 4)*

AC – Not applicable

NOTE *Unlike most commands, you can execute a `Select` command on any file, regardless of its validation status.*

When you reset or insert a card into the reader, the master file is selected by default. A successful `Select` command makes the specified file the current file. If the current file is a linear EF, the record pointer is undefined. If the current is a cyclic EF, the current record is the most recently written one.

You can use the `Select` command to navigate to any of the following files:

- Any EF or DF located in the current DF
- The parent DF of the currently selected DF or EF
- The master file (3F00)

Return Data – A successful `Select` command returns a data length in the status word `61`*xx*. (The data length appears in place of the *xx* byte.) Use the *xx* length in a `Get Response` command to retrieve information about the file. (For information about data you can retrieve, see the tables that follow, starting on page 178.)

Command Format

Command	CLA	INS	P1	P2	Lc
`Select`	`C0`	`A4`	`00`	`00`	`02`

Parameters

Name	Length	Value / Meaning
P1	1 B	`00`h
P2	1 B	`00`h
Lc	1 B	`02`h: Length of the input data (the file ID).
Input Data	2 B	ID of the file you want to select.
SW1, SW2	2 B	Status word bytes the card returns.

Response Data Available *(DF Selected)*

If you select a DF and immediately call a `Get Response` command, you can retrieve the data in the following table. Note that the amount of potential response data is affected by the presence or absence of relevant EF_{CHV2} files.

Bytes	Description of Data	Length
1–2	RFU	2 B
3–4	Number of unused EEPROM bytes available in the DF	2 B
5–6	File ID	2 B
7	File type (`38`h = DF)	1 B
8–11	ACs (See bytes 8–11 of the `Create File` command on page 31.)	4 B
12	File status: • `01`h = Activated • `00`h = Invalidated	1 B
13	Length of bytes to follow: • `05`h = No CHV data • `07`h = CHV1 data • `09`h = CHV1 and CHV2 data	1 B
14	Directory characteristics: If bit 8 (LSB) = 0, no valid, relevant CHV1 is present. If bit 8 (LSB) = 1, a relevant, active CHV1 is present.	1 B
15	Number of EFs under the current DF	1 B
16	Number of DFs under the current DF	1 B
17	Number of PINs and unblock CHV PINs: • `00`h = No relevant CHV files present • `02`h = CHV1 and unblock CHV1 present • `04`h = CHV1, unblock CHV1, CHV2, and unblock CHV2 present	1 B
18	RFU (`00`h)	1 B
19	Number of remaining CHV1 attempts, if a relevant EF_{CHV1} is present	1 B
20	Number of remaining unblock CHV1 attempts, if a relevant EF_{CHV1} is present	1 B

Bytes	Description of Data	Length
21[1]	CHV2 key status (Available only if the DF has a relevant EF_{CHV2}): bits 0–3 = Number of remaining CHV2 attempts bits 4–6 = RFU bit 7 = Activation status, either: • 0 = CHV2 key is invalidated • 1 = CHV2 key is active.	1 B
22[1]	CHV2 unblocking key status (Available only if the DF has a relevant EF_{CHV2}): bits 3–0 = Number of remaining unblock CHV2 attempts bits 4–6 = RFU bit 7 = Activation status: • 0 = CHV2 unblocking key is absent or invalidated. • 1 = CHV2 unblocking key is present and active.	1 B
23[1]	RFU	1 B

1 Bytes 21–23 of the response data are available only if the card contains a relevant EF_{CHV2}. Otherwise, 20 bytes of data are available. If the card does not contain a relevant EF_{CHV1}, not all of the 20 bytes contain useful data.

Response Data Available *(EF Selected)*

If you select an EF and immediately call a `Get Response` command, you can retrieve the data in the following table.

Bytes	Description of Data	Length
1–2	RFU	2 B
3–4	File size (body only)	2 B
5–6	File ID	2 B
7	File type: `01`h = Transparent EF `02`h = Fixed-length linear EF `04`h = Variable-length linear EF `06`h = Cyclic EF	1 B
8–11	ACs (See bytes 8–11 of the `Create File` command on page 32.)	4 B
12	File status: `01`h = Activated `00`h = Invalidated	1 B

Bytes	Description of Data	Length
13	Length of the data to follow (`01`h or `02`h)	1 B
14	RFU	1 B
15	Length of records, if the file is a fixed-length linear EF	1 B

Status Words Returned

Hex Value	Meaning
`61`*xx*	The file is selected and *xx* bytes of response data are available for return by a `Get Response` command.
`6281`	The data may be corrupted.
`6702`	The Lc value entered does not match the length of the file ID. Enter `02`h.
`6CYYh` `(YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6A82`	The file ID specified in Lc was not found.
`6B00`	Incorrect values entered for P1, P2, or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.

See Also: `Get Response` command on page 138

Select EMV

NOTE *The Select EMV command is not available for the 32K+e-gate card or the Cryptoflex 32K card version 1.*

Use the `Select EMV` command to select a a Europay/MasterCard/Visa (EMV) application by specifying its AID. After you select the application, you can send commands to it.

S/R: Send/Receive *(Case 4)*

AC – Not applicable

The card selects the dedicated file EF-DIR, located under the MF, then uses the AID you specify in Lc to select the corresponding application file and its contents. The card finds the application's DF file ID in the ER FIR file, which contains a table of AIDs and their corresponding FIDs.

After you select an EMV application, the application receives the commands you issue. This behavior continues until you execute a standard `Select` command (either successful or unsuccessful), or you reset the card. From this point on, commands are no longer directed to the EMV application.

NOTE *For more information about EMV applications and standards, see the Europay International website at www.europay.com.*

Command Format

Command	CLA	INS	P1	P2	Lc
`Select EMV`	`00`	`A4`	`04`	`00`	`05 — 10`

Parameters

Name	Length	Value / Meaning
P1	1 B	`04h`
P2	1 B	`00h`
Lc	1 B	Length of the AID: `05–10h` (5–16 B).
Input Data	5–16 B	AID of the application you want to select.
SW1, SW2	2 B	Status words returned by the card.

Response Data Available *(with ADF Selected)*

If you select an EMV application dedicated file (ADF), then immediately call a `Get Response` command, the following data (in Tag/Length/Value, TLV format) is available for return. (You must specify `00`h as the value for the CLA byte of the `Get Response` command.)

Bytes	Description of Data	Length
1	*Tag* – of File Control Information (FCI) template: `6F`h	1 B
2	*Length* – of FCI template: L (Σ data). Length of the remaining data	1 B
3	*Tag* – of DF name (AID): `84`h	1 B
4	*Length* – of DF name: `05`–`10`h (referred to as *X*)	1 B
4+*X*	*Value* – of DF name (AID)	5–16 B
5+*X*	*Tag* – of FCI proprietary template: `A5`h	1 B
5+*X*	*Length* – of FCI proprietary template. either: • `00`h = Single application on the card • `03`h = Multiple applications on the card	1 B
7+*X*	*Value: Tag* – of application priority indicator: `87`h (if the card contains multiple applications)	1 B
8+*X*	*Value: Length* – of application priority indicator: `01`h (if the card contains multiple applications)	1 B
9+*X*	*Value: Value* – Application priority indicator (if the card contains multiple applications)	1 B

Response Data Available *(with PSE Selected)*

If you select an EMV Payment System Environment (PSE), then immediately call a `Get Response` command, the following TLV-formatted data is available for return. (You must specify `00`h as the value for the CLA byte of the `Get Response` command, or the command will fail.)

Bytes	Description of Data	Length
1	*Tag* – of FCI template: `6F`h	1 B
2	*Length* – of FCI template: L (Σ data). Length of the remaining data.	1 B
3	*Tag* – of DF name (AID): `84`h	1 B
4	*Length* – of DF name: `0E`h	1 B
5–18	*Value* – of DF name: 1PAY.SYS.DDF01	14 B
19	*Tag* – of FCI proprietary template: `A5`h	1 B
20	*Length* – of FCI proprietary template: `03`h	1 B
21	*Value: Tag* – of Short File Identifier (SFI) of the directory elementary file: `88`h	1 B
22	*Value: Length* – of SFI of the directory elementary file: `01`h	1 B
23	*Value: Value* – of SFI of the directory elementary file	1 B

Status Words Returned

Hex Value	Meaning
`61`*xx*	The file and corresponding application are selected, and *xx* bytes of response data are available for return by a `Get Response` command.
`6700`	The specified Lc value does not match the file name data or is an unsupported length or the AID included as input data is an unsupported length. (The AID must be between `5` and `10`h, or 5–16 characters.)
`6A82`	The specified file ID was not found.
`6A86`	Incorrect values entered for P1, P2, or both.
`6D00`	Unknown instruction value entered.
`6E00`	Incorrect class value entered. Enter `00`h for the CLA byte.
`6F00`	Technical problem without a specified diagnostic.

See Also:

- `Get Response` command on page 138
- `Read Record EMV` command on page 160

SHA-1 Intermediate

You can use a Secure Hash Algorithm 1 (SHA-1) to hash a message of any length into a 20-byte digest (for example, to prepare it for encryption with an RSA key. Use the `SHA-1 Intermediate` command only if the message is longer than 64 bytes. Call the command once to process each 64-byte block of data that precedes the final data block.

Call the `SHA-1 Last` command for the final data block (of 64 bytes or less) in a series. For a message that is no more than 64 bytes long, call the `SHA-1 Last` command alone.

S/R: Send *(Case 3)*

AC – Not applicable

Command Format

Command	CLA	INS	P1	P2	Lc
`SHA-1 Intermediate`	`14`	`40`	`00`	`00`	`40`
`SHA-1 Intermediate (ISO-2)`	`10`	`40`	`00`	`00`	`40`

Parameters

Name	Length	Value / Meaning
P1	1 B	`00h`
P2	1 B	`00h`
Lc	1 B	`40h`: Length of the input data. (Must be 64 bytes.)
Input Data	64 B	The first or intermediate hash block, entered LSB-first.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
6740	Incorrect Lc value entered: Either the value specified is not 40h, or does not match the length of the data block.
6CYYh (YY=P3)	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
6B00	Incorrect values entered for P1, P2, or both. (Enter 00h for P1 and P2.)
6D00	Unknown command instruction value entered.
6E00	Incorrect command class value entered.
6F00	Technical problem without a specified diagnostic.
9000	The command succeeded: The card created a SHA-1 digest of the data.

See Also: `SHA-1 Last` command on page 186

SHA-1 Last

Use the `SHA-1 Last` command to calculate the final (or only) data block of a SHA-1 operation. The `SHA-1 Last` command processes a data block that is no more than 64 bytes long. If the message is longer than 64 bytes, hash it in parts by using the `SHA-1 Intermediate` command until you reach the final data block.

S/R: Send/Receive *(Case 4)*

AC – Not applicable

Call the `Get Response` command immediately after `SHA-1 Last` to retrieve the 20-byte digest produced by the final hash operation.

Command Format

Command	CLA	INS	P1	P2	Lc
`SHA-1 Last`	`00`	`40`	`00`	`00`	*lgth*
`SHA-1 Last (ISO-3)`	`04`	`40`	`00`	`00`	*lgth*

Parameters

Name	Length	Value / Meaning
P1	1 B	`00`h
P2	1 B	`00`h
Lc	1 B	Length of the input data block: `01-40`h, octet.
Input Data	*lgth* B	The final (or only) hash block: `01-40`h or 01-64 bytes, octet. You must enter the data in LSB-first format.
SW1, SW2	2 B	Status word bytes the card returns.

Response Data Available

If you follow a `SHA-1 Last` command immediately with a `Get Response` command, the card returns the following data in LSB-first format:

Bytes	Description of Data
1–14	The 14-byte digest produced by the final SHA-1 hash operation, returned in LSB-first format.

Status Words Returned

Hex Value	Meaning
`6114`	The command succeeded. The 20-byte digest is available for return by a `Get Response` command.
`6700`	Incorrect Lc value entered: Lc = `00`h. The length must be `01–40`h octet.
`6740`	Incorrect Lc value entered: Lc > `40`h. The length must be `01–40`h octet.
`6CYYh` `(YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6B00`	Incorrect values entered for P1, P2, or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.

See Also:

- `SHA-1 Intermediate` command on page 184
- `Get Response` command on page 138

Unblock CHV

Use the `Unblock CHV` command to unblock and reset the value of a blocked PIN in the relevant EF_{CHV1} or EF_{CHV2}. If the user enters a PIN value incorrectly until the remaining attempt counter reaches a value of `00`, the CHV key becomes blocked. Once this happens, the card allows no further PIN entry attempts and does not allow access to any of the files protected by the blocked CHV key.

S/R: Send *(Case 3)*

AC – Not applicable

If the `Unblock CHV` command succeeds, it also enables access to the files protected by the CHV file. The CHV attempt counter is reset to the value specified in the input parameter string for the EF_{CHV1} (file 0000), and the unblock CHV attempt counter is reset to 10 (`0A`h). The number of allowed unblock CHV mechanisms is decremented. *(For more information about CHV files, see page 21.)*

If the `Unblock CHV` command fails, its attempt counter is decremented. If the counter reaches zero, the command returns the status word `6983`, and you can no longer execute the command.

If you exhaust all the available unblock attempts, you may be able to use the AAK to regain access to the CHV file and update the data in byte 23 to reset the number of remaining unblock attempts. For this procedure to work, the CHV file you want to modify must be protected by the AAK, and the AAK must not be blocked. You can reset the value of byte 23 by repersonalizing the card.

NOTE *When you unblock a CHV file, byte 22 of the file is automatically reset to 10 (`0A`h). Byte 22 specifies the maximum number of unblock attempts. You cannot change this default reset value.*

Command Format Select the EF_{CHV1} or EF_{CHV2} that contains the PIN you want to unblock and call the Unblock CHV command in the following format.

Command	CLA	INS	P1	P2	Lc
`Unblock CHV`	`F0`	`2C`	`00`	*CHV nbr*	`10`

Parameters

Name	Length	Value / Meaning
P1	1 B	`00h`
P2	1 B	Type of CHV file: • `01h` = EF_{CHV1} (file ID 0000), or • `02h` = EF_{CHV2} (file ID 0100).
Lc	1 B	`10h`: The length of the input data.
Input Data	16 B	Data: Bytes 1–8: The unblock CHV key value, and Bytes 9–16: The new CHV key value.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6300`	The CHV unblocking key entered is incorrect. At least one more attempt is possible.
`6581`	Memory-related problem: The EEPROM may have failed.
`6710`	Incorrect Lc value entered. Enter `10h`.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	The CHV key specified in P2 does not exist.
`6983`	The unblocking key is blocked; no further attempts are possible.
`6B00`	Incorrect values entered for P1, P2, or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The blocked PIN is now unblocked, and access is granted to the files it protects.

See Also:

- `Verify CHV` command on page 198
- `Change CHV` command on page 93

Update Binary

Use the `Update Binary` command to update data in the currently selected transparent EF. To update values in record files (either linear or cyclic EFs), use the `Update Record` command.

S/R: Send *(Case 3)*

AC – To update binary data in a transparent EF, you must first satisfy the access condition (AC) specified for the `Update Binary` command in that EF's input parameter string. (To find out which key number is required to satisfy an AUT or PRO AC, call the `Get AC Keys` command, described on page 133.)

The file must already contain space for the new data you plan to write. (When you create the file you must allocate sufficient space for all the data the file will eventually contain.) Once the file is created, you cannot change its size.

Use Lc to specify an amount of data to write. Use P1 and P2 to specify the offset for the data byte to be overwritten. The frame of reference for the offset is big-endian and zero-based. In other words, the most significant byte (MSB) in the sequence is stored first at the lowest storage address (`0000`). For example, to update bytes 1–8 in a file, enter `07`h for P1, `00`h for P2, and `08`h for Lc.

NOTE *You cannot update data in an invalidated file.*

If you overwrite the AAK value (key 1 in the external key file), make sure you keep a record of the new key value. If you forget the AAK value, you will be unable to satisfy the access condition for the root directory of the card system. You will lock yourself out of the card.

Command Format Select the transparent EF you want to update and call the `Update Binary` command in the following format.

Command	CLA	INS	P1	P2	Lc
`Update Binary`	`C0`	`D6`	*offset MSB*	*offset LSB*	*lgth + X*

Parameters

Name	Length	Value / Meaning
P1	1 B	Most significant byte of offset to begin writing data
P2	1 B	Least significant byte of offset to begin writing data
Lc	1 B	Length of the data: A maximum of 255 bytes (`FF`h), unless executed in PRO mode. In PRO mode, the data is limited to a maximum of 251 bytes (`FB`h), including the PRO cryptogram.
Input Data	*lgth* + *X*	New data to write into the file, plus the cryptogram (*X*) if the PRO AC applies. If no PRO AC applies, *X* = 0.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6283`	The currently selected EF is invalidated.
`6300`	PRO authentication failed because the cryptogram is wrong.
`6581`	Memory-related problem: The EEPROM may have failed.
`67`*xx*	The specified Lc value does not match the data included or is too long. The maximum amount of input data is 255 bytes (`FF`h), or 251 bytes (`FB`h) if you execute the command in PRO mode.
`6CYYh` `(YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	The AC cannot be satisfied because either the cryptographic key, CHV key, or key file is missing.
`6982`	The required AC was not satisfied because the key is invalid or no key was presented.
`6983`	The key required for PRO authentication is blocked.
`6985`	The PRO AC was not satisfied because the host did not send a `Get Challenge` command to the card.
`6986`	A DF is currently selected. Select the transparent EF you want to update and call the command again.
`6A80`	A linear or cyclic EF is currently selected. Select the transparent EF you want to update and call the command again.
`6B00`	The offset specified in P1 and P2 is outside the boundaries of the EF.

Hex Value	Meaning
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The card updated the specified byte values.

Example: Updating the Application Authorization Key

A new card has an 8-byte application authorization key (AAK) stored in the external key file, located in the master DF. The card allows you three attempts to verify the default AAK. The data in the external key file's first key slot (key 0) is used for an 8-byte factory key, which SchlumbergerSema does not distribute. The AAK (key 1) is stored in the second key slot.

The following steps call the `Update Binary` command and change the value of the AAK and its key attempt counter. Note that the key file that contains the AAK is the relevant file for its own AUT access condition. (You use the AAK to gain access to the file that contains the AAK.)

1. Insert the card in the reader and reset it.

   The card responds with the ATR string and selects the master file (3F00).

2. Authenticate the user's access rights to the external key file by sending a `Verify Key` command with the key number, key length, and AAK string to the card.

3. Select the external key file (0011) by sending a `Select` command.

4. Call an `Update Binary` command and send the12 bytes of data shown in the following table to overwrite bytes 14–25 of the external key file. Provide a new 8-byte key (represented in the table by the `XX`s) that uses the DES algorithm (`00`), allows a maximum of 10 (`0A`h) key attempts, and sets the remaining key attempt counter to 10 (`0A`h).

CLA	INS	P1	P2	Lc	Data
`CO`	`D6`	`00`	`0D`	`0C`	`08 00 XX XX XX XX XX XX XX XX 0A 0A`

5. Send a `Verify Key` command to test the new AAK.

See Also:

- `Update Record` command on page 195
- `Update Binary Enciphered` command on page 193
- "Commands That Are Subject to Access Conditions," on page 64

Update Binary Enciphered

Use the `Update Binary Enciphered` command to use DES-encrypted data to update the values of a string of bytes in the selected transparent EF.

S/R: Send *(Case 3)*

AC – To update data with the `Update Binary Enciphered` command, you must first satisfy the access condition (AC) specified for the command in the input parameter string of the target EF. (To find out which key number is required to satisfy an AUT or PRO AC, call the `Get AC Keys` command, described on page 133.)

Decryption Key — To decrypt the data, the card uses the key number specified in the input parameter string of the currently selected file. The key number refers to a key in the relevant external key file. (The `Update Binary Enciphered` key is specified in the LSN of byte 15 in the `Create File` structure data. This key number applies to a DES or 3DES key in the relevant external key file.) To retrieve this key number, call a `Dir Next` command for the target EF and examine the LSN of byte 13.

Once the data is decrypted, the card writes it into the file. The target EF must contain sufficient available space for the new data. Use Lc to specify the amount of data to write, and use P1 and P2 to define the offset for the beginning of the new data string. The data length must be evenly divisible into blocks of 8 bytes, since the card performs no padding for this command.

The frame of reference for the offset is big-endian and zero-based. In other words, the most significant byte (MSB) in the sequence is stored first, at the lowest storage address (`0000`). For example, to update bytes 12–19 in a file, enter `0B`h for P1, `12`h for P2, and `08`h for Lc.

Command Format Select the transparent EF you want to update and call the `Update Binary Enciphered` command in the following format.

Command	CLA	INS	P1	P2	Lc
`Update Binary Enciphered`	`04`	`D6`	*offset MSB*	*offset LSB*	*lgth*

Parameters

Name	Length	Value / Meaning
P1	1 B	Most significant byte of offset to begin writing data.
P2	1 B	Least significant byte of offset to begin writing data.
Lc	1 B	Length of the input data (and number of bytes to be updated): A value between `08–E8`h in octet format (8–232 bytes decimal).
Input Data	*lgth* B	The enciphered data to write to the EF.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6283`	The currently selected EF is invalidated.
`6581`	Memory-related problem: The EEPROM may have failed.
`6700`	The input data is too long.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	The AC cannot be satisfied because either the card contains no relevant external key file or the specified AC key is missing.
`6982`	The required AC was not satisfied because the key is invalid or no key was presented.
`6985`	Algo Id not authorized for the key used.
`6986`	A DF is currently selected. Select a transparent EF.
`6A80`	A linear or cyclic EF is currently selected. Select a transparent EF.
`6B00`	The offset specified in P1 and P2 is outside the boundaries of the EF.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The card has received, deciphered, and stored the new data in the specified location.

See Also:

- `Update Binary` command on page 190
- “Commands That Are Subject to Access Conditions,” on page 64

Update Record

Use the `Update Record` command to write new data into a specified record in the currently selected linear EF or the oldest record in a cyclic EF.

NOTE *To change number values in cyclic file records, you can use the* `Decrease` *and* `Increase` *commands as well as the* `Update Record` *command. To update data in a transparent EF, use the* `Update Binary` *command.*

S/R: Send *(Case 3)*

AC – To issue a successful `Update Record` command, you must first satisfy the access condition (AC) specified for this command in the input parameter string of the currently selected linear or cyclic EF. (To find out which key number is required to satisfy an AUT AC, call the `Get AC Keys` command, described on page 133.)

Specifying a Record to Update

Use P2 and P1 to specify the record selection mode and record to update.

Cyclic EF — You must use the *previous* option in current mode (P2 = `02`h). The oldest record is updated, and the record pointer is set to this record.

Linear EF — Specify a record in either of these ways:

- Use the absolute mode (P2 = `04`h) and choose a record by number (in P1), or
- Use the current mode (P2 = `00`–`03`h) and choose the *first*, *last*, *next*, or *previous* record in the file.

  To update the first or last record in the file, set P2 to `00`h (current mode, first record) or `01`h (current mode, last record). You can then call the command again to update the *next* or *previous* record. If the first `Update Record` call is for the *next* record, the card updates the first record in the file. If the first `Update Record` call is for the *previous* record, the card updates the last record in the file.

Command Format Select the linear or cyclic EF that contains the record you want to update and call the `Update Record` command in the following format.

Command	CLA	INS	P1	P2	Lc
`Update Record`	`C0`	`DC`	*rec nbr*	*mode*	*lgth* + *X*

Parameters

Name	**Length**	**Value / Meaning**
P1	1 B	Record number: • `00`h = No record number specified (Current mode) • `01`–*xx*h = Number of the linear record to update (Absolute mode, P2 = `04`h)
P2	1 B	Record selection mode: For cyclic records, specify: • `02`h = Next record, current mode (P1 = `00`h). For linear records, specify one of these options: • `00`h = First record, current mode (P1 = `00`h). Updates first record in the file. • `01`h = Last record, current mode (P1 = `00`h) Updates last record in the file. • `02`h = Next record, current mode (P1 = `00`h) Updates the record after the currently selected one. If no record is selected, updates the first record in the file. • `03`h = Previous record, current mode (P1 = `00`h) Updates the record before the currently selected one. If no record is selected, updates the last record in the file. • `04`h = Absolute mode (if P1 is a non-null value), or Current record (if P1 = `00`h). (The current record is the record currently selected by the record pointer.) The record pointer does not change location.
Lc	1 B	Length of the record to be updated + *X*. (*X* =Cryptogram length if AC = PRO, or 0 if no PRO AC.) The card updates the specified number of bytes, starting with the first data byte in the record. If you send fewer bytes of data than the record contains, the record's last data bytes are not updated. If the PRO AC applies, Lc must be less than `FC`h (252 bytes), or the command fails, and the card returns status word `6700`.
Input Data	*lgth* B	Data to write to the record + *X* (the cryptogram, if the AC = PRO). If no PRO AC applies, *X* = 0.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6283`	The currently selected EF is invalidated.
`6300`	PRO authentication failed because the cryptogram is wrong.
`6581`	Memory-related problem: The EEPROM may have failed.
`67`*xx*	The value entered for Lc does not match the specified record's length. (The value that appears in place of *xx* is the correct Lc entry.)
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	The AC cannot be satisfied because either the cryptographic key, CHV key, or key file is missing.
`6982`	The required AC was not satisfied because the key is invalid or no key was presented.
`6983`	The key required for PRO authentication is blocked.
`6985`	The PRO AC was not satisfied because the host did not send a `Get Challenge` command to the card.
`6986`	A DF is currently selected. Select a linear or cyclic EF.
`6A80`	A transparent EF is currently selected. Select a linear or cyclic EF.
`6A83`	The specified record number is not found; or you have reached the end of the file, and no *next* record remains to be updated.
`6B00`	Incorrect values entered for P1, P2, or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The card overwrote the specified record values with the new data.

See Also:

- `Read Record` command on page 157
- `Create Record` command on page 105
- `Decrease` command on page 107
- `Increase` command on page 143
- "Commands That Are Subject to Access Conditions," on page 64

Verify CHV

Use the `Verify CHV` command to verify the access rights of the current card user. You ask the user to supply a CHV value stored in the relevant EF_{CHV1} (0000) or EF_{CHV2} (0100) on the card. After a successful verification, the card grants the corresponding CHV1 or CHV2 access condition (AC).

S/R: Send *(Case 3)*

AC – Not applicable

You can issue a successful `Verify CHV` command only if a relevant CHV file exists, contains a valid PIN, and the PIN is not blocked. The CHV file must also be active (not invalidated).

If the verification succeeds, the card grants the corresponding CHV AC to the user. The AC persists until one of the events described on page 67 occurs.

A counter keeps track of how many times the user tries to verify the PIN, decrementing the counter for each failure. If the counter reaches a null value, no further attempts are allowed, and the card blocks access to the files protected by the CHV key. If this happens, a card administrator can perform an `Unblock CHV` operation to reset the counter, assign a new PIN value, and allow the user access to the protected files. After each successful attempt, the counter retrieves the maximum default value.

Command Format

Navigate to the EF_{CHV1} or EF_{CHV2} that contains the PIN you want to verify and call the `Verify CHV` command in the following format.

Command	CLA	INS	P1	P2	Lc
`Verify CHV`	`C0`	`20`	`00`	*CHV nbr*	`08`

Parameters

Name	Length	Value / Meaning
P1	1 B	`00h`
P2	1 B	Specifies the type of CHV: `01h` = CHV1 `02h` = CHV2
Lc	1 B	`08h`: Length of the input data (the PIN value).

Name	Length	Value / Meaning
Input Data	8 B	PIN value.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	Meaning
`6300`	The PIN value entered is incorrect.
`6708`	The PIN length entered for the Lc parameter is unsupported or does not match the length of the input data. The correct Lc value is `08`h.
`6CYYh (YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	Either the CHV key specified in P2 does not exist, or no relevant CHV file exists.
`6983`	The CHV key is blocked: No further verification attempts are allowed.
`6B00`	Incorrect values entered for P1, P2, or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The user’s access rights are established.

See Also: `Verify Key` command on page 200

Verify Key

Use the `Verify Key` command to verify the access rights of the host application user. The user must present a key stored on the card in the relevant external key file (0011). After a successful verification, the card grants the corresponding AUT access condition. No encryption is involved.

The most common use of the `Verify Key` command is to unlock a card. For example, you use this command to present the SchlumbergerSema-supplied AAK stored as key 1 in the external key file (0011), located in the root directory (3F00).

S/R: Send *(Case 3)*

AC – Not applicable

You can issue a successful `Verify Key` command only if a relevant external key file exists, the file contains the necessary key, and the key is not blocked. The external key file must also be active (not invalidated).

If the verification succeeds, the card grants the corresponding AUT AC to the user. The AC persists until one of the events described on page 67 occurs.

A new card allows you three attempts to verify the AAK. A counter keeps track of the number of unsuccessful attempts, decrementing the counter with each failure. If the counter reaches a null value, the key is blocked and you have no further access to the card. See the `Update Binary` command (described on page 190) for an example of how to change the AAK value and number of verification attempts allowed. If you have further questions about card blocking, contact SchlumbergerSema technical support.

Command Format

Navigate to the DF that contains the external key you want to verify and call the `Verify Key` command in the following format.

Command	CLA	INS	P1	P2	Lc
`Verify Key`	`F0`	`2A`	`00`	*key nbr*	*lgth*

Parameters

Name	**Length**	**Value / Meaning**
P1	1 B	`00h`
P2	1 B	Key number to use for verifying a particular user's access rights, located in the relevant external key file. Enter a value from `00`h (for key 0) to `0F`h (for key 15).
Lc	1 B	Length of the input data—the key value. (`10`h for the AAK or a DES key, `0F`h for a double-length 3DES key.)
Input Data	*lgth* B	Key value.
SW1, SW2	2 B	Status word bytes the card returns.

Status Words Returned

Hex Value	**Meaning**
`6300`	The specified key value is incorrect.
`67`*xx*	The value entered for Lc does not match the length of the input data. (The value that appears in place of *xx* is the correct Lc entry.)
`6CYYh` `(YY=P3)`	**32K+e-gate USB mode only**: wLength of SendData request does not correspond to P3.
`6981`	Either the key slot specified in P2 does not exist, or no relevant external key file exists.
`6983`	The key is blocked: No further verification attempts are allowed.
`6B00`	Unsupported values entered for P1, P2, or both.
`6D00`	Unknown command instruction value entered.
`6E00`	Incorrect command class value entered.
`6F00`	Technical problem without a specified diagnostic.
`9000`	The command succeeded: The user's AC is established.

See Also: `Verify CHV` command on page 198

5

Writing a Card Application

This section contains step-by-step instructions for setting up a simple Cryptoflex card application. The sample application demonstrates some of the card's basic capabilities—capabilities you can use regardless of the programming environment or standards you use to develop your card application.

In the exercises that follow, you make the card act as a personal identification token by adding a CHV key and user data to the card. You set up the card with a key to protect a Public Key Infrastructure (PKI) directory, then add a directory to hold the PKI data. You also generate an RSA key pair, store it on the card, and generate a digital signature.

NOTE *The simplified example described throughout this section is for illustration purposes only. The example does not create a file structure consistent with COVE personalization profiles for PKI applications, SchlumbergerSema CryptoAPI applications, or SchlumbergerSema PKCS #11 applications. If you develop card applications with the Cyberflex Access Software Development Kit, use the COVE application to personalize the card with an appropriate file structure. (For more information about the COVE application, see the Cyberflex Access Software Development Kit User's Guide.)*

This section includes examples of the following commands:

- `Change CHV` (page 225)
- `Create File`, to create these types of files:
  - CHV key (page 216)
  - Dedicated file (page 218)
  - RSA private key (page 219)
  - RSA public key (page 221)
  - Fixed-length linear elementary file (page 217)
- `Generate RSA Keys` (page 226)
- `Get Response` (page 229)
- `Logout AC` (page 230)
- `RSA Signature (Internal Auth)` (page 229)
- `Select` (page 215)
- `SHA-1 Intermediate` and `SHA-1 Last` (page 227)
- `Update Binary` (page 224)
- `Update Record` (page 223)
- `Verify Key` (page 217, page 218)

Tasks Covered in the Exercises

The exercises are divided into two general phases, which include a total of twelve steps:

Custom Card Pre-personalization

In this phase, you unlock the card and add security and structural files at the root level. The elements you add support the actions you want the card to be able to perform. The steps are:

1. Unlock the card with the application authorization key (AAK) (page 213).
2. Add a key to the default external key file, which you can use for protecting a PKI application (page 213).
3. Add a CHV key file to hold the cardholder's PIN (page 215).
4. Add a file to hold the cardholder's personal identification data (page 216).
5. Add a directory to hold PKI application data (page 217).
6. Add a file to hold the RSA private key (page 218).
7. Add a file to hold the RSA public key (page 220).

Personalization for the Card's End User

In this phase, you add card-specific data to the files you have created:

8. Store the cardholder's personal identification data (page 223).
9. Store the cardholder's initial PIN in the EF_{CHV1} you created, and learn how to update the PIN value (page 224).
10. Generate and store the RSA key pair on the card (page 226).
11. Compute a SHA-1 hash of data you want to send or store (page 226).
12. Generate and retrieve a digital signature of the hash (page 227); then log out (clear) the user's access rights (page 229).

Testing and Development Phases

Smart cards undergo several processes before they are ready to run applications. A Cryptoflex card designed for testing and development goes through these phases:

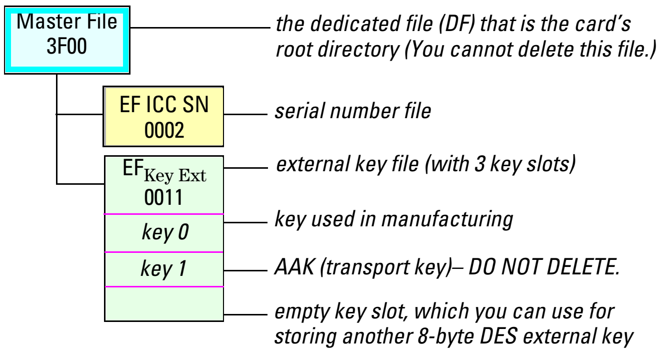
- **Default pre-personalization** — Standard Cryptoflex cards are given a rudimentary file system at the factory before they are shipped to you.
- **Experimentation** — Use the COVE utility and the Smart Card Toolkit to set up the test card and learn the card's capabilities and conventions. When you finish experimenting, delete the unneeded files from the card.
- **Custom personalization** —Build a card file system and add the keys you need to support the operations of the application you are developing.
- **Final personalization** —Add end user data for testing.
- **Testing** —Run commands on the card to see if the application behaves as you expect.

When your development work is finished, you know which structural files, cryptographic key files, and CHV files you need to support your card application(s). If your project involves issuing a large number of cards, your company may ask SchlumbergerSema to pre-personalize cards with your custom file structure.

The final personalization can be carried out at your company's site or off site by an appointed service provider. The personalization agent adds a variety of information to the card (such as the users' personal identification data; initial PIN values for the cardholder and card administrator; and values for external keys, internal keys, and key pairs).

The Card's Default File System

When you receive a new Cryptoflex test card, it has the rudimentary file system SchlumbergerSema added during default pre-personalization. The card has a master file (the root directory), which contains the files shown in the following illustration.


Master File
3F00
the dedicated file (DF) that is the card's root directory (You cannot delete this file.)
EF ICC SN
0002
serial number file
EF Key Ext
0011
external key file (with 3 key slots)
key 0
key used in manufacturing
key 1
AAK (transport key)– DO NOT DELETE.
empty key slot, which you can use for storing another 8-byte DES external key

Default File System on a Cryptoflex Card

Master file – The master file has the reserved file ID 3F00. Its access conditions (ACs) require that you verify your access rights by presenting the AAK before you can perform most commands, such as creating or deleting files, or calling the `DIR Next` command.

Serial number file – The serial number files has the reserved file ID 0002. The serial number file uniquely identifies the card's microprocessor. SchlumbergerSema typically uses the serial number to test the card and control processing flows. You can also use the serial number for specific purposes in your applications, if you like. This file can hold an additional identifier for the card series.

External key file – The external key file has the reserved file ID 0011. The external key file has three key slots:

- Key 0 – A key used only during manufacturing. (Corresponds to AUT0 in the Smart Card Toolkit.)
- Key 1 – The application authorization key (AAK), also called the transport key, which you use to unlock the card. Be very careful about any changes you make to this key. Never delete the AAK, or you will be permanently locked out of the card.

- At the end of the external key file are 8 bytes of blank EEPROM memory. You can add another single-length DES key to this key slot. For example, you could add a key for protecting access to one or more application directories you plan to put on the card.

Custom Pre-personalization

Custom pre-personalization is the process of building a file system on the card and adding the key files needed to support your application's operations. In the following custom pre-personalization, you add a CHV key (PIN) file and add a new key to the external key file. You build a simple file structure to contain these elements:

- Cardholder identification data (ID 2001)
- A cryptographic key to protect a PKI application you plan to add to the card (ID 0000)
- A directory to hold the PKI application data (ID 4F01)
- Files to store an RSA key pair (private key file ID 0012 and public key file ID 1012)

File System Details

The following table summarizes information about the example file system.

DF ID	EF ID	Description	Length	File Type	AC Settings	AC Keys
3F00		Master file, containing the subsidiary DF and 4 EFs below:	—	DF	`4F4444`	`1X1111`
	0002	Serial number	8 B	transparent EF	`04FFFF`	`X1XXXX`
	0011	External keys	54 B	transparent EF	`F4FF44`	`X1XX11`
	0000	CHV1 keys	23 B	transparent EF	`F4FF44`	`X1XX11`
	2001	Data file for the cardholder's identification data	24 B	fixed-length linear EF	`04FFFF`	`X1XXXX`
4F01		PKI application directory, containing 2 EFs:	840 B	DF	`4F44FF`	`2X22XX`
	0012	Private keys	400 B	transparent EF	`F1FF44`	`XXXX22`
	1012	Public keys	400 B	transparent EF	`01FF44`	`XXXX22`

NOTE *__AC Settings__ and __AC Keys__ columns – For information about the commands protected by the access condition (AC) settings, see "Setting Access Rights on Card Operations," on page 62, and the table on page 221. The AC key settings specify a key number for each nibble protected by an AC.*

Calculating the Size of an Application Directory

The table that follows shows the calculations to determine the size of the example application directory (DF). It is important to set an appropriate file size when you create directories and elementary files (EFs). You cannot change a file's size after you create it.

When you calculate a DF's size, be sure to include space for the input parameter strings for DFs you plan to include (which have 24-byte input parameter strings) and EFs (which have 16-byte input parameter strings).

Minimum Size for the PKI Directory

The following table shows the two EFs the new directory will contain. The total size shown is the minimum number of EEPROM bytes the directory must span in order to accommodate the planned files. Note that the directory's input parameter string size is not included—this data is not part of the directory itself.

File Name	File ID	Content Size	Input Para Size	Total Size
$EF_{RSA\ PRI}$ (private key)	0012*	400 B	16 B	416 B
$EF_{RSA\ PUB}$ (public key)	1012*	400 B	16 B	416 B
				832 B

NOTE *File IDs marked with an asterisk (*) are reserved IDs.*

Minimum Available EEPROM Memory Needed

The table below shows all the files that will exist on the card after the exercise is complete. Total these file sizes to find the amount of EEPROM memory needed to accommodate the planned application. When you create your own applications, you may have many more calculations to make.

File Name	File ID	Content Size	Input Parameters	Total Size
$EF_{ICC\ SN}$ (serial number)	0002*	8 B	16 B	24 B
$EF_{Key\ Ext}$ (external key)	0011*	37 B	16 B	53 B
EF_{CHV1}	0000*	23 B	16 B	39 B
User ID	2001	24 B	16 B	40 B
PKI directory	4F01	832 B	24 B	856 B
				1020 B

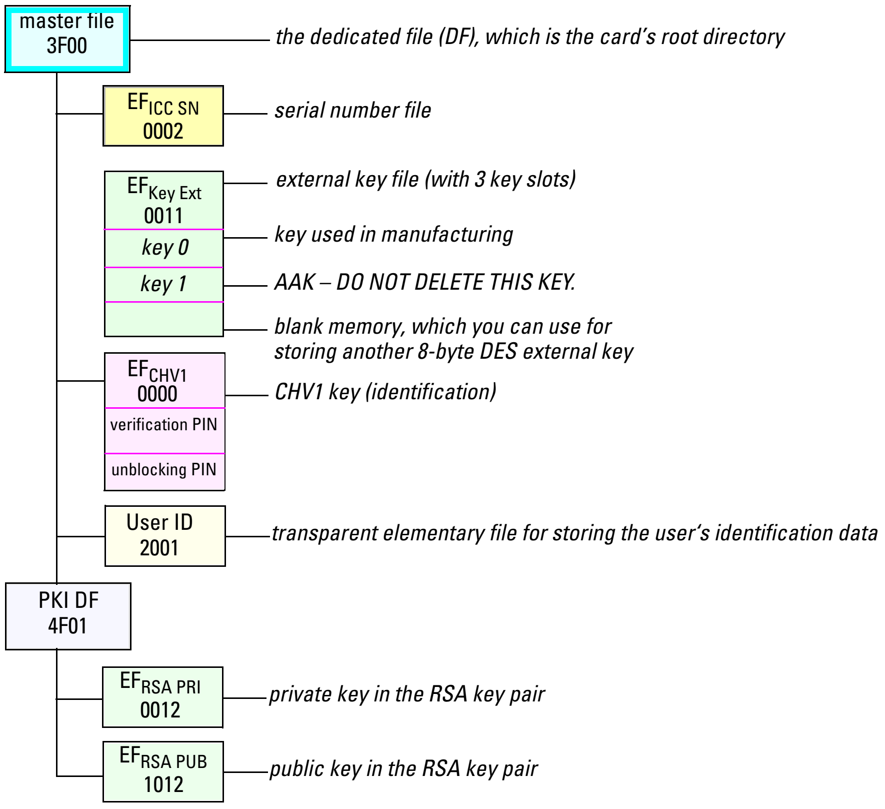
NOTE *File IDs marked with an asterisk (*) are reserved IDs.*

The amount of available EEPROM memory on a particular card depends on the size of the card's softmask. For example, the total amount of EEPROM available on the Cryptoflex 16K card is 14,400 bytes by default. A pre-personalized Cryptoflex 16K card typically has more than 6800 bytes of EEPROM available.

As you can see from the table, enough EEPROM memory remains on the card for another application to be added later.

The Custom Card File System

The following illustration shows the card's file system after the example application files are added.


master file
3F00
the dedicated file (DF), which is the card's root directory
EF ICC SN
0002
serial number file
EF Key Ext
0011
external key file (with 3 key slots)
key 0
key used in manufacturing
key 1
AAK – DO NOT DELETE THIS KEY.
blank memory, which you can use for storing another 8-byte DES external key
EF CHV1
0000
CHV1 key (identification)
verification PIN
unblocking PIN
User ID
2001
transparent elementary file for storing the user's identification data
PKI DF
4F01
EF RSA PRI
0012
private key in the RSA key pair
EF RSA PUB
1012
public key in the RSA key pair

Customized Card File System

Pre-personalizing the Card with a Custom File Set

Once you finish planning the card's custom file set, you pre-personalize the card by sending it ISO-format APDU commands. The APDU commands are presented in table format with these types of information displayed:

CLA	Command class denominator
INS	Command instruction denominator (The CLS and INS bytes together identify the type of command.)
P1	Parameter 1 value (The type of information in the P1 and P2 parameters is command-specific.)
P2	Parameter 2 value
Lc	Parameter 3 value for a send mode command: the input data length
Le	Parameter 3 value for a receive mode command: the output data length
Data	Input data that you include with the command, or output data the card returns as a result of the command

NOTES

- *Enter all APDU values in hexadecimal format.*
- *For information about the structure of input and output APDU commands, see "The Communication Interface" starting on page 231.*
- *If the command completes successfully, the card returns a status word (SW) of `9000`, or returns a status that indicates how many bytes of data are available for return by a `Get Response` command. Status words are described for each command in the section "Cryptoflex Card Commands" starting on page 87.*

Unlocking a New Card

Begin by using the AAK, also called the transport key, to unlock the card.

Step 1: Unlock the Card

Use the `Verify Key` command to unlock a new card. This demonstrates that you have the SchlumbergerSema-supplied AAK and establishes access rights to create files under the master file. The AAK is stored as key 1 in the external key file (0011), located in the master file (3F00). (For more information about the `Verify Key` command, see page 200.)

Use the following format for the `Verify Key` command:

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**
`F0`	`2A`	`00`	`01`	`08`	*AAK value* (8 bytes) The AAK value for your Cryptoflex card is shipped with the card.

NOTE *For information about using the Smart Card Toolkit application (available in the Cyberflex Access SDK) to verify the AAK, see page 19.*

Setting Up Additional Security Support

In steps 2 and 3, you create the keys and key files you need at the card's root level. These are the key elements the application requires that were not added to the card during default pre-personalization. You add an additional external key and an EF_{CHV1}. You will use these keys for ACs you set as you create directories in your file structure.

Step 2: Add Key 2 to the External Key File

In step 2, you add an additional external key to the card. The new key can be used to protect a PKI application.

Select the External Key File, 0011

Before you can add a key to the external key file, the external key file must be the currently selected file. Call the `Select` command with the following APDU string:

CLA	INS	P1	P2	Lc	Data
C0	A4	00	00	02	0011

NOTE *For more information about the `Select` command, see page 177.*

Add Key 2 to External Key File, 0011

Add key 2 to the external authentication key file (external key file, 0011) to support the PKI security features.

Call the `Update Binary` command with the following APDU string. (For more information about the `Update Binary` command, see page 190.)

CLA	INS	P1	P2	Lc	Data
C0	D6	00	19	0C	08 00 1234567812345678 0A 0A

When you call the `Update Binary` command, make sure you accurately enter the offset for the data you want to update. In the example, the key file contains two 8-byte key slots before key slot 2. The first byte is RFU. To instruct the card to write the new key information to the appropriate memory locations (bytes 26 through 38), set the offset to 25. (P1 and P2 define the offset, as described on page 190.)

Data Field Values

`08`	Specifies the key length, 8 bytes. (The card uses this value as the increment to advance through the key file when it searches for a key.)
`00`	Specifies that the new key is a DES key.
`1234567812345678`	The value you assign to the key.

`0A`	The maximum number of verification attempts, set to 10. The user is allowed 10 attempts to present the key correctly before the key is blocked.
`0A`	The current number remaining of verification attempts, set to 10. These are the number of attempts that have not been used. This number is revised with each failed attempt and is reset to 10 if a presentation attempt succeeds.

As you can see, anyone who knows the value of key 1 will be able to update all the key values in the master file. In your own application, make sure you implement adequate security to keep the value of key 1 confidential.

Step 3: Create the CHV File

In this step, you add an EF_{CHV1} to the master file.

The default behavior for some commands on Cryptoflex 16K cards requires verification of the CHV1 key. For this reason, you need to create a EF_{CHV1} for the card application, even if you do not plan to set any CHV ACs on files you create. (To find out which commands require CHV1 verification, see the table on page 91.) In your own application, you may want to create an EF_{CHV1} and EF_{CHV2}, so you can assign access rights for a card administrator and a cardholder.

Select the Master File

Before you can create a file in the master file (MF), the MF must be the currently selected file. The card selects the MF automatically whenever the card's microprocessor is powered up—either when the card is inserted into a reader or is reset. If you have performed an action that changed this selection, call the `Select` command with the following APDU string:

CLA	INS	P1	P2	Lc	Data
`C0`	`A4`	`00`	`00`	`02`	`3F00`

NOTE *For more information about the `Select` command, see page 177.*

Create the CHV1 File

With the MF already selected, you are ready to create the EF$_{CHV1}$ (reserved file ID 0000). Call the `Create File` command with the following APDU string:

CLA	INS	P1	P2	Lc	Data
`F0`	`E0`	`00`	`01`	`10`	`FFFF 0017 0000 01 00 F4FF44 01 03 X2XX22`

QUESTION or NOTE *should X2XX22 be X1XX11 to match table on p. 216?*

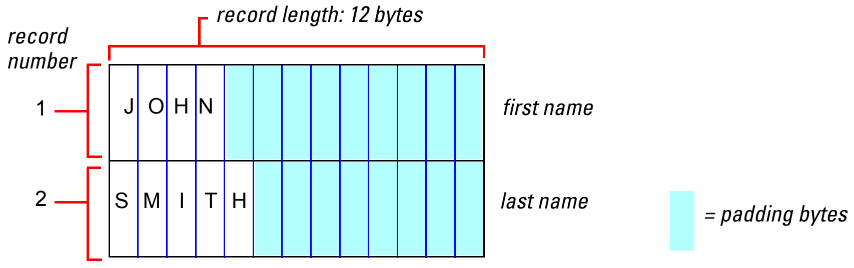
NOTE *In the example, the AUT AC is set on some of the CHV file's commands, so you specify key numbers to use for satisfying AUT ACs. For more information about the ACs set throughout this exercise, see the AC summary table on page 221.*

Adding a Cardholder Identification File

In step 4, you add a file to hold information about the cardholder. This is a simple data file, not one of the essential structural or security files.

Step 4: Add the Cardholder's Name File

Create a file to store the cardholder's identification data. The new file is a fixed-length linear EF with the file ID 2001, located under the master file (3F00). The body of the file will contain the data shown here.



Data Body for the Cardholder's Name File (Fixed-Length Linear EF)

Select the Master File

Once again, before you can create the file, you must select the directory that will contain it. If the master file (3F00) is not currently selected, use the `Select` command to reset the card file pointer to it, as described on page 215.

Create the Cardholder Name File

Create a fixed-length linear elementary file (with file ID 2001). Use the following APDU string to call the `Create File` command:

CLA	INS	P1	P2	Lc	Data
`F0`	`E0`	`00`	`02`	`11`	`FFFF 0018 2001 02 00 04FFFF 01 04 X2XXXX 0C`

QUESTION or NOTE *should X2XXXX be X1XXXX to match table on p. 216?*

NOTE *Make sure the Lc and data byte 13 values are compatible. In this example, you create a fixed-length linear EF, so the Lc value (the length of the file structure data) is* `11`*h. You set data byte 13 (the length of the data from byte 14 to EOF) to* `04`*h. To create a DF, you set Lc to* `10` *h, and data byte 13 to* `03`*h.*

You will add the cardholder's name data later in the exercises (page 223).

Adding Structure for a PKI Application

In steps 5 through 7, you add structural and security files for a PKI application. When you create an application of your own, make sure you implement adequate security to protect the sensitive data contained in the application directory.

Step 5: Add the PKI Directory

Add a directory for the PKI application, a DF with the file ID 4F01, under the master file (3F00).

Before you can create this file, the external key file must exist on the card (located under the MF) and must contain key data in key slot 2. This is the relevant key for the current context, which you must verify before you can create the new directory. (For information about relevant keys, see page 219.)

1. Begin by selecting the master file, if it is not already selected. (See page 215.)
2. Before you can create new files, you must satisfy the `Create File` access condition (AC) for the current context (the MF). The AC requires you to verify key 1 (the AAK) in the EF_{KeyExt} file (0011), located in the root directory (3F00).

Use the following format for the `Verify Key` command:

CLA	INS	P1	P2	Lc	Data
F0	2A	00	01	08	*AAK value* (8 bytes) This key value is shipped with the card.

NOTE *For more information about the `Verify Key` command, see page 200.*

3 Create a new PKI DF, with file ID 4F01, by calling the `Create File` command with the following APDU string:

CLA	INS	P1	P2	Lc	Data
F0	E0	00	01	10	FFFF 0340 4F01 38 00 4F44FF 01 03 2X22XX

Step 6: Add a Private Key File

Continue to build the key file set for the PKI application by adding a private key file (EF$_{RSA\ PRI}$ file 0012) under the PKI directory (4F01). The new file is a transparent EF that will contain the private key. The private key is the secret RSA key the card uses to encrypt and decrypt sensitive messages or data.

Steps for Adding the Private Key File

If the parent directory (file 4F01 under 3F00) is still selected, you do not need to execute another `Select` command.

1 Select the new file's parent directory, if necessary, by calling the `Select` command. The parent directory of the file you are going to create is the master file, 4F01 (page 217).

2 Before you can create new files in the 4F01 DF, you must satisfy the `Create File` access condition (AC) for 4F01. The AC requires you to verify key 2 in the EF$_{KeyExt}$ file (0011), located in the root directory (3F00). This is the key you added in step 2 on page 213.

The external key 2 is the *relevant* key for the current context. To establish the appropriate permission to create a file, perform an external authentication by calling the `Verify Key` command with the following APDU string. (For more information about the `Verify Key` command, see page 200.)

CLA	INS	P1	P2	Lc	Data
F0	2A	00	02	08	1234567812345678

Relevant Key Files

The card's file structure is divided into security domains by the key files you add. CHV, external, and internal key files' domains extend to:

- *The key file's parent directory and its contents, and*
- *Files that are located below the key file's directory*

The key file's domain extends downward in the card file structure until the key file is replaced by another key file of the same type on a lower level.

You verify your access rights for a given domain by presenting the appropriate key from the relevant key file. The relevant key file is the one that protects the currently selected file. When you leave that key file's domain, you lose your access rights, and must prove your access rights again for the new domain you enter. (Other circumstances can also cause you to lose your security clearance, as described on page 220.)

3 Create the private key file (reserved file ID 0012) by calling a `Create File` command with the following APDU string:

CLA	INS	P1	P2	Lc	Data
`F0`	`E0`	`00`	`01`	`10`	`FFFF 0190 0012 01 00 F1FF44 01 03 XXXX22`

You will generate the private key data during personalization (page 226).

Step 7: Add a Public Key File

In this step, you add the public key file for the PKI application under the directory 4F01. The new file is a transparent EF that will contain the public modulus, *N*, for the RSA key.

Prerequisites

- If the parent directory (file 4F01 under 3F00) is still selected, you do not need to execute another `Select` command.
- You verified external key #2 in the exercise on page 218. If your access rights are still in force, you do not have to re-establish the AC to create new files.
- If the private key file has been created, it automatically becomes the current selected file. You must first select DF 4F01, then create the public key file.

Persistence of Access Rights

Once you establish access rights, they remain in force until one of these conditions occurs:

- *The card session ends (for example, power is reset or the card is removed from the reader).*
- *You navigate to a part of the card's file structure that is protected by a different external key file.*
- *The PIN is blocked by a series of unsuccessful authentication attempts.*
- *The parent directory is invalidated.*
- *The user AC you established is logged out, removing your privileges.*

Create the PKI Application's Public Key File

Call the `Create File` command with the following APDU string:

CLA	INS	P1	P2	Lc	Data
`F0`	`E0`	`00`	`01`	`10`	`FFFF 0190 1012 01 00 01FF44 01 03 XXXX22`

You will add the public key data during the personalization process (page 226).

NOTE *You have now completed custom pre-personalization. In your own approach, you could continue adding more directories. For example, you might add a directory to hold a loyalty application that stores loyalty points given to the cardholder with each purchase from your company's Web store.*

Reviewing the Access Conditions You Have Set

The following table summarizes the access conditions (ACs) that are now set for the files in the example card application. Refer to this table to see which operations are permitted for particular files, and which keys must be verified to perform operations that are permitted, but protected. Each of the six AC nibbles can protect one or more commands. (Note that some commands apply only to certain file types.)

Directory ID	Elementary File ID	1: Read Bin, Seek, Dir Next	2: Update Bin, Decrease	3: Increase, Delete File	4: Create Record, Create File	5: Rehabilitate	6: Invalidate
3F00		AUT1	*not applicable*	AUT1	AUT1	*not applicable*	*not applicable*
	0002	ALW	AUT1	NEV	NEV	NEV	NEV
	0011	NEV	AUT1	NEV	NEV	AUT1	AUT1
	0000	NEV	AUT1	NEV	NEV	AUT1	AUT1
	2001	ALW	AUT2	NEV	NEV	NEV	NEV
4F01		AUT2	*not applicable*	AUT2	AUT2	*not applicable*	*not applicable*
	0012	NEV	CHV1	NEV	NEV	AUT2	AUT2
	1012	ALW	CHV1	NEV	NEV	AUT2	AUT2

NOTE *For more information about access conditions, see page 62.*

Personalizing the Card for the End User

At this point, you have pre-personalized the card and are ready to personalize it. This means that you will add the card-specific information needed for your application.

In the next few topics, you will learn how to personalize the card with these types of data:

- General cardholder identification data
- The initial PIN the cardholder presents to start using the card
- Encryption keys to use for digital signatures

You will also learn how to update the cardholder's PIN, compute hash digests, and log out currently verified access rights.

Adding User-Specific Data to the Card

Let's say one of the cards in the sample series goes to John Smith, a recently hired employee. John needs a card to gain access to his workstation, and to encrypt and sign email. Personnel gives John a card that identifies him, gives him network clearance, and sets up secure email privileges for him.

In steps 8 and 9, you personalize the card with John's name and PIN. The card's PIN data can only be updated by someone who knows the value of the current CHV1 PIN. At this point, only the personalization administrator has this information. The administrator performs the following steps to personalize the card with John's information.

Step 8: Add the User's Name to the Card

You have already verified external key #2, so you have access rights to create new files.

1. Call the `Select` command and select the file 2001, located under 3F00. (For more information about the `Select` command, see page 177.)

2. Add the user's first name to the card by calling the `Update Record` command with the following APDU string. (For more information about the `Update Record` command, see page 195.)

CLA	INS	P1	P2	Lc	Data
`C0`	`DC`	`01`	`04`	`0C`	'JOHN'`3030303030303030`

Data – Enter the hexadecimal values for JOHN, and follow with padding bytes to fill the record body.

3. Add the user's last name to the card by calling the `Update Record` command with the following APDU string:

CLA	INS	P1	P2	Lc	Data
`C0`	`DC`	`02`	`04`	`0C`	'SMITH'`30303030303030`

Step 9: Add (and Update) the User's PIN

In the example scenario, the administrator puts a default PIN in the EF_{CHV1}. You use the `Update Binary` command to initialize the EF_{CHV1} data. The application can use the `Change CHV` command to update the PIN when the cardholder first uses the card and is prompted to change the original PIN.

Initialize the CHV File with a Default PIN Value

Write the user's PIN data to the card by calling the `Update Binary` command (described on page 190) with the following APDU string:

CLA	INS	P1	P2	Lc	Data
C0	D6	00	00	17	FF FFFF 3030303030303030 0A 0A 3131313131313131 0A 0A

Data Field Values

FF	Activation byte. The value of bit 0 must equal 1 to specify that the file is active. (1 B)
FFFF	RFU (2 B)
30303030 30303030	The first 4 bytes contain the user's initial PIN value, expressed in ASCII (00000000 in decimal format). The last 4 bytes repeat the PIN value. (8 B)
0A	The current count of remaining verification attempts. (1 B)
0A	The number of total unblock mechanisms. (1 B)
31313131 31313131	The first 4 bytes contain the unblock PIN value, expressed in ASCII (11111111 in decimal format). The last 4 bytes repeat the unblock PIN value. (8 B)
0A	The maximum number of allowed unblocking attempts (10). (1 B)
0A	The current count of the remaining unblocking attempts (10). (1 B)

Update the User's PIN Value

When John uses the card for the first time, the application asks him to change the default PIN. The steps that follow show how to update the PIN value.

1 The application calls the `Select` command (described on page 177) to select the EF_{CHV1} (file 0000, located in 3F00).

2 The application updates John's PIN by calling the `Change CHV` command with the following APDU string:

CLA	INS	P1	P2	Lc	Data
`F0`	`24`	`00`	`01`	`10`	*old CHV value* || *new CHV value*

Data Field Values

`FF`	Activation byte, of which the 0 bit value sets the file to active (1) or inactive (0). To set a file to be active, you can use any hexadecimal value that corresponds to a decimal value with 1 in the 0 bit position, such as `FF`h (11111111) or `01`h (000000001). (1 B)
`FFFF`	RFU (2 B)
CHV value	The first 4 bytes contain the current PIN value, and the last 4 bytes contain the updated PIN value. (8 B)
`03`	The maximum number of allowed verification attempts (3). (1 B)
`03`	The current count of the remaining verification attempts (3). (1 B)
unblock CHV value	The first 4 bytes contain the current unblock PIN value, and the last 4 bytes contain the updated unblock PIN value. (8 B)
`03`	The maximum number of allowed unblocking attempts (3). (1 B)
`03`	The current count of the remaining unblocking attempts (3). (1 B)

Generating and Storing a Key Pair

In step 10, you generate the private and public keys of the RSA key pair you will use for digital signatures. The card automatically stores the key pair values: the private key algorithm in the $EF_{RSA\ PRI}$ file (which you created on page 218) and the public key modulus in the $EF_{RSA\ PUB}$ file (which you created on page 221). Once the card generates the key pair, you can retrieve the public key modulus.

Step 10: Generate and Store the Key Pair on the Card

To generate and store the key pair, follow these steps:

1 Call the `Select` command (described on page 177) to select the relevant RSA-PRI Key file. In this example, the relevant key file is 0012, located in the PKI DF (4F01) under the MF (3F00).

2 If you have not already established the CHV1 AC for the current context, call the `Verify CHV` command (as described on page 198) to authenticate the CHV1 AC.

3 Call the `Generate RSA Keys` command (as described on page 127), and use the following APDU string to create and add the key pair to the $EF_{RSA\ PRI}$ file you created on page 219.

CLA	INS	P1	P2	Lc	Data
`F0`	`46`	`03`	`80`	`04`	*public exponent* (bytes 4–1)

If you perform this command with a reader that is not completely ISO-compliant, the reader may time out before the operation is complete. To avoid this problem, include the key generation in a loop. This ensures that the host application does not interrupt the operation by asking the card for an answer too early.

The time required for key generation is 5 seconds on average, but may be any length of time from 2 seconds to 1 minute. The `Generate RSA Keys` command replaces the existing values for the specified key slot. The old key values are not saved.

Retrieve the Public Key Modulus

Call the `Read Binary` command, and use the following APDU string to retrieve the public key modulus from the $EF_{RSA\ PUB}$ file:

CLA	INS	P1	P2	Le	Output Data
`C0`	`B0`	`00`	`03`	`80`	*value of the public key modulus*

Generating and Retrieving a Digital Signature

In steps 11 and 12, you compute a digital signature on the card. You then retrieve the signature so you can attach it to data that you are sending or storing.

Step 11: Compute a Hash of the Data

You compute the signature with a SHA-1 hash. The command or commands you call to compute a SHA-1 hash are determined by the amount of data you want to hash:

- *For data that is 64 bytes or less in length:* Call the `SHA-1 Last` command.
- *For data that is more than 64 bytes long:* Call one or more `SHA-1 Intermediate` commands (one for each initial or intermediate 64-byte block), followed by a `SHA-1 Last` command to process the final data block (which must be no longer than 64 bytes).

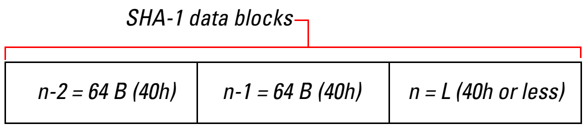
SHA-1 Intermediate

Call the `SHA-1 Intermediate` command (described on page 184) with the following APDU string:

CLA	INS	P1	P2	Lc	Data
`14`	`40`	`00`	`00`	`40`	*message* = [64 x k+1 — 64 x (k+1)]

NOTE *You must enter the message data in little-endian format (LSB first).*

$0 \le k \le n\text{-}2$



n = The final data block
n-1 = The next-to-final data block
L = Length of the last final block, an octet value between `08`h and `40`h.

SHA-1 Data Block Sequence

SHA-1 Last

To complete the hash, call the `SHA-1 Last` command (described on page 186) with the following APDU string:

CLA	INS	P1	P2	Lc	Data
`04`	`40`	`00`	`00`	`L`	*message* = [64 x (n-1)+1 — 64 x (n-1)+L]

`L` = Length of the final data block

NOTE *You must enter the message data in little-endian format (LSB first).*

Retrieving the Hash

To retrieve the hash, call the `Get Response` command (described on page 138) with the following APDU string:

CLA	INS	P1	P2	Le	Output Data
C0	C0	00	00	14	*hash value* (1–20 bytes)

NOTE *The card returns the hash output data in little-endian format.(LSB first).*

Step 12: Sign the Hash

Sign the hash by sending it to the card as part of an `RSA Signature (Internal Auth)` command (described on page 164). You then retrieve the signature from the card.

1 If you have not already established the CHV1 AC for the current context, use the following APDU to call the `Verify CHV` command (described on page 198).

CLA	INS	P1	P2	Lc	Data
C0	20	00	01	08	3030303030303030

The input data for the `Verify CHV` command is the CHV1 PIN value, 00000000, expressed in ASCII.

2 To sign the hash, call the `RSA Signature (Internal Auth)` command with the following APDU string:

CLA	INS	P1	P2	Lc	Data
C0	88	00	03	80	*padded hash* (1–128 bytes)

3 To retrieve the signature, call the `Get Response` command with the following APDU string:

CLA	INS	P1	P2	Le	Data
C0	C0	00	00	80	*signature* (1–128 bytes)

NOTE *The Cryptoflex card does not currently support any padding functions.*

Access Conditions

Appending a digital signature to transmitted or stored data is required by law to be a conscious act—the user must be aware that the signature is being added to data. To satisfy this requirement, the user typically must enter a PIN when signing a message. This can lead to security problems if a succession of user ACs are logged in to the card. You can avoid such problems by logging out the CHV1 AC after each signature.

To clear the verified CHV1 AC, call the `Logout AC` command with the following APDU string:

CLA	INS	P1	P2	Lc
F0	22	02	00	00

A

The Communication Interface

This section describes the format for the input and output data that passes between the reader and the card. This section also describes the events involved in resetting a card. Communication between the reader and a Cryptoflex card is based on the standardized, half-duplex ISO protocol T=0.

Data is sent to the card in command APDU format, which consists of a mandatory header (the CLA, INS, P1, and P2 components) and an optional body (the Lc, input data, and Le components). Data returned to the host is in response APDU format, which consists of an optional body (the response data) and a mandatory trailer (the SW1 and SW2).

The following terms identify the command parts and attributes described in this guide.

Command-Response Components and Attributes

CLA	Class of the command (1 byte). If you order custom-manufactured cards, you can specify custom command class values. Once manufacturing is complete, command class values cannot change.
INS	Instruction identifier of the command (1 byte). The CLA and INS bytes together uniquely identify each command.
P1, P2	Input parameters (1 byte each), which contain command-specific data.
Lc	Input parameter (1 byte) that specifies the number of input data bytes the host sends the card as part of the command. The Lc value is typically an explicit value between `01–FF`h. For example, if you send the card a command with 2 bytes of input data, the Lc value is `02`h.
Input Data	Command-specific data the host application sends the card, included with commands in send (S) or send/receive (S/R) mode.
Le	Input parameter (1 byte), that specifies the number of data bytes the host expects the card to return as response data. The Le value specifies the length of the return data field, regardless of the amount of available data. If the available data does not fill the specified Le field, the terminal byte(s) are null. Available data that does not fit in the specified Le field is not returned. *Note: For more information about command input/output formats, see "ISO Protocol Basics," on page 234.*
Response Data	Data the card returns to the host application, if the command mode is receive (R) or send/receive (S/R). The card returns response data in response to an internally called `Get Response` command.
SW1, SW2	*Status words:* Two bytes the card returns to the host application at the end of a command transaction, which indicate whether the command succeeded or failed. If the command failed, the status words typically indicate the type of failure. *(For more information about error codes, see "Status Words," on page 239.)*

Mode	A user-friendly term for the case of the command, which corresponds to the command cases specified by ISO 7816-4: • — (None): Send no input data and receive no response data — Case 1. Case 1 command components are CLA + INS + P1 + P2. The card returns status words (SW). • S: Send input data to the card — Case 2. Case 2 command components are CLA + INS + P1 + P2 + Lc + input data. The card returns SW. • R: Receive response data from the card — Case 3. Case 3 command components are CLA + INS + P1 + P2 + Le. The card returns SW and response data. • S / R: Send input data to the card, and receive response data — Case 4. Case 4 command components are CLA + INS + P1 + P2 + Lc + input data + Le. The card returns SW and response data. *Note: The input/output APDU protocols are illustrated on page 234.*

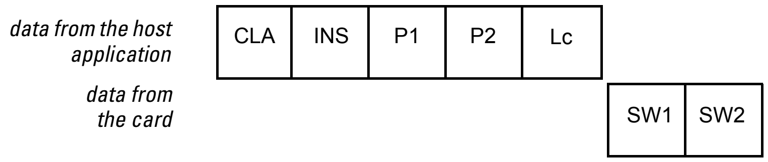
TPDU Protocol

APDU data is transmitted between the host and card by a transport protocol, or transmission protocol data units (TPDUs), which conform to the ISO 7816-3 specification. The Cryptoflex card uses the T=0 protocol—a byte-oriented protocol in which the smallest data unit that can be transmitted is a byte.

ISO Protocol Basics

This topic describes T=0 ISO input and output command formats for the exchange of APDU data between the card and host application (or terminal). This information will help you avoid ISO protocol errors. The illustrations that follow show how data bytes are exchanged between the host application and card.

Case 1: No Input or Output



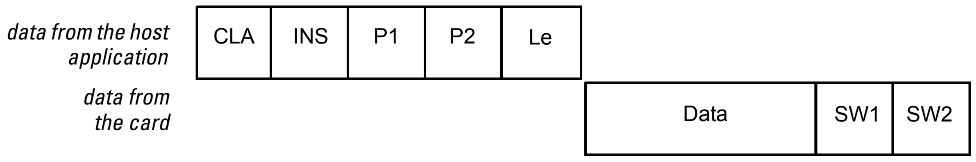
In this type of command, no input data or output data are exchanged between the host application and card. These events occur:

1. The host application sends the card a command with no input data, and an Lc value of `00`h (0 bytes). (Lc typically specifies the length of input data.)
2. The card returns two bytes of status word data that indicate the operation's success or failure. If the operation fails, the status words may identify the reason for the failure.

Commands with no input or output data are in neither send nor receive mode.

Example: `Generate DES Key` command on page 124

Case 2: Receive Mode



In a receive mode command, the host application sends no input data with the command, but receives output data from the card. These events occur:

1. The host application sends the card a command that is expected to generate or locate output data. The command's Le value indicates the expected length (number of bytes) of the output data.

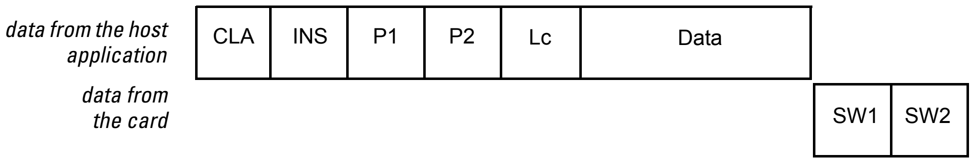
   The Le value is either:

   - An explicit value between `01`h and `FF`h, which specifies the actual number of data bytes the host application expects the card to return.
   - A value of `00`h — The convention for retrieving the maximum amount of response data, 256 bytes.

2. The card returns the output data and status words.

Example: `Get Challenge` command on page 136

Case 3: Send Mode



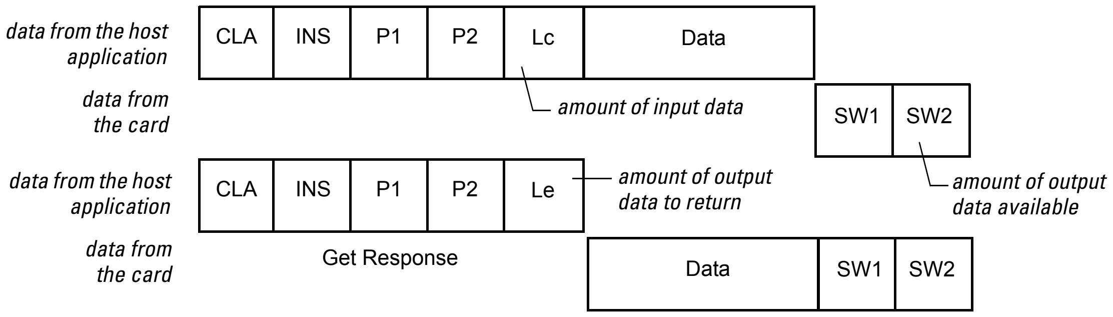
In a send mode command, the host application sends input data with the command, and receives no output data from the card. These events occur:

1 The host application sends the card a command that requires input data, but will not produce any output data. Lc specifies a *length* (number of bytes of input data).

2 The card returns status words.

Example: `Update Binary` command on page 190

Case 4: Send/Receive Mode


data from the host application
data from the card
CLA
INS
P1
P2
Lc
Data
amount of input data
SW1
SW2
data from the host application
data from the card
CLA
INS
P1
P2
Le
amount of output data to return
amount of output data available
Get Response
Data
SW1
SW2

In a Send/Receive (S/R) mode command, the host application sends input data with the command, and retrieves output data through a follow-up `Get Response` command, called automatically or explicitly. These events occur:

1. The host application sends the card a command with input data, which is intended to produce output data.
2. The card returns status words (SW1 and SW2) that indicate the initial operation's success or failure. If the operation succeeds, SW2 indicates the amount of data available for return.
3. The host follows with a `Get Response` command, in which the Le value is equal to or less than the amount of data reported in SW2.
4. The card returns the amount of data specified in Le.

Example: `Select` command on page 177

NOTE *For more information about APDUs, see the ISO 7816-3 and 7816-4 specifications.*

Resetting a Card

When a host application sends a reset signal to the card, the card's operating system performs these actions:

1. Logs out currently granted access rights.
2. Selects the root level in the file system. (Selects the master file as the current file.)
3. Sends the Answer To Reset (ATR) string to the reader.

NOTE *For information about ATRs, see page 14.*

B

Status Words

This appendix lists many of the status word (SW) bytes Cryptoflex cards can return in response to commands. Note that EMV commands have different SW patterns from standard commands.

Hex Value	Meaning
`6100` – `61`*xx*	*Command succeeded, data available for return* — Data are available for return if you send a follow-up `Get Response` command. The second status word byte (the value that appears in place of *xx*) indicates the number of data bytes available. Following an `RSA Signature Last` command, SW `6100` indicates `100`h (256) bytes of data are available. Send/Receive mode (case 4) commands return SWs of this type.
`6281`	*Read operation failed* — Data possibly corrupted.
`6283`	*Invalid context* — • File already invalidated (if attempting to invalidate), or • File not invalidated (if attempting to rehabilitate), or • Action not permitted because the selected file or MF is invalidated.
`6300`	*Key input error* — Authentication failed due to an error in the cryptogram, CHV key, CHV unblocking key, or cryptographic key.
`6581`	*Memory problem* — Write operation failed, possibly due to an EEPROM failure. (Extremely rare.)
`6700` – `67`*xx*	*Length error* — The Lc or Le value is unsupported or does not match the corresponding data. If the card returns a value other than `00`h for the second SW, enter that value as the Lc or Le value.
`6981`	*Missing key or key slot error* — No relevant cryptographic key, CHV key, or key file. For RSA key generation, indicates the key slot is the wrong length for the key.

Hex Value	Meaning
`6982`	*Invalid or no key presented* — Access condition not satisfied due to an invalid key / CHV key, or because no key / CHV key was presented.
`6983`	*Blocked or missing key* — Typically, the necessary key is blocked. For an `RSA Signature Last` command, indicates the key is not found.
`6985`	*AC failure or misconfigured key* — AC failed because the host did not send a `Get Challenge` command. For internal authentication, indicates the key's input parameters are not self-consistent. *(For `Read Record EMV`, SW `6985` indicates context is inappropriate.)*
`6986`	*Inappropriate context* — Selected file is inappropriate for the action.
`6A80`	*Inappropriate context* — Selected file is inappropriate for the action, or • File ID already exists (if creating a file), or • File type does not support the record length (if creating a file), or • Search string not found (if calling `Seek`), or • DF is not empty (if deleting DF).
`6A82`	*Missing FID or file* — File ID not found, or • No more files to describe (for `Dir Next`) • Key file(s) missing (for `Generate RSA Keys`)
`6A83`	*Insufficient space or out of range* — EF, DF, or card has insufficient EEPROM to create the record/key/file, or • Key to be overwritten is the wrong length (if generating RSA keys) • No more records or ID not found (if reading/updating record)
`6A84`	*Insufficient space* — Insufficient EEPROM available for the action.
`6A86`	*EMV P1 / P2 error* — Incorrect value for P1, P2, or both (if reading EMV record).
`6B00`	*P1 / P2 error* — Incorrect value for P1, P2, or both. Can also indicate the offset defined by P1 and P2 is out of range.
`6C`*xx*	*EMV inappropriate context* — Selected file is inappropriate for the action (if reading EMV record).
`6D00`	*Command instruction error* — Unknown command instruction.
`6E00`	*Command class error* — Incorrect class for command.
`6F00`	*Unspecified problem* — Unidentified technical problem.
`9000`	*Successful completion* — Command completed successfully.
`9850`	*Value limitation* — Increase/decrease cannot be performed due to limitation of maximum/minimum value.

Technical Details and Procedures

Physical and Electrical Characteristics

SchlumbergerSema smart cards have the following physical dimensions:

Length	85.5 mm	**Width**	54.0 mm	**Thickness**	0.80 mm

Electrical Contacts

SchlumbergerSema smart cards support eight contacts, six of which are currently used. The positions of these contacts comply with part 2 of the ISO/DIS 7816 standard. The minimum contact surface area is 1.7 mm x 2.0 mm. Contact dimensions match those of a standard credit card and comply with part 1 of the ISO/DIS 7816 specification.

Contact Pin-Out Description

Contact	Function
C1	Vcc supply voltage 5V +/- 0.5V
C2	RST (reset)
C3	CLK (clock)
C4	RFU
C5	GND (ground)
C6	Not used
C7	I/O bidirectional line
C8	RFU

Power Supply

The maximum power supply is 50 mA at 5 MHz. The typical power supply is 5 mA at 5 MHz. To reduce noise, follow these electrical recommendations:

- Couple the Vcc with a capacitor of 0.1 µF.
- Use a pull-up resistor of 20 kΩ for the input/output pin.
- Use a cable of 15–30 cm in length between the prober and the PC.

Changing the Reader-to-Card Data Transmission Speed

By default, data travels between the card reader and the card at a speed of 9600 baud, if f_{clock} = 3.57 Mhz. You can increase the data transmission speed, provided your card reader can support the higher speed.

To change the card's protocol parameter, send a PPS command to the card after the Answer To Reset (ATR), as described in the following tables.

If the card receives a correct PPS command, it returns `FF` as confirmation. If the card receives an incorrect PPS card command, it returns no value. After a successful PPS request, data transmits from the reader to the card at the speed specified by the protocol parameter for the remainder of the card session.

Protocol Parameter Selection for a Cryptoflex 16K Card

Baud Rate	Clock Frequency	PPS Command	ISO Parameters
19,200	3.57 MHz	`FF 10 12 FD`	T=0, Fi=372, Di=2, F1=1, D1=2
38,400	3.57 MHz	`FF 10 13 FC`	T=0, Fi=372, Di=4, F1=1, D1=3
55,800	3.57 MHz	`FF 10 94 7B`	T=0, Fi=512, Di=8, F1=9, D1=4
76,8000	3.57 MHz	`FF 10 15 FA`	T=0, Fi=372, Di=8, F1=1, D1=4

3

3DES — triple data encryption standard.   A symmetric key system that uses the DES algorithm to encrypt and decrypt data. The most common form of double-key 3DES processes data three times—forward (encrypted) with the first key, backward (decrypted) with the second key, and forward again (encrypted) with the first key. Decryption reverses those steps.   *Also see: DES.*

3F00 — The file ID for the master file (root directory) on a Cryptoflex card. 3F00 is a reserved file ID, which you must not assign to any other file.

3F11 — A dedicated file ID reserved for internal use on a Cryptoflex card. Do not assign this reserved file ID to any file.

A

AC — access condition.   A value assigned to Cryptoflex card commands or to specific directories or elementary files. Setting ACs prevents execution of the protected commands unless the specified security condition is satisfied. ACs protect card resources on a localized basis—they protect commands on particular card files.

ADF — application dedicated file.   A Cryptoflex card file that contains information about an EMV application.

ALW — Always.   A Cryptoflex access condition (AC) that sets the command to be always possible. As with all ACs, *Always* can only affect commands in a particular context.   *Also see: AC, AUT, CHV, NEV, PRO.*

APDU — application protocol data unit.   A sequence of hexadecimal values that conform to the low-level format for data exchanged between the host application and the card (through a card reader or terminal). Command and response APDU formats are defined by the ISO 7816-4 specification.   *Also see: case, CLA, INS, Lc, Le, mode, P1/P2/P3.*

API — application programming interface.   Software that defines the calling conventions an application uses to gain access to lower-level services performed by an operating system or another application.

ATR — answer to reset. The card-specific data string the card sends to the host (through the reader) when power is first applied to the card. An ATR signals the reader that the power-up was successful, and sends identification and (possibly) protocol data.

AUT — Authenticate. A Cryptoflex access condition (AC) that sets a command to be possible only if the appropriate AUT key is currently authenticated. The host establishes AUT by executing a successful `Verify Key` or `External Authenticate Using DES` command. As with all ACs, AUT affects commands in a particular context. *Also see: AC, ALW, CHV, NEV, PRO.*

authentication — In cryptography, authentication is the ability of the sender and receiver to confirm each other's identity. *Also see: mutual authentication.*

C

CA — *See certificate authority (CA).*

case — APDU command type, as designated by the ISO 7816-4 specification. The type of input and output data included by each case of command is described below:

- Case 1 — No input data and no response data. Also referred to as a no-mode command (—).
- Case 2 — No input data, but elicits response data (which the card either returns automatically or in response to a follow-up `GetResponse` command). Also referred to as a Receive mode command (R).
- Case 3 — Input data, but no response data. Also referred to as a Send mode command (S).
- Case 2 — Send/Receive (S/R). Input data and potential response data (which the card either returns automatically or in response to a follow-up `GetResponse` command). Also referred to as a Send/Receive mode command (S/R).

Also see: APDU, Lc, Le.

CBC — cipher block chaining. A DES mode of encryption and decryption in which data blocks (plaintext or ciphertext) are bitwise exclusive-ORed with previous data blocks. The resulting data is encrypted or decrypted with a DES or 3DES key. In this way, each data block is affected by the previous blocks. CBC makes it easy to determine if a message has been altered. *Also see: EBC.*

certificate, digital certificate — A message that contains a user's public key, digitally signed by a certificate authority (CA) to assure the recipient that the key belongs to the user. The recipient can use the CA's signature to verify the authenticity of the certificate. Digital certificates may be kept in registries, which authenticated users search to find other users' public keys.

certificate authority (CA) — A trusted entity that maintains information about a client user's identities and issues a digital certificate to each user, which verifies to other parties the authenticity of the user's claimed identity. A CA may be an independent enterprise or an arm of the user's company. The usefulness of the endorsement depends on the certificate issuer's standing as a known and trusted authority. As a result, CAs may be endorsed by more widely known CAs, creating a chain of trust.

challenge — A random number. A challenge may be provided by one party, encrypted by a second party, and returned to prove the first party's identity.

checksum — A count of the number of bits in a transmission unit, typically included with the transmission so the receiver can make sure the correct number of bits arrived.

CHV — cardholder verification. On a file-based smart card operating system, a CHV file holds a CHV key, or personal identification number (PIN). The PIN is used to verify a cardholder's or card administrator's identity.

Cryptoflex cards support CHV access conditions that set commands in a specific context to be executed only if the appropriate CHV1 or CHV2 user is logged in. *Also see: AC, ALW, AUT, NEV, PRO.*

CLA — class. The first byte of an APDU, which identifies the command class. The class and instruction (INS) bytes uniquely identify the command type. *Also see: APDU, INS, P1/P2/P3.*

confidentiality — In cryptography, confidentiality means restricting access to the meaning of information, so that the information can be understood only by the intended recipient. *Also see: authentication, cryptography, integrity, nonrepudiation.*

COVE — Cryptographic Object Viewer and Editor. A Cyberflex Access SDK utility for personalizing cards and setting up keys and key files on the card.

CR — characters remaining.

CRT — Chinese remainder theorem.

CryptoAPI — Cryptographic Application Programming Interface. A PC/SC-supported high-level programming environment that routes high-level function calls to a *cryptographic service provider* (CSP) as an interface to card-based services. CryptoAPI is included in all 32-bit Windows software, and provides the cardholder with an easy-to-use interface for identity checks and digital signature creation. *Also see: CSP.*

cryptogram, cryptograph — A block of encrypted data. Smart cards use cryptograms to demonstrate possession of a secret key without revealing the key itself.

cryptography — The science of information security, in which plaintext (ordinary text) is encrypted into ciphertext, then decrypted.

cyclic elementary file (CY) — A cyclic elementary file is a data file that contains a ring of records of equal length. (Used primarily on a file-based smart card operating system.) Cyclic elementary files (EFs) are especially useful for storing "last ten" operations, such as dates or transaction amounts. Since cyclic EFs map data into a series of locations in EEPROM rather than into a single physical address, this type of file is ideal for records that require an unusually high number of overwrite operations. *Also see: elementary file (EF), fixed-length linear elementary file (EF), linear elementary file (EF), variable-length linear elementary file (EF).*

D

dedicated file (DF) — On a file-based smart card operating system, a dedicated file is a smart card directory that contains other DFs or elementary files (EFs). *Also see elementary file (EF), master file (MF).*

DES — Data Encryption Standard. A symmetric algorithm (specified in ANSI X3.92 and X3.106) that uses a single key for both encryption and decryption. DES is suitable for use when keys are distributed and stored dependably and securely, or when keys are exchanged between systems that have already authenticated each other (and the key life is restricted to the session or transaction). DES is commonly used to protect data from eavesdropping during transmission. *Also see: 3DES.*

digital certificate — *See certificate.*

digital signature — A digital string generated from a private key (such as an RSA private key). Since only the key's possessor has the key, the signature can come only from that person. A recipient of a message (encrypted or not encrypted) that has a digital signature attached can use the sender's public key to verify it. The receiver can verify the sender's identity and can determine whether the message was altered after it was signed. In addition to its other characteristics, a digital signature is difficult to repudiate.

E

EBC — Electronic Book Code. A mode of DES encryption and decryption, in which a plaintext or ciphertext block is encrypted independently with a DES or 3DES key (rather than bitwise exclusive-ORed with the previous block, as in CBC mode). EBC mode is as secure as the underlying block cipher, but unlike CBC, does not conceal plaintext patterns. EBC allows easy parallelization, so it takes less time than CBC. *Also see: CBC.*

EEPROM — Electrically erasable programmable read-only memory. Microprocessor memory that does not lose its data when power is removed from the microprocessor, and that can be erased and reprogrammed repeatedly with applied electrical voltage.

e-gate — commercial name of the FMSC smart card, capable of interfacing with a standard ISO 7816 reader or with the USB port of a PC.

elementary file (EF) — An elementary file is a data file located under a dedicated file (directory) on a file-based smart card operating system. *Also see: cyclic elementary file (CY), dedicated file (DF), fixed-length linear elementary file (EF), linear elementary file (EF), master file (MF), variable-length linear elementary file (EF).*

EMV — Europay / MasterCard / Visa. Standards designed to provide interoperability between a range of smart card hardware and software programs in the payment industry. EMV standards were developed by an alliance of bankcard associations (sometimes called *EMV'96*). The Cryptoflex card is EMV Light-compliant, meaning that it supports the most basic operations needed for EMV applications.

EOF — end of file.

external authentication — The process a host-side application uses to establish its credentials for gaining access to a smart card. For example, the host asks the card for a challenge (a random number), which the host application encrypts. The card performs a parallel operation on the challenge string and compares the resulting cryptograms. If the cryptograms match, the host has proven that it possesses a key stored on the card. *Also see: challenge, internal authentication.*

F

FID — file identifier.

file-based smart card — A smart card whose operating system is based on a file system. *Also see: Global Platform specifications, Open Platform smart card, SCOS.*

fixed-length linear elementary file (EF) — A smart card data file that contains records of a fixed length. Used primarily on a file-based smart card. *Also see: cyclic elementary file (CY), elementary file (EF), linear elementary file (EF), variable-length linear elementary file (EF).*

FS — file size.

H

hash — A digest (typically of a fixed length) of a longer string of characters or numbers. SHA-1 is a widely used one-way hash code. *Also see: MAC, SHA.*

I

ICC — integrated circuit card. ISO term for a smart card.

IFD — interface device. ISO term for a smart card reader. *Also see: CAD.*

inherited CHV / key file — See *relevant CHV / key file.*

INS — instruction. The second byte of an APDU, which identifies the command instruction. The class (CLA) and instruction bytes uniquely identify the command type. *Also see: APDU, CLA, P1/P2/P3.*

integrity — In cryptography, integrity is the ability to detect whether transmitted or stored information has been altered. *Also see: authentication, confidentiality, cryptography, nonrepudiation.*

internal authentication — The process for establishing the card's credentials with the host system. For example, the host sends the card a challenge (a random number), which the card encrypts. The host application performs a parallel operation on the challenge string and compares the resulting cryptograms. If the cryptograms match, the card is proven to be trustworthy.

IOP — interoperability layer. The SchlumbergerSema foundation software that acts as a translator for calls from the mid-level card implementations to the Microsoft Windows Resource Manager.

ISO 7816 — International Standardization Organization specification #7816. A set of standards for smart card physical attributes, data elements, basic commands, and security architecture.

IV — initialization vector. A value used in DES block encryption and decryption.

key — A number or character string used for security purposes:

- A *cryptographic key* is a number chosen for mathematical properties that are useful in encryption and decryption. The key length typically determines how strong the key is—how difficult it is to decrypt the ciphertext without having the key.
- An *identification key* is used to prove identification, such as a PIN the cardholder or card administrator presents that must match a stored value.

key pair — A private and public key set: complementary components of a key (such as an RSA key) used for asymmetric encryption and decryption. RSA keys are used for such purposes as secure transmission of data and for digital signatures. You use the private key to decrypt text that has been encrypted with your public key by someone else (who can find out what your public key is from you or from a certificate authority's public directory). In addition to the role in encrypting messages, RSA key pairs enable you to authenticate yourself to others by using your private key to encrypt a digital certificate. When the message arrives, the recipient uses your public key to decrypt it.
Also see: digital signature, private key, public key, RSA.

L

Lc — length of command data. The fifth byte of an APDU command in Send mode (case 3) or Send/Receive mode (case 4). The Lc specifies the length of the input data included immediately after the Lc byte. In a Receive mode command, the Lc may also be referred to as parameter 3 (P3). *Also see: Le.*

Le — length of response data. The APDU component byte included at the end of a Receive mode command (case 2) or Send/Receive mode command (case 4) command, which specifies the length of the data you expect the card to return. In a Receive mode command, the Le may also be referred to as parameter 3 (P3). *Also see: Lc.*

linear elementary file (EF) — A smart card data file that contains subdivisions called *records*, which are either fixed or variable in length. (Used primarily with file-based smart card operating systems.) *Also see: fixed-length linear elementary file (EF), variable-length linear elementary file (EF).*

LF — *See fixed-length linear elementary file (EF).*

LOFB — length of fill block. Parameter in the PRO authentication process for a Cryptoflex card. The LOFB is the number of bytes needed to make the command instruction block evenly divisible by 8 bytes.

LOUD — length of useful data. Parameter in the PRO authentication process for a Cryptoflex card. For example, the LOUD bytes in an operation to change a PIN number are the bytes of the new PIN data.

LSB — least significant byte.

LSN — least significant nibble.

LV —variable-length linear elementary file. A file format used primarily on a file-based smart card. *Also see variable-length linear elementary file (EF).*

M

master file (MF) — A special dedicated file that is the root of the file system on a file-based smart card. (Directories are referred to as *dedicated files*, or DFs.) The master file (MF) can contain dedicated files (other directories) and elementary files (data files). The MF's reserved file identifier is 3F00. *Also see: dedicated file (DF), elementary file (EF).*

mode — Shorthand designator for the type of APDU command, which corresponds to one of the cases defined by the 7816-4 specification, as described below:

- — (no mode) — Case 1 command: Includes no input data and does not create or locate response data from the card.
- R — (Receive mode) Case 2 command: Includes no input data, but prompts the card to return data.
- S — (Send mode) Case 3 command: Includes input data, but does not create or locate response data from the card.
- S/R — (Send/Receive mode) Case 4 command: Includes input data and prompts the card to return data.

modulus — *(abbreviated as mod)* An RSA key element. Mathematically, the modulus is the number by which a logarithm in one system must be multiplied to obtain the corresponding logarithm in another system.

MSB — most significant bit.

MSN — most significant nibble.

mutual authentication — The process of confirming the identity of two communicating parties, such as the host system and a smart card application (the on-card agent that is currently processing incoming commands). On an Open Platform Cyberflex Access card, mutual authentication is essential for establishing a secure channel. *Also see: keyset, security domain.*

N

NEV — Never. A Cryptoflex access condition (AC) that sets the command to never be possible under any conditions. As with all ACs, *Never* affects commands in a particular context. *Also see: AC, ALW, AUT, CHV, PRO.*

nonrepudiation — In cryptography, nonrepudiation is assurance that the originator of the information cannot later deny that he or she was the information source. *Also see: authentication, confidentiality, cryptography, integrity.*

NR — number of records.

O

octet — *As applied to data:* evenly divisible into 8-byte blocks.

operating system — On a smart card, the operating system is an application that can execute a set of instructions which form the basic operations that enable a card program to run. *Also see: Card Manager, file-based smart card, Global Platform specifications, Open Platform smart card.*

P

P1, P2, P3 — Parameters 1, 2, 3. The third, fourth, and fifth bytes of an APDU, which supply extra information about the command. The type of data P1 and P2 contain is command-specific. P3 is typically the length of input data (Lc) or length of expected response data (Le). *Also see: APDU, CLA, INS.*

padding — Extra characters or bytes inserted into data to standardize the size of the data block. For example, you might apply padding to data to perform an operation that requires octet data (data that is evenly divisible by 8 bytes).

PC / SC — Personal Computer / Smart Card. A PC-based, open architecture for interoperation between hardware and software components from different vendors. The PC/SC architecture was developed by a group of smart card and PC operating system vendors, including SchlumbergerSema, Microsoft, Siemens Nixdorf, HP, and CP8 Transac. For more information about the PC/SC workgroup, see *http://www.pcscworkgroup.com.*

PIN — personal identification number. An alphanumeric string which can be used as a password to establish person-to-card authentication. *Also see: global PIN.*

PKCS #11 — Public Key Cryptography Standard #11. An RSA Laboratory-sponsored set of intervendor standard protocols developed to facilitate secure information exchange. Cyberflex Access series cards support PKCS #11-compliant card programs by supplying a library of functions that provide cryptographic and security services to the PKCS #11 interface, Cryptoki.These functions are part of the SchlumbergerSema Smart Card middleware.

PKI — Public Key Infrastructure. An infrastructure model for exchanging data and money through the use of a public and private key pair, which is obtained and shared through a trusted authority. PKIs provide for digital certificates that can identify individuals and organizations, and for directory services that can store and revoke the digital certificates. An Internet standard for PKI is currently underway.

PPS — protocol parameter selection.

private key — The secret key component of an asymmetric key pair, such as an RSA key pair. The private key value is known only by its owner—it is never shared with anyone. You can use the key pair for such purposes as email encryption and decryption, and for digital signatures. *Also see: key pair, public key, RSA.*

program — Software on the host system or smart card designed to execute special operations.

PRO — protected command mode. On a Cryptoflex card, the PRO AC prevents the specified command (in a particular context) from executing unless it has a verified digital signature attached. *Also see: AC, ALW, AUT, CHV, NEV.*

PSE — Payment System Environment. An EMV structured transaction environment, compliant with the EMV'96 ICC Specification for Payment.

public key — The publicly available key component of an asymmetric key pair, such as an RSA key pair. The public key is published and available to anyone who wants to send an encrypted communication to the owner of the private key. *Also see: key pair, private key, RSA.*

R

R mode — receive mode. *See: Le, mode.*

RA — *See: registration authority (RA).*

registration authority (RA) — registration authority. An entity that acts as the verifier for a certificate authority before a digital certificate is issued to a requestor. *Also see: certificate authority.*

relevant CHV / key file — The file on a Cryptoflex card that contains the PIN or key that protects the currently selected EF (data file) or DF (directory). A key file is relevant to these files:

- DF that contains the protecting key file
- EFs and DFs contained in the DF that holds the key file
- Files located lower in the file hierarchy, until the point that a new key file of the same type occurs

RFU — reserved for future use.

RL — record length.

RNG — random number generator. An agent that generates 0s and 1s in a sequence designed so that, at any point, the next bit cannot be predicted from analyzing the previous bits. In some cases, a pseudo random number generator may be used. In this case, the key may be an apparently random string generated from a relatively small random seed.

RSA — A widely used *asymmetric key system* known by the initials of its originators: Rivest, Shamir, and Adleman. In this guide, RSA stands for the 512-bit, 768-bit, or 1024-bit encryption algorithm used by Cyberflex Access series smart cards. RSA uses a public and private *key pair*, with the *public key* published openly, while the *private key* remains secret. *Also see: key pair, private key, public key.*

S

S mode, S/R mode — send mode, send/receive mode. *See: Lc, Le, mode.*

SCOS — SchlumbergerSema Card Operating System. The operating system on a Cryptoflex card. The term SCOS is also used to refer to the system for adding customer-specified soft masks to Cryptoflex cards' EEPROM at the factory.

session key — A key that is generated during a card session, which expires with the card session.

SFI — short file identifier. A shorthand designation for an EMV application elementary file on a Cryptoflex card.

SHA, SHA-1 — Secure Hash Algorithm. An algorithm similar to the MD4 family of hash functions, which is specified in ANSI X9.30. SHA-1 is a technical revision of SHA (FIPS 180).

signing key — *See key pair.*

T

TPDU — transmission protocol data unit. Basic element of the lower-level protocol for exchanging APDU data between the host application and the card. TPDU protocols are defined by the ISO 7816-3 specification. The T=0 protocol is an asynchronous, byte-oriented, half-duplex transmission protocol in which a byte is the smallest transmissible data unit. The T=1 protocol is an asynchronous, block-oriented, half-duplex transmission protocol.

transparent elementary file (EF) — A file that contains a single data envelope, useful for storing objects such as keys or user identification data.

transport key(s) — The key or keys used to unlock a new Cryptoflex card.

triple-DES — *See 3DES.*

V

variable-length linear elementary file (EF) — A data file found on a Cryptoflex card that can contain records of varying lengths. Variable-length linear EFs conserve valuable EEPROM memory when you have data of varying lengths to store in the same file. On the other hand, variable-length linear EFs require more seek time for read and write operations, and require slightly larger headers than fixed-length linear EFs. *Also see: cyclic elementary file (CY), elementary file (EF), fixed-length linear elementary file (EF), linear elementary file (EF).*

W

weak key — A key with regularities that result in a poor level of encryption. For cards that generate DES keys check for four weak and twelve semi-weak DES keys, which the card always discards.

Windows for Smart Cards — A smart card runtime environment developed by Microsoft, based on Windows and Visual Basic tools and principles.

Index