BlueZ Secure Systems
IBM Zurich Research Laboratory

BlueZ PKCS#15
An implementation for Open Platform Java Cards

(Rev 2.01, 30/06/2003)

Technical contact: javacard@zurich.ibm.com

This paper is for general guidance only.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any references to any IBM product, program or service is not intended to imply that only IBM's product, program or service may be used. Any functionally equivalent product, program or services may be used instead.

Table of Contents

1 Abstract

This documentation provides detailed information about the BlueZ PKCS#15 implementation. After a brief introduction into the architectural concept of the application it mainly documents its features, usage and command APDU specification. It is not intended to aim as an introduction into the PKCS#15 or ISO 7816-x standards.

2 Introduction

The purpose of the PKCS#15 Cryptographic Token information Syntax Standard ([1]) is to promote interoperability between host applications and cryptographic tokens, such as smart cards, with respect to security-related information stored on such tokens. For example, the holder of a PKCS#15 compliant smart card should be able to present the card to any application running on any host connected to any smart card reader and successfully use it to present his credentials or authenticate himself.
To achieve this the standard specifies certain properties of an application residing on a smart card, which has support for an ISO/IEC 7816-4/-5/-6 hierarchical file system. These properties are basically the file format (the content of the files) and to some extend the file system structure of a PKCS#15 compliant application. The standard does, however, not define its own set of commands to access these files but refers to the ISO standards ([2], [6], [7]) providing file system commands and in addition to that also commands for cryptographic operations.
The BlueZ PKCS#15 application is a JavaCard implementation of these standards. It provides an emulation of a PKCS#15 compliant ISO file system layout in combination with the ISO commands mentioned above. Features as dynamic memory management and a large variety of cryptographic algorithms lead to a very flexible implementation, which allows to be set up in accordance with different application profiles.

3 Implementations

Three implementations of the BlueZ PKCS#15 application are available. One, which is fully Java Card 2.1.1 compliant and can run on any Java Card 2.1.1 implementation providing sufficient resources. The second is optimized for the BlueZ JCOP platform and offers more performance and functionality (see table of supported algorithms) as well as significant savings in resource consumption. Typically the JCOP platform (e.g. JCOP31bio) comes with this PKCS#15 application already included in the ROM mask and thus the complete EEPROM space remains free for application use (e.g. keys, certificates, data etc.). Finally, the third implementation is a variant of the PKCS#15 application for JCOP which is about to received FIPS 140-2 certification as part of the JCOP21id product.

4 The File System

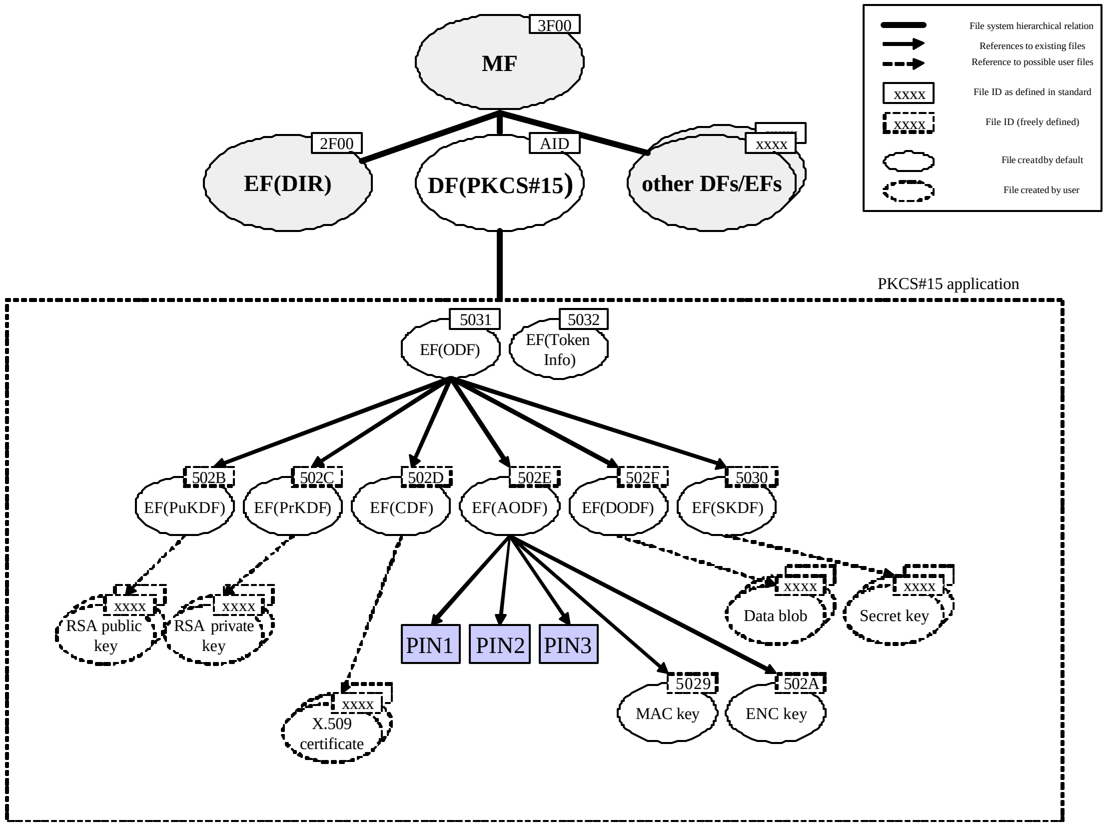
The picture below illustrates a typical instance of the BlueZ PKCS#15 application. The PKCS#15 standard defines that compliant IC cards should support direct application selection as defined in [2] and [4] (the full AID is to be used as parameter for a 'SELECCT FILE' command). This means that the basic parts of a file system-based card like the MF and an EF(DIR) are not necessary for application selection. Thus this implementation does not emulate an MF or EF(DIR). The direct selection of the PKCS#15 application via its AID makes it to the current selected applet on an Open Platform Java Card and subsequent commands are handled by the BlueZ applet, which then emulates the relevant part of the file system.

All files under the DF(PKCS#15) directory are allocated out of one file system memory pool. The size of this pool can be defined at application installation time. The application manages this memory pool so that newly created files (using the CREATE FILE command) are placed into the free space of the pool whereas on file deletion (using the DELETE FILE command) the space is freed. The application also performs memory defragmentation during file deletion. The files represented by the solid ovals are allocated by default during application installation. The EF(ODF) and EF(Token Info) are mandatory and have fixed file identifiers. Other directory files are in general optional but always present in this implementation. Their file identifiers are freely chosen by the implementation.

After application installation and initialization the issuer or the user can create (or delete) files (e.g. RSA key pairs, certificates, etc.). The dashed ovals represent such dynamically allocated (non-default) files. All files are direct children of the DF(PKCS#15).

Whereas the lines represent hierarchical relations in the file system the arrows are referencing relations in PKCS#15 terms. This means the ASN.1 encoded entries in the directory files, maintained by the host application, include references to files. These references are in fact the file identifiers. See [1] for details on the file format. One exception is the references to the authentication objects (PIN1, PIN2, PIN3). The PINs are not files in the file system and therefore not referenced by file identifiers but by logical PIN reference numbers. The fact that the token provides three PINs is a fixed property and cannot be configured. PIN 3 acts as the security officer PIN (soPIN), which allows unblocking or changing the user PINs (PIN1 and PIN2).

The two files, "MAC key" and "ENC key", are always present and host DES keys available for setting up a secure channel between the application and the host computer. The setting up of a secure channel is based on mutual authentication involving the MAC key. Upon authentication, data can be sent over the secure channel either in plain or encrypted and MACed. These variants can be used as authentication object. For example, it is possible to allow updating a file only via an encrypted secure channel.



5 Application Selection

As mentioned in the previous chapter the BlueZ PKCS#15 application is selected directly via its AID as described in [2]. Since the AID is $A000000063504B43532D3135_{HEX}$, the command to select the DF(PKCS#15) is $00A404000CA000000063504B43532D3135_{HEX}$. Upon receipt of the select command the token responds with its FCI (File Control Information), which conveys some information about the token and its current state.

5.1 DF(PKCS#15) FCI

Byte	Value[HEX]	Remarks
1	6F	FCI tag
2	XX	FCI length
3	81	No. of bytes used in the file system tag
4	02	Length
5	XX	No. of bytes used in the file system (high byte)
6	XX	No. of bytes used in the file system (low byte)

7	82	File descriptor byte tag
8	01	File descriptor byte length
9	38	File descriptor byte (DF)
10	84	DF name tag
11	02	DF name length
12-23	XX	PKCS#15 AID
24	86	Proprietary security attributes tag
25	03	Length
26	XX	Tries remaining PIN 1
27	XX	Tries remaining PIN 2
28	XX	Tries remaining PIN 3
29	85	Proprietary information
30	XX	Length
31-34	XX	Unique chip ID
35	XX	Number of files in the file system
36	XX	File identifier 1 (high byte)
37	XX	File identifier 1 (low byte)
. . .	. . .	. . .
XX	XX	File identifier N (high byte)
XX	XX	File identifier N (low byte)
XX	90	SW1
XX	00	SW2

6 Application Installation

The BlueZ PKCS#15 application is installed in accordance with [9]. Since during installation time certain application resources (e.g. the default file system layout) are allocated the following application specific install parameters must be passed.

Byte	Value[HEX]	Remarks
1	C9	Application specific install parameters tag (see [9])
2	1A	Length
3	XX	Security attributes for default files (byte 1)
4	XX	Security attributes for default files (byte 2)
5	XX	Security attributes for default files (byte 3)
6	XX	Overall file system space(high byte)
7	XX	Overall file system space(low byte)
8	XX	Size EF(PuKDF) (high byte)
9	XX	Size EF(PuKDF) (low byte)
10	XX	Size EF(PrKDF) (high byte)
11	XX	Size EF(PrKDF) (low byte)

12	XX	Size EF(CDF) (high byte)
13	XX	Size EF(CDF) (low byte)
14	XX	Size EF(AODF) (high byte)
15	XX	Size EF(AODF) (low byte)
16	XX	Size EF(DODF) (high byte)
17	XX	Size EF(DODF) (low byte)
18	XX	Size EF(SKDF) (high byte)
19	XX	Size EF(SKDF) (low byte)
20	XX	Size EF(ODF) (high byte)
21	XX	Size EF(ODF) (low byte)
22	XX	Size EF(TokenInfo) (high byte)
23	XX	Size EF(TokenInfo) (low byte)
24	01-7F	Retry counter limit for PIN 1
25	01-7F	Retry counter limit for PIN 2
26	01-7F	Retry counter limit for PIN 3
27	XX	Zero means RSA private keys are importable, else otherwise
28	01-05	Defines the authentication object, which protects the CREATE FILE Command Values: 1 - PIN 1 2 - PIN 2 3 - PIN 3 4 - AUTH 5 - SM

7 Application Initialization and Personalization

The application is initialized base on the Open Platform secure channel as described in the section “Token Personalization Commands”. Further personalization (e.g. key generation, certificate upload etc.) can be done at any later point in time based on the standard commands described in this document.

8 Supported Cryptographic Algorithms

The following table describes the algorithms supported by the token and their algorithm identifier, which can be used in the MANAGE SECURITY ENVIRONMENT command. It also indicates in which CRT and thus in which command the algorithm identifiers are to be used.

Algorithm Identifier [HEX]	Algorithm	Key length [bit]	Block length [bit]	Automatic padding	Automatic digest	RSA public exponent	CRT

01	DES MAC	64(56)	64	-	-	-	CCT
03	DES ECB	64(56)	64	-	-	-	CT
04	DES CBC	64(56)	64	-	-	-	CT
11	DES3 MAC	128(112)	64	-	-	-	CCT
13	DES3 ECB	128(112)	64	-	-	-	CT
14	DES3 CBC	128(112)	64	-	-	-	CT
21	AES MAC[1]	128	128	-	-	-	CCT
23	AES ECB[1]	128	128	-	-	-	CT
24	AES CBC[1]	128	128	-	-	-	CT
31	AES MAC[1]	192	128	-	-	-	CCT
33	AES ECB[1]	192	128	-	-	-	CT
34	AES CBC[1]	192	128	-	-	-	CT
41	AES MAC[1]	256	128	-	-	-	CCT
43	AES ECB[1]	256	128	-	-	-	CT
44	AES CBC[1]	256	128	-	-	-	CT
69/02	RSA	512-2048[3]	(key length)	PKCS#1	-	-	CT/DST
6F/12/22	RSA	512-2048[3]	(key length)	PKCS#1	Digest Info (SHA/MD5)	-	DST
6A/00	RSA	512-2048[3]	(key length)	-	-	-	CT/DST[2]
6B	RSA[1]	512-2048	(key length)	PKCS#1	SHA-1	-	DST
6C	RSA[1/4]	512-2048	(key length)	PKCS#1	MD5	-	DST
6D	RSA[1]	512-2048	(key length)	ISO9796	SHA-1	-	DST
6D	RSA key generation	512-2048[3]	(key length)	-	-	3	DST
6E	RSA key generation	512-2048[3]	(key length)	-	-	65537 (Fermat-4)	DST
57	SHA-1[1]	-	512	-	-	-	HT
58	MD5[1/4]	-	512	-	-	-	HT

[1] supported on the JCOP platform only.
[2] PKCS#1 padding must be done by the host.
[3] RSA key length of 512-2048 bits is supported on the JCOP platform only, otherwise the key length is fixed to 1024 bits.
[4] Not supported in the FIPS 140-2 certified version of BlueZ PKCS#15 (included in JCOP21id).

9 Key Format In Files

The format used by the BlueZ PKCS#15 token to store cryptographic keys in transparent files is described in the following. If keys are imported using the UPDATE BINARY command this key format must be honored.

9.1 DES or AES Keys

DES and AES keys are stored in plain format. This means the key material starts with the byte at offset zero in the key file. The token determines the length of the key material by evaluating the algorithm reference defined in the Security Environment for a specific type of operation. Files holding symmetric keys must be smaller than files holding the smallest possible RSA private key.

9.2 RSA Keys

9.2.1 RSA Public Key

RSA public keys are stored in files as defined below. This format also applies for the response APDU of a GENERATE PUBLIC KEY PAIR command.

Number of bytes	Description
1	Key type (value: 04_{HEX})
1	Key length in byte/4 (e.g. 20_{HEX} for a 1024 bits key)
N	Modulus (e.g. N = 128 for a 1024 bits key)
4	Public exponent (3 or Fermat-4)

9.2.2 RSA Private Key

RSA private keys are stored in files as defined below.

Number of bytes	Description
1	Key type (value: 05_{HEX})
1	Key length in byte/4 (e.g. 20_{HEX} for a 1024 bits key)
N	Modulus (e.g. N = 128 for a 1024 bits key)
N	Private exponent

9.2.3 RSA Private Key in CRT Format

RSA private keys in CRT (Chinese Remainder Theorem) format are stored in files as defined below.

Number of bytes	Description
1	Key type (value: 06_{HEX})
1	Component length in byte/4 (e.g. 10_{HEX} for a 1024 bits key)
N	Prime-1 (P) (e.g. N = 64 for a 1024 bits key)
N	Prime-2 (Q)
N	Exponent-1 (DP)
N	Exponent-2 (DQ)
N	Coefficient (QP)

10 Key Management

The restrictions described in this section apply in **addition** to the standard access conditions defined for file operations.

10.1 RSA Private Key Import

During applications installation it is possible to define whether the token allows the import of RSA private keys or not. If private key import is forbidden then it is not possible to update files that can potentially be used in private key operations. Furthermore, it is ensured that private key files are no longer readable or modifiable upon key pair generation.
The size of a file in combination with its access conditions indicate if it can hold a private key or not. The minimum length of a valid private key is 130 bytes for the JCOP specific version of the application and 258 bytes for Java Card compatible version. Additionally, the access condition for signing or decryption must be other than NEVER. In contrary, public keys can only be used for encryption so that it is still possible to import public keys of any size as long as signing and decryption is forbidden on these files. Also, DES and AES keys are still importable since they do not need to be larger than 32 bytes anyway.

10.2 FIPS 140-2

For the FIPS 140-2 certified version of the BlueZ PKCS#15 application (as included in JCOP21id) additional restrictions apply. These restrictions are:

- The size of files holding symmetric (DES/AES) keys must be equals the size of the key. Consequently, the size of such key files can be in the range of 8-32 bytes.
- Read, update and erase operations on secret or private key files are only allowed if secure messaging is used. To enforce this it is not possible to access files smaller than 33 bytes which allow any of the crypto operations (sing, encrypt, decrypt) without secure messaging. The same is true for larger files that allow sign or decrypt operations since these files might hold RSA private keys. Files greater than 32 bytes which allow the encryption operation only, however, can be updated without secure messaging since they can only be used for public key operations.
- It is not possible to define the authentication object that protects the modification of the secure messaging keys. The access condition is automatically set to “SM”, which means secure messaging is required to update these keys. The corresponding byte in the personalization command is silently ignored.

Apart from the modified key management, the FIPS 140-2 version of the PKCS#15 applet does not support any algorithms involving the execution of the MD5 hash algorithm. Furthermore, the token automatically verifies the correctness of RSA key pairs upon key generation and enters halt mode if an error occurs. Also, secure messaging must be used whenever a PIN related command (CHANGE REFERENCE DATA, RESET RETRY COUNT, VERIFY) holding one ore more PIN values is sent.

11 APDU Command Details

11.1 Considerations

As specified in [2], in general all command APDUs can hold a payload of 255 data bytes and response APDUs can hold 256 bytes. Please be aware that in case of secure messaging this payload is decreased by the number of bytes needed for the secure messaging format defined in [3]. This limits the maximum payload for command APDUs to 239 bytes and for response APDUs to 231 bytes during secure messaging. As a result, it is, for instance, not possible to generate a 2048 bits signature while secure messaging is enabled.

11.2 Token Personalization Commands

The personalization of the PKCS#15 application is based on certain Open Platform functionality. For detailed information about Open Platform application life cycles, Open Platform secure messaging etc. see [9].
Upon installation of the BlueZ PKCS#15 application the application life cycle state is “LOADED”. In this state it only accepts a special set of commands, which allow to initially personalize the application via Open Platform secure messaging based on Card Manager keys. After a successful personalization the life cycle of the application is transitioned to “PERSONALIZED”. See the description of the command below for details of the personalization process.
This process is necessary to initially put keys and PINs onto the card in a secure manner. Typically one would open up another secure channel based on these keys later on to do further personalization of the token (e.g. generating keys) if necessary.

11.2.1 Open Platform INITIALIZE UPDATE Command

This command initiates the personalization process. Together with the Open Platform EXTERNAL AUTHENTICATE COMMAND it carries out mutual authentication between the card and the host application and sets up a Open Platform secure channel. This is based on symmetric keys of the Open Platform Card Manager. See [9] for details.
After successful processing of the command the card is authenticated and it expects the Open Platform EXTERNAL AUTHENTICATE COMMAND.

Command APDU:

Code	Value [HEX]	Remarks
CLA	80	Proprietary
INS	50	INITIALIZE UPDATE command
P1	00 or 01 to 7F	Key set version
P2	00	Key index
LC	08	Length of data
Data	XX	Host challenge
LE	00	

Response APDU:

As defined in [9].

11.2.2 Open Platform EXTERNAL AUTHENTICATE Command

This command authenticates the host and completes the setting up of the secure channel. A previous and successful INITIALIZE UPDATE COMMAND is necessary prior to processing this command. The security level “Encryption and MAC” is mandatory for token personalization. See [9] for details.

Command APDU:

Code	Value [HEX]	Remarks
CLA	84	Proprietary with secure messaging
INS	82	EXTERNAL AUTHENTICATE command
P1	03	Security level
P2	00	Parameter
LC	10	Length of data
Data	XX	Host cryptogram and MAC
LE	-	Not present

Response APDU:

Empty (status word 9000_{HEX} only).

11.2.3 INITIALIZE TOKEN Command

Once the secure channel is set up the token can be personalized. Using this command the initial secure messaging keys (for ISO secure messaging) and the initial PIN values are set on the token. Upon successful processing of this command the life cycle state of the application is set to “PERSONALIZED”.

Command APDU:

Code	Value [HEX]	Remarks
CLA	84	Proprietary with secure messaging
INS	F0	INITIALIZE TOKEN command
P1	00	
P2	00	
LC	51	Length of data
Data	XX	MACed and encrypted (Open Platform secure messaging): 16 byte - MAC key 16 byte - ENC key 16 byte – PIN 1 16 byte – PIN 2 16 byte – PIN 3 1 byte – reference number of the authentication object, which protects the secure messaging key files (value: 2,3,4,5 or 6 as defined in security attributes below)
LE	-	Not present

Response APDU:

Empty (status word 9000_{HEX} only).

11.3 File System Commands

The following commands represent a subset of the ISO/IEC standards [2]/[6]/[7]. The format is described for the case when they are sent in clear (no secure messaging). All commands can also be sent in the context of a secure channel using secure messaging. This affects the format of the commands as defined previously. In the case of secure messaging the token expects a class byte of $0C_{HEX}$.

11.3.1 PUT DATA Command

This command can be used to set the MODIFY access flag of a file to NEVER. The command can be performed only if the security status for MODIFY is satisfied.

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	DA	PUT DATA command

P1	01	Indicates proprietary application data
P2	00	
LC	02	Length of data
Data	XX	File identifier
LE	-	Not present

Response APDU:

Empty (status word 9000_{HEX} only).

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
6A	82	File not found
6A	86	Incorrect P1,P2
67	00	Wrong APDU data length

11.3.2 CREATE FILE Command

This command allows creating new transparent files under DF PKCS#15. The operation is bound to one of the authentication objects (as defined during token installation) and thus can only be performed upon successful authentication. The file identifier, the file size and the access conditions must be defined in the command. The requested size plus eight additional bytes for file information must be available in the pool of overall file system space (allocated during applet installation). It's not allowed to allocate files of zero length. After successful processing of the command the newly created file becomes the currently selected file. Duplicated file identifiers are not allowed. Upon allocation the file data is set to all bytes zero.

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	E0	CREATE FILE command
P1	00	Require file identifier in command data
P2	00	
LC	12	Length of data
Data	XX	File Control Parameters (FCP)
LE	-	Not present

The command data defines the file control parameters in accordance to [2] and [9]. The expected format is defined below.

Response APDU:

Empty (status word 9000_{HEX} only).

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
6A	80	Invalid file descriptor byte
6A	86	Incorrect P1,P2
67	00	Wrong APDU data length
6A	89	File already exists
6A	84	File system full

11.3.2.1 Command Data

Byte	Value[HEX]	Remarks
1	62	FCP tag
2	10	FCP length
3	80	File size tag
4	02	File size length
5	XX	File size high byte
6	XX	File size low byte
7	82	File descriptor byte tag
8	01	File descriptor byte length
9	01	Transparent EF
10	83	File identifier tag
11	02	File identifier length
12	XX	File identifier high byte
13	XX	File identifier low byte
14	86	Security attributes, proprietary format tag
15	03	Security attributes length
16	XX	Security attributes (byte 1) as defined below
17	XX	Security attributes (byte 2) as defined below
18	XX	Security attributes (byte 3) as defined below

Security Attributes

The security attributes of a file are encoded in three bytes whereas each nibble represents the access conditions for a certain type of operation as can bee seen below:

Byte 1		Byte 2		Byte 3	
READ	MODIFY	SIGN	ENCIPHER	DECIPHER	DELETE

The access conditions encoded in each of the nibbles are defined as follows:

Bit 4	Bit 3	Bit 2	Bit 1	Condition
-	0	0	0	ALWAYS
-	0	0	1	NEVER
X	0	1	0	PIN 1
X	0	1	1	PIN 2
X	1	0	0	PIN 3
-	1	0	1	AUTH
-	1	1	0	SM
-	1	1	1	RFU

Bit 4 is the "one-time" bit. If this bit is set the PIN will be invalidated after each operation for which it is required. This allows, for instance, enforcing PIN validation before each key usage for signature generation (as required in [8]).
It is to be noted that the access conditions AUTH and SM both relate to the secure channel (established via a MUTUAL AUTHENTICATE command). AUTH indicates a secure channel with no encryption (mutual authentication only) and SM indicates authentication with subsequent secure messaging. Consequently, if secure messaging is done (SM) the AUTH condition is satisfied as well but not vice versa.

11.3.3 DELETE FILE Command

This command allows to either delete a specific file, by passing its file identifier in the command data, or to delete the current selected file. The command can be performed only if the security status for DELETE is satisfied. After successful processing of the command DF PKCS#15 is selected (no EF is selected).

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	E4	DELETE FILE command
P1	00 or 02	(see Select File command in [2])
P2	00	
LC	none or 02	Length of data
Data	XX	File identifier
LE	-	

Response APDU:

Empty (status word 9000_{HEX} only).

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
67	00	Wrong APDU data length
69	86	No EF selected
6A	82	File not found

11.3.4 CHANGE REFERENCE DATA Command

This command allows to either update the value of a PIN, which is already validated, or to verify the PIN first and then update its value if the verification was successful. After successful processing of the command the PIN in question is always validated. The command will fail if the PIN is blocked (see RESET RETRY COUNTER command for PIN unblocking).

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	24	CHANGE REFERENCE DATA command
P1	00 or 01	00 – verify PIN first and then update 01 – update only
P2	XX	PIN number coded in bit 1 - bit 5
LC	10 or 20	Length of data
Data	XX	LC = 20 – old PIN value followed by new PIN value LC = 10 – new PIN value
LE	-	

Response APDU:

Empty (status word 9000_{HEX} only).

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
6A	86	Incorrect P1,P2
67	00	Wrong APDU data length
69	83	PIN blocked

11.3.5 RESET RETRY COUNTER Command

This command allows to either reset the retry counter of a PIN (explicit unblock) or to update a PIN value (implicit unblock). This functionality might be used by a security officer to unblock a user PIN or to set a new user PIN, therefore the command can only be processed if the security officer PIN (PIN 3) is already validated or the PIN value of the security officer PIN is sent in the command data. After the operation the security officer PIN is always validated and the target PIN is invalidated. The target PIN cannot be the security officer PIN since only user PINs can be unblocked.

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	2C	RESET RETRY COUNTER command
P1	00 – 03	See table of P1/LC combinations below
P2	01 - 03	PIN number coded in bit 1 - bit 5
LC	none or 10 or 20	Length of data
Data	XX	LC = 10 – security officer PIN (P1=1) or new PIN (P1=2) LC = 20 - security officer PIN followed by new PIN
LE	-	

Response APDU:

Empty (status word 9000_{HEX} only).

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
6A	86	Incorrect P1,P2
67	00	Wrong APDU data length

11.3.5.1 Valid P1/LC combinations

P1[HEX]	LC[HEX]	Description
00	20	The provided security officer PIN is verified. If it is correct the user PIN in question is updated (the new value is set).

01	10	The provided security officer PIN is verified. If it is correct the user PIN in question is unblocked (the retry counter is reset).
02	10	The user PIN in question is updated (the new value is set), if the security officer PIN is already validated.
03	None	The user PIN in question is unblocked (the retry counter is reset), if the security officer PIN is already validated.

11.3.6 READ BINARY Command

This command allows reading up to 256 bytes of data from the currently selected transparent EF. The command can be performed only if the security status for READ is satisfied and an EF is selected. Be aware that if the token operates in the context of a secure channel using secure messaging less than 256 can be read. In this case payload + padding + 17 byte must be less or equals to 256. This means the maximum payload is 231 bytes.

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	B0	READ BINARY command
P1	XX	Offset to the first byte to be read in data units from the beginning of the file.
P2	XX	
LC	-	
Data	-	
LE	XX	Number of bytes to be read (00 means 256)

Response APDU:

The data bytes from the file.

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
69	86	No EF selected
6A	86	Incorrect P1,P2 (out of bounds)

11.3.7 UPDATE BINARY Command

This command allows updating up to 255 bytes in the currently selected transparent EF with the data passed in the command. The command can be performed only if the

security status for MODIFY is satisfied and an EF is selected. Be aware that if the token operates in the context of a secure channel, less than 255 can be updated. In this case payload + padding + 13 byte must be less or equals to 255. This means the maximum payload is 239 bytes.
If the token (Java Card compatible version) is configured to prevent RSA private key import, it does not process this command on files, which are greater than 257 bytes and allow signature or decipher operations. Only files with these properties can be used for RSA private key operations. DES/AES keys or RSA public keys are still importable since they fit in smaller files.
If the token (BlueZ JCOP compatible version) is configured to prevent RSA private key import, it does not process this command on files, which are greater than 129 bytes and allow the sign or decipher operation. Only files with these properties can be used for RSA private key operations. (RSA public keys are still importable since they are only used for encipher operations).

If the target file is one of the secure messaging key files, secure messaging is mandatory. The file update is transactional.

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	D6	UPDATE BINARY command
P1	XX	Offset to the first byte to be updated in data units from the beginning of the file.
P2	XX	
LC	01-FF	Length of data
Data	XX	Data to be written
LE	-	

Response APDU:

Empty (status word 9000_{HEX} only).

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
69	86	No EF selected
6A	86	Incorrect P1,P2 (out of bounds)
67	00	Wrong APDU data length
69	86	Command not allowed (RSA private key not importable)

11.3.8 ERASE BINARY Command

This command sets (part of) the content of the currently selected transparent file to its logical erased state, sequentially, starting from a given offset. If command data is sent, it codes the offset of the first byte not to be erased. Otherwise the command erases up to the end of the file. The command can be performed only if the security status for MODIFY is satisfied and an EF is selected. The file modification is not transactional.

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	0E	ERASE BINARY command
P1	XX	Offset to the first byte to be erased in data units from the beginning of the file.
P2	XX	
LC	None or 02	Length of data
Data	XX	Empty or end offset
LE	-	

Response APDU:

Empty (status word 9000_{HEX} only).

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
69	86	No EF selected
6A	86	Incorrect P1,P2 (out of bounds)

11.3.9 SELECT FILE Command

This command allows selecting a transparent EF under the DF PKCS#15. If the file in question is found, it is set as selected and corresponding file control information is returned. After application reset (e.g. application selection) DF PKCS#15 is selected (no EF is selected).

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	A4	SELECT FILE command
P1	00 or 02	(see Select File command in [2])
P2	00	Return FCI

LC	02	Length of data
Data	XX	File identifier
LE	00	

Response APDU:

File control information (FCI) as defined below.

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
6A	82	File not found
6A	86	Incorrect P1,P2
67	00	Wrong APDU data length

11.3.9.1 EF FCI

Byte	Value[HEX]	Remarks
1	6F	FCI tag
2	13	FCI length
3	80	File size tag
4	02	File size length
5	XX	File size high byte
6	XX	File size low byte
7	82	File descriptor byte tag
8	01	File descriptor byte length
9	01	File descriptor byte (transparent EF)
10	83	File identifier tag
11	02	File identifier length
12	XX	File identifier high byte
13	XX	File identifier low byte
14	86	Security attributes, proprietary format
15	03	Security attributes length
16	XX	Security attributes (byte 1) (see CREATE FILE command)
17	XX	Security attributes (byte 2) (see CREATE FILE command)
18	XX	Security attributes (byte 3) (see CREATE FILE command)
19	85	Proprietary information
20	06	Proprietary information length
21	XX	Command processing counter high byte
22	XX	Command processing counter low byte
23	XX	Modification counter high byte
24	XX	Modification counter low byte

25	XX	Signature counter high byte
26	XX	Signature counter low byte
27	90	SW1
28	00	SW2

11.3.10 VERIFY Command

This command verifies the provided PIN value and either sets the PIN status to ‘validated’ or decrements the retry counter and sets the PIN status to ‘invalid’. The value passed in the command data must be padded with 00_{HEX} bytes to a length of 16 bytes. If the PIN in question is already blocked the token returns an error. If the verification fails the token returns an error code of $63CX_{HEX,}$ where X is the number of remaining tries before the PIN is blocked.
If the command is sent without command data and the PIN is not yet validated also $63CX_{HEX}$ is returned or (if the PIN is validated) 0x9000 is returned. This allows checking the state and the remaining retries without sending reference data (tying to validate).

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	20	VERIFY command
P1	00	
P2	XX	PIN number coded in bit 1 - bit 5
LC	10 or none	Length of data
Data	XX or none	Padded PIN value
LE	-	

Response APDU:

Empty (status word 9000_{HEX} only) or $63CX_{HEX}$ as described above.

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
63	CX	Verification failed, X remaining tries
6A	86	Incorrect P1,P2
67	00	Wrong APDU data length
69	83	PIN blocked

11.3.11 GET CHALLENGE Command

This command allows getting an eight byte true random challenge from the token to be used in authentication (via MUTUAL AUTHENTICATE command) or otherwise.

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	84	GET CHALLENGE command
P1	00	
P2	00	
LC	-	
Data	-	
LE	08	

Response APDU:

The eight bytes challenge.

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
6A	86	Incorrect P1,P2

11.4 Commands To Perform Security Operations

The commands outlined here provide access to the cryptographic capabilities of the token. For details see also [6].

11.4.1 MUTUAL AUTHENTICATE Command

This command, in combination with a previous GET CHALLENGE command, allows to do mutual authentication between the host and the card and to set up a secure channel. Since the challenge is only valid for one command, the mutual authentication must take place right after the GET CHALLENGE command. Upon successful processing of command the token has authenticated the host, session keys are derived and the token is ready to accept commands via the secure channel. If the secure channel was established with P1 equals zero then the token doesn't process plain commands anymore. All subsequent command must be encrypted and MACed. This state corresponds to the access condition "SM". If P1 was not equals zero the token continues to expect plain commands (no secure messaging). This state corresponds to the access condition "AUTH". The secure channel can be terminated by either setting up a new secure channel (out of the current secure channel) or by resetting the token (application selection). For details on secure messaging see section 12.

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	82	MUTUAL AUTHENTICATE command
P1	XX	zero means secure messaging required, plain otherwise
P2	00	
LC	10	Length of data
Data	XX	Eight bytes host challenge followed by eight bytes host cryptogram
LE	00	

Response APDU:

Eight bytes card cryptogram, which authenticates the card.

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or cryptogram invalid
69	84	Challenge invalid
67	00	Wrong APDU data length

11.4.2 MANAGE SECURITY ENVIRONMENT Command

This command is used to set up certain parameters on the card before security operations (e.g. sign, encipher etc.) are executed. The Security Environment is initially (after token reset) empty.

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	22	MANAGE SECURITY ENVIRONMENT command
P1	XX	C1 - SET target CRT F3 – RESTORE an empty security environment

P2	XX	Tag of target CRT (if SET command): B4 – cryptographic checksum template (CCT) B6 – digital signature template (DST) AA – hash template (HT) B8 – confidentiality template (CT) (for details see [6]) Zero if RESTORE command.
LC	00,0A,0B,14 or 1C	Length of data
Data	XX	CRT data (CRDOs) as defined below. Empty if RESTORE command.
LE	-	

Response APDU:

Empty (status word 9000_{HEX} only).

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect
6A	86	Incorrect P1, P2
6A	81	Function not supported (only SET/RESTORE command allowed)

11.4.2.1 Command Data

The token expects the following command data format. The hatched data is optional. This leads to a command data length of 10, 20 or 28 bytes. Which **C**ontrol **R**eference **D**ata **O**bjects (CRDO) are required for which **C**ontrol **R**eference **T**emplate (CRT) can be seen in the section "Required CRDOs in CRTs". CRDOs not required for a CRT but not optional in the command data can be set to zero since they will be ignored by the token.

Byte	Value[HEX]	Remarks
1	80	Algorithm identifier tag
2	01	Algorithm identifier length
3	XX	Algorithm identifier (as defined in Token Info file)
4	81	File reference tag
5	02	File reference length
6	XX	File identifier of key to be used in operation (high byte)
7	XX	File identifier of key to be used in operation (low byte)
8	84	Key reference tag
9	01	Key reference length
10	XX	Key reference (this value is ignored and can be zero)

11	87	Initialization Vector (IV) tag
12	08 or 10	IV length (IV length is 8 bytes for DES and 16 bytes for AES)
13	XX	IV bytes
…	…	
20/28	XX	

To prepare the token for RSA key pair generation the following command data format with the length of 11 bytes must be used:

Byte	Value[HEX]	Remarks
1	80	Algorithm identifier tag
2	01	Algorithm identifier length
3	XX	Algorithm identifier (as defined in Token Info file)
4	81	File reference tag
5	02	File reference length
6	XX	File identifier of file to receive public key (high byte)
7	XX	File identifier of file to receive public key (low byte)
8	81	Key reference tag
9	02	Key reference length
10	XX	File identifier of file to receive private key (high byte)
11	XX	File identifier of file to receive private key (low byte)

11.4.2.2 Required CRDOs in CRTs

CRT\CRDO	Algorithm reference	File reference 1	File reference 2	IV (optional - default is zero)	Key referece
CCT	X	X	-	X	-
DST	X	X	X(for key gen. only)	-	-
HT	X	-	-	-	-
CT	X	X	-	X	-

11.4.3 COMPUTE CRYPTOGRAPHIC CHECKSUM Command

This command initiates the computation of a cryptographic checksum. This typically means the card generates an 8/16 bytes MAC using DES/AES in CBC mode (see section Supported Cryptographic Algorithms). The command can be performed only if the security conditions for SIGN (of the key file referenced in the Security Environment) are satisfied. Before submitting this command the parameters for the desired operation (algorithm reference, key reference, IV) must be set in the Security Environment. The input data must be a multiple of the block length of the algorithm. Command chaining is not supported.

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	2A	PERFORM SECURITY OPERATION command
P1	8E	Output data is the cryptographic checksum
P2	80	Input is plain data
LC	XX	Length of data
Data	XX	Input data (multiple of block length)
LE	00	

Response APDU:

The cryptographic checksum (typically 8 bytes for DES and 16 bytes for AES).

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
67	00	Wrong data length (not multiple of block length)
69	88	Algorithm reference or key material invalid
6A	82	Key file not found

11.4.4 ENCIPHER Command

This command enciphers data using various algorithms (see section 8). The command can be performed only if the security conditions for ENCIPHER are satisfied. Before submitting this command the parameters for the desired operation (algorithm reference, key reference, IV) must be set in the Security Environment. For algorithms not providing automatic padding the input data must be a multiple of the block length of the algorithm. For RSA operations with PKCS#1 padding the input data must be less or equals modulus length - 11 bytes, since only one block can be returned in the response data. Command chaining is not supported. Enciphering using RSA private keys is not allowed.
If the input data has a length of 256 bytes, P2 holds the first byte of the input data. Also, if 256 bytes are to be returned, the response APDU does not hold the padding indicator byte. This allows processing 2048 bit operations with one command response pair.

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	2A	PERFORM SECURITY OPERATION command

P1	86	Output data is one padding indicator byte followed by the plain cryptogram. Padding indicator byte can be: 02_{HEX} – no padding 80_{HEX} – PKCS#1
P2	XX	80 – Input is plain data XX – first byte of input data if 2048 bit
LC	XX	Length of data
Data	XX	Input data (multiple of block length for algorithms without padding)
LE	00	

Response APDU:

Padding byte followed by plain cryptogram or cryptogram only if 256 bytes are to be returned.

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
67	00	Wrong data length
69	88	Algorithm reference or key material invalid
6A	82	Key file not found

11.4.5 DECIPHER Command

This command deciphers data using various algorithms (see section 8). The command can be performed only if the security conditions for DECIPHER are satisfied. Before submitting this command the parameters for the desired operation (algorithm reference, key reference, IV) must be set in the Security Environment. The input data must always be a multiple of the block length of the algorithm. For RSA operations with PKCS#1 padding the output data is less than the block length since padding bytes are automatically removed. The token returns the general error code $6F00_{HEX}$ if it encounters malformed padding. Command chaining is not supported. Deciphering using RSA public keys is not allowed.
If the input data has a length of 256 bytes, P2 holds the first byte of the input data. Also, the padding indicator byte is suppressed. The type of padding is implicitly known from the algorithm identifier in the security environment. This allows processing 2048 bit operations with one command response pair.

Command APDU:

Code	Value [HEX]	Remarks

CLA	00	Plain according to [2]
INS	2A	PERFORM SECURITY OPERATION command
P1	80	Output data is plain (padding is removed)
P2	XX	86 - Input data is one padding indicator byte followed by the plain cryptogram. Padding indicator byte can be: 02_{HEX} – no padding 80_{HEX} – PKCS#1 XX - first byte of input data if 2048 bit
LC	XX	Length of data
Data	XX	Input data (multiple of block length + padding indicator byte) (no padding indicator byte if 2048 bit)
LE	00	

Response APDU:

Decrypted data (padding is removed).

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
67	00	Wrong data length
69	88	Algorithm reference or key material invalid
6A	82	Key file not found

11.4.6 COMPUTE DIGITAL SIGNATURE Command

This command computes a digital signature for the given input data using a RSA private key. Depending on the algorithm used the input data can be a hash, digest info structure and hash, full block (PKCS#1 padded) or plain data. Command chaining is not supported. If the input data has a length of 256 bytes, P2 holds the first byte of the input data. This allows processing 2048 bit operations with one command response pair.

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	2A	PERFORM SECURITY OPERATION command
P1	9E	Output data is the digital signature

P2	XX	9A - Input data is the plain data to be integrated in the signing process XX - first byte of input data if 2048 bit
LC	XX	Length of data
Data	XX	Input data for digital signature
LE	00	

Response APDU:

The digital signature.

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
67	00	Wrong data length
69	88	Algorithm reference or key material invalid
6A	82	Key file not found

11.4.7 HASH Command

This command initiates the calculation of a hash code (SHA-1 or MD5). For this command the token supports command chaining (see [6]) so that an arbitrary amount of data can be feed into the hash engine. All blocks but the last must have a size that is a non-zero multiple of 64 byte. Upon receipt of the last block the token finalizes the hash and returns it in the response APDU. The command chaining is initiated by setting the chaining bit. The final command is indicated by sending a HASH command without the chaining bit set. Any other command than a HASH command terminates command chaining (the hash state is destroyed).

Command APDU:

Code	Value [HEX]	Remarks
CLA	00 or 10	Plain according to [2] Command chaining: 00 – for the last (or only command) 10 – for a command which is not the last command
INS	2A	PERFORM SECURITY OPERATION command
P1	90	Output data is the hash code (upon final or only command)
P2	80	Input data is the plain data to be hashed
LC	XX	Length of data
Data	XX	Input data for hash calculation

LE	00	

Response APDU:

The hash code (empty for intermediate commands during command chaining).

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
67	00	Wrong data length
69	88	Algorithm reference invalid

11.4.8 GENERATE PUBLIC KEY PAIR Command

This command initiates the generation and storing of a RSA public key pair in the token. The token supports key generation with a public key value of either 3 or 65537 (Fermat-4). The command requires two files to be present (previously set in the DST CRT of the Security Environment), which can receive the resulting public key and private key values. The size of the public key file defines indicates the desired key length. The token always generates a private key in the CRT format. This means the target file for the private key must be of appropriate size (e.g. 322 bytes for a 1024 bit key). The command can be performed only if the security conditions for MODIFY (of both files) are satisfied. If the LE byte of the command APDU is zero the access conditions for the two files are modified during key generation. The MODIFY access condition of both files is set to NEVER and the READ access condition of the private key file is also set to NEVER. If the token does not allow importing RSA key material this is always the case (independent of LE).
Upon successful key generation the token returns the public key in the response APDU. If the key length is greater than 1984 bit only the modulus is returned. The public exponent is implicitly known anyway. This indicates that the key material is successfully stored. The key generation process is not transactional. For a detailed description of the key format see section 9.

Command APDU:

Code	Value [HEX]	Remarks
CLA	00	Plain according to [2]
INS	46	GENERATE PUBLIC KEY PAIR command
P1	00	Generate and store PK pair
P2	00	
LC	-	
Data	-	
LE	XX	Zero or length of public key value

Response APDU:

Public key value encoded as defined in section 9 (modulus only if key length > 1984 bit).

Status conditions:

$SW1_{HEX}$	$SW2_{HEX}$	Remarks
69	82	Secure messaging incorrect or PIN not verified
6A	86	Incorrect P1,P2
69	88	Algorithm reference or key file invalid
6A	82	Key file not found

12 Secure Messaging

Secure Messaging defines a cryptographic protocol that allows setting up a secure channel, which ensures integrity and confidentiality of the APDU communication between the smart card and the reader device. The cryptographic operations for secure messaging like mutual authentication, session key generation, data encryption and MAC generation are as defined in Open Platform ([9]) since the ISO specifications leave this open. The secure messaging format, however, is as defined in [3]. This section provides an overview of the secure messaging protocol implemented by the BlueZ PKCS#15 application. For further details please contact the BlueZ Secure Systems team.

12.1 Mutual Authentication

To do mutual authentication between a host and the card (PKCS#15 application) two command response pairs must be exchanged. At first the host sends the GET CHALLENGE command to the card to get an 8 byte true random value known as the "card challenge" (Crnd). The host then also generates an 8 byte random value known as the "host challenge" (Hrnd) and calculates the "host cryptogram" (Hcg) as follows:

$Hcg = DES3MAC(Key_{SES\_ENC} , Crnd|Hrnd|0x8000000000000000)$

where

$Key_{SES\_ENC} = DES3ENC(K_{ENC}, Cl \mid Hf) \mid DES3ENC(K_{ENC}, Cf \mid Hl)$

and

Hf/Hl : host challenge (first/last 4 bytes)
Cf/Cl : card challenge (first/last 4 bytes)
K_{ENC} : static encryption and authentication key (ENC key)

Key_{SES_ENC} : encryption and authentication session key
DES3MAC : triple DES CBC MACing (as described in next chapter) with zero ICV
DES3ENC : triple DES CBC Encryption (as described in next chapter) with zero ICV

The host then sends the host challenge along with host cryptogram to the card using the MUTUAL AUTHENTICATE command. The card verifies the host cryptogram and calculates the "card cryptogram" (Ccg) as follows:

$$Ccg = DES3MAC(Key_{SES\_ENC}, Hrnd|Crnd|0x8000000000000000)$$

The card cryptogram is returned as response to the MUTUAL AUTHENTICATE command so that the host can verify it.
As part of this mutual authentication process both, the card and the host, calculate the MAC session key (Key_{SES_MAC}) as follows:

$$Key_{SES\_MAC} = DES3ENC(K_{MAC}, Cl \mid Hf) \mid DES3ENC(K_{MAC}, Cf \mid Hl)$$

where

K_{MAC} : static encryption and authentication key (MAC key)

The MUTUAL AUTHENTICATE command indicates whether the secure channel shall be secured (encrypted and MACed) or plain. If it is to be secured all subsequent APDU are secured as described in the following using the two session keys established during mutual authentication. Therefore, in the following it is assumed that the secure channel setup already took place and the session keys (Key_{SES_ENC} and Key_{SES_MAC}) are available.

12.2 Cryptographic Algorithms

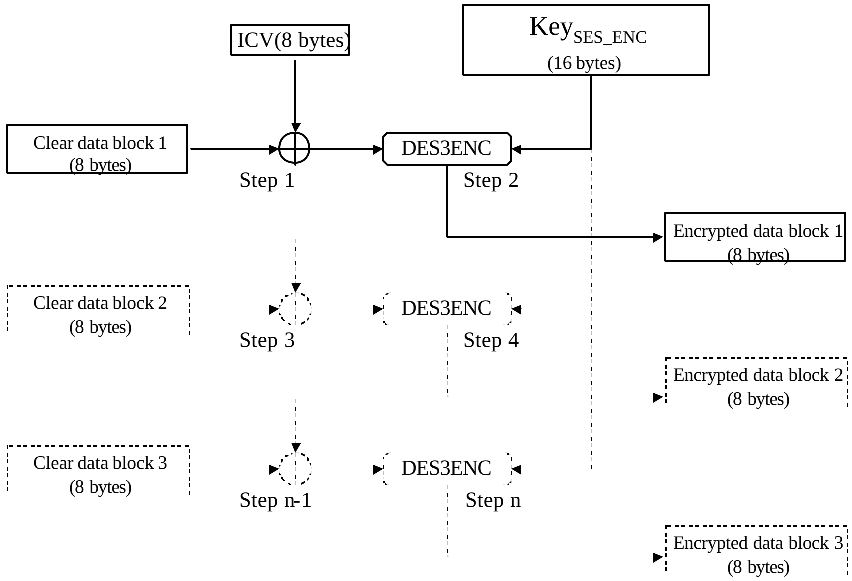
All cryptographic operations described are based on the triple DES algorithm using a double length (16 byte) key.

12.2.1 Padding Algorithm

Padding is used for MAC generation as well as for data encryption, whereas in case of MAC generation the padding bytes are not to be transmitted. Padding shall consist of one mandatory byte valued to 80_{HEX} followed, if needed, by 0 to k-1 bytes set to 00_{HEX}, until the respective data block is filled up to k bytes (k is a multiple of 8 bytes). This represents the padding algorithm defined in [2] and [10].

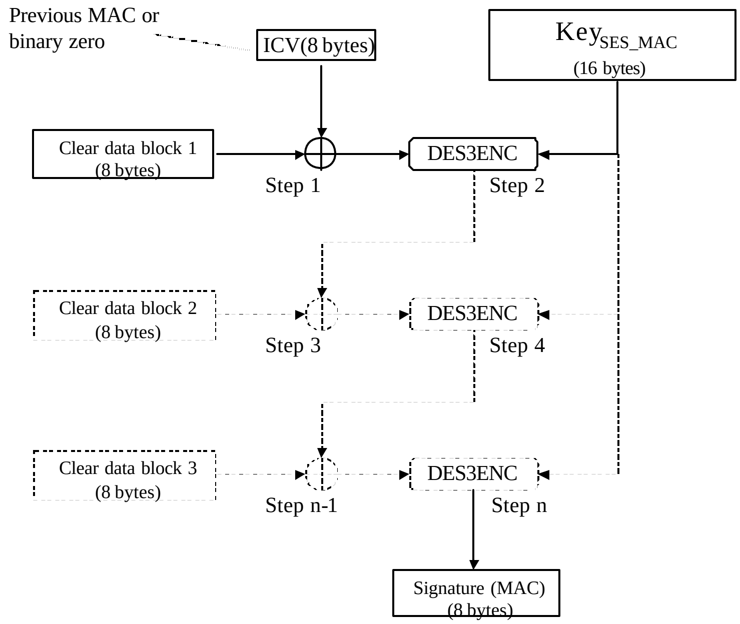
12.2.2 Data Encryption

The encryption of APDU command and response data is done using triple DES encryption in CBC mode as outlined below. The Initial Chaining Vector (ICV) is always 8 bytes of binary zero (00_{HEX}).



12.2.3 Authentication Cryptogram (MAC) Generation

To sign APDUs a triple DES based MAC algorithm operating in CBC mode is used. To generate a MAC for a command APDU the MAC of the previous command APDU is used as the ICV. To generate a MAC for a response APDU the MAC of the previous response APDU is used as the ICV. This chaining process does not span multiple secure channel sessions. For the first MAC generated for a command APDU in the context of a secure channel session the host cryptogram is used as the ICV. For the first MAC generated for a response APDU in the context of a secure channel session the card cryptogram is used as the ICV.



12.3 Secure Messaging Format

The format of the secure messaging used to maintain the secure channel is defined in [3]. Nevertheless, the exact impact of secure messaging on the structures of APDU messages is again described below.
In the context of a Secure Channel all command and response APDUs must be signed (status words and command headers need to be protected). Furthermore, in those cases where they hold command or response data, this data must be encrypted.

12.3.1 Abbreviations

CC cryptographic checksum (triple DES MAC)
CG cryptogram (padded and encrypted command or response data)
CLA^* class byte indicating secure messaging (value $0C_{HEX}$)
T_{CC} tag indicating a CC (value $8E_{HEX}$)
L_{CC} length of a CC (value 08_{HEX})
CH command header (CLA^* INS P1 P2)
PB padding bytes
T_{SW} tag indicating a status word (value 99_{HEX})
L_{SW} length of a status word (value 02_{HEX})
T_{LE} tag indicating a L_e field (value 96_{HEX})
L_{LE} length of a L_e field (value 01_{HEX})
$T_{PI\ CG}$ tag indicating a PI followed by a CG (value 86_{HEX})
$L_{PI\ CG}$ length of a PI followed by a CG

PI padding indicator byte (value 01_{HEX})

12.3.2 Case 1 APDU

The unsecured command-response pair is as follows.

Command header	Command body
CLA INS P1 P2	Empty

Response body	Response trailer
Empty	SW1 SW2

The secured command APDU is as follows.

Command header	Command body
CLA* INS P1 P2	New L_c field (value $0A_{HEX}$) || New data field (10 bytes) || New L_e field (value 00_{HEX})

New data field = One data object = T_{cc} || L_{cc} || CC

Data covered by CC = One block = CH || PB

The secured response APDU is as follows.

Response body	Response trailer
New data field	SW1 SW2

New data field = Two data objects = T_{sw} || L_{sw} || SW1 SW2 ||
T_{cc} || L_{cc} || CC

Data covered by CC = One block = T_{sw} || L_{sw} || SW1 SW2 || PB

12.3.3 Case 2 APDU

The unsecured command-response pair is as follows.

Command header	Command body
CLA INS P1 P2	L_e field

Response body	Response trailer
Data field	SW1 SW2

The secured command APDU is as follows.

Command header	Command body
CLA* INS P1 P2	New L_c field (value $0D_{HEX}$) || New data field (13 bytes) || New L_e field (value 00_{HEX})

New data field = Two data objects = T_{LE} || L_{LE} || LE ||
T_{cc} || L_{cc} || CC

Data covered by CC = Two blocks = CH || PB || T_{LE} || L_{LE} || LE || PB

Note: LE is the value of the L_e field in the unsecured command

The secured response APDU is as follows.

Response body	Response trailer
New data field	SW1 SW2

New data field = Three data objects = $T_{PI\ CG}$ || $L_{PI\ CG}$ || PI || CG ||
T_{sw} || L_{sw} || SW1 SW2 ||
T_{cc} || L_{cc} || CC

Data covered by CC = One or more blocks = $T_{PI\ CG}$ || $L_{PI\ CG}$ || PI || CG ||
T_{sw} || L_{sw} || SW1 SW2 || PB

12.3.4 Case 3 APDU

The unsecured command-response pair is as follows.

Command header	Command body
CLA INS P1 P2	L_c field || Data field

Response body	Response trailer
Empty	SW1 SW2

The secured command APDU is as follows.

Command header	Command body
CLA* INS P1 P2	New L_c field || New data field || New L_e field (value 00_{HEX})

New data field = Two data objects = $T_{PI\ CG}$ || $L_{PI\ CG}$ || PI || CG ||

T_{cc} || L_{cc} || CC

Data covered by CC = Two or more blocks = CH || PB ||
$T_{PI\ CG}$ || $L_{PI\ CG}$ || PI || CG || PB

The secured response APDU is as follows.

Response body	Response trailer
New data field	SW1 SW2

New data field = Two data objects = T_{sw} || L_{sw} || SW1 SW2 ||
T_{cc} || L_{cc} || CC

Data covered by CC = One block = T_{sw} || L_{sw} || SW1 SW2 || PB

12.3.5 Case 4 APDU

The unsecured command-response pair is as follows.

Command header	Command body
CLA INS P1 P2	L_c field || Data field || L_e field

Response body	Response trailer
Data field	SW1 SW2

The secured command APDU is as follows.

Command header	Command body
CLA* INS P1 P2	New L_c field || New data field || New L_e field (value 00_{HEX})

New data field = Three data objects = $T_{PI\ CG}$ || $L_{PI\ CG}$ || PI || CG ||
T_{LE} || L_{LE} || LE ||
T_{cc} || L_{cc} || CC

Data covered by CC = Two or more blocks = CH || PB ||
$T_{PI\ CG}$ || $L_{PI\ CG}$ || PI || CG ||
T_{LE} || L_{LE} || LE || PB

The secured response APDU is as follows.

Response body	Response trailer
New data field	SW1 SW2

New data field = Three data objects = $T_{PI\ CG}$ || $L_{PI\ CG}$ || PI || CG ||

T_{sw} || L_{sw} || SW1 SW2 ||
T_{cc} || L_{cc} || CC

Data covered by CC = One or more blocks = $T_{PI\ CG}$ || $L_{PI\ CG}$ || PI || CG ||
T_{sw} || L_{sw} || SW1 SW2 || PB

13 References

[1] PKCS#15 v1.1: Cryptographic Token Information Syntax Standard, RSA Laboratories, June 6 ,2000

[2] ISO/IEC 7816-4, Part 4: Interindustry commands for interchange, First edition 1995-09-01

[3] ISO/IEC 7816-4, Part 4: Interindustry commands for interchange, First edition 1995-09-01, Amendment 1: Impact of secure messaging on the structures of APDU messages

[4] ISO/IEC 7816-5, Part 5: Numbering system and registration procedure for application identifiers, First edition 1994-06-15

[5] ISO/IEC 7816-6, Part 6: Interindustry data elements, First edition 1996-05-15

[6] ISO/IEC 7816-8, Part 8: Security related interindustry commands, First edition 1999-10-01

[7] ISO/IEC 7816-9, Part 9: Additional interindustry commands and security attributes, First edition 2000-09-01

[8] PKCS#15: Conformance Profile Specification, RSA Laboratories, August 1, 2000

[9] Visa Open Platform, Card Specification, Version 2.0.1

[10] ISO/IEC 9797-1 (1999): Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher.