TS 101 206-3 V1.3.2 (1998-12)

Technical Specification

Identification card systems; Telecommunications IC cards and terminals; Part 3: Application independent card requirements

Reference

RTS/PTS-00013 (b60r0j1f.PDF)

Keywords

card

Contents

Intellectual Property Rights

Foreword

This Technical Specification (TS) has been produced by ETSI Project Pay Terminal and Systems (PTS).

Version 1.1.1 of ETSI TS 101 206-3 is an equivalent of the original handover version to CEN for becoming EN 726-3, but is not published by ETSI.

Version 1.2.1 of ETSI TS 101 206-3, 4 and 7 represent an update of the documents handed over to CEN for becoming EN 726-3, 4, and 7. PTS has used version 1.2.1 rather than the original handover version to CEN (version 1.1.1) for producing the conformance testing specifications for CEN EN 726-3, 4 and 7 (CEN prEN13343/4/5). Version 1.2.1 is published by ETSI and has since been handed over to CEN for the update process of EN 726, to give a consistent set of standards and conformance testing specifications in CEN. Once published by CEN as an EN, ETSI will withdraw these TSs.

Versions 1.3.x onwards of ETSI TSs had been produced by PTS for working use within ETSI. These are not a copy of any CEN published EN. These ETSI TSs incorporate change requests accepted by PTS at the time of publication. The intention is that these documents can be used for the update of EN 726 within CEN at some time in the future. These ETSI TSs may not match the CEN published conformance testing specifications for EN 726 (prEN13343/4/5). Update of the CEN conformance testing specifications will need to be considered when these ETSI TSs (versions 1.3.x onwards) are handed over to CEN.

History of EN 726

CEN EN 726 was prepared by ETSI STC TE9 (at present PTS), adopted by CEN/TC 224 and submitted to the CEN formal vote. EN 726 consists of seven parts covering Identification card systems; Telecommunications IC cards and terminals; as identified below:

Part 1:	"System overview";
Part 2:	"Security framework";
Part 3:	**"Application independent card requirements";**
Part 4:	"Application independent card related terminal requirements";
Part 5:	"Payment methods";
Part 6:	"Telecommunication features";
Part 7:	"Security module".

ETSI deliverables on EN 726 family

TS 101 206-3	**"Identification card systems; Telecommunications IC cards and terminals; Part 3: Application independent card requirements".**
TS 101 206-4	"Identification card systems; Telecommunications IC cards and terminals; Part 4: Application independent card related terminal requirements".
TS 101 206-7	"Identification card systems; Telecommunications IC cards and terminals; Part 7: Security module".

ETSI deliverables on EN 726 conformance testing family

TS 101 203-1/2/3	"Identification card systems; Telecommunications IC cards and terminals; Test methods and conformance testing for EN 726-3". (prEN 13343)
TS 101 204-1/2/3	"Identification card systems; Telecommunications IC cards and terminals; Test methods and conformance testing for EN 726-4". (prEN 13344)
TS 101 207-1/2/3	"Identification card systems; Telecommunications IC cards and terminals; Test methods and conformance testing for EN 726-7". (prEN 13345)

1 Scope

The present document specifies the application-independent characteristics of multi-application Integrated Circuit (IC) cards and plug-in modules for telecommunication applications in order to ensure interoperability for telecommunication cards with the various systems and terminals. Mono-application cards are considered to be a subset of multi-application cards. All common characteristics, necessary for the interactions between the card and the external world are defined.

The present document does not preclude cards from other sectors from containing telecommunication application(s) based on the present document.

The application-specific characteristics are not defined in the present document. They are defined and described in the relevant application requirements.

The present document does not specify any internal technical implementation. It describes:

a) the requirements for the physical characteristics of the card, the electronic signals and the transmission protocols;

b) the application-independent logical model which should be used as a basis for the design of the logical structure of, optionally, several applications in the card;

c) the security facilities concerning the access to the different parts within the card and the possible interactions between these parts. Also the description of security functions which should be needed generally by the various applications. They should be available as a common set;

d) the description of the application-independent functions between card and external world, should be used as a standardized common set for all basic functions used in international applications;

e) the mapping of these application messages (commands and responses) under standardized protocols;

f) the contents of the Master File (MF);

g) the interoperability of IC cards;

h) the overall security aspects for card-manufacturers, application providers and card-issuers.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.
- A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[1] ENV 1375-1: "Identification card systems - Intersector integrated circuit(s) card additional formats - Part 1: ID-000 card size and physical characteristics".

[2] ENV 1375-2: "Identification card systems - Intersector integrated circuit(s) card additional formats - Part 2: ID-00 card size and physical characteristics".

[3] Void.

[4] EN 27811-1 (1989): "Identification cards - Recording technique - Part 1: Embossing".

[5] EN 27811-2 (1989): "Identification cards - Recording technique - Part 2: Magnetic stripe".

[6] Void.

[7] EN 27816-1 (1989): "Identification cards - Integrated circuit(s) with cards contacts - Part 1: Physical characteristics".

[8] EN 27816-2 (1989): "Identification cards - Integrated circuit(s) cards with contacts - Part 2: Dimensions and location of the contacts".

[9] EN 27816-3 (1992): "Identification cards - Integrated circuit(s) cards with contacts - Part 3: Electronic signals and transmission protocols".

[10] EN 27816-3 (1992/A1:1993): "Identification cards - Integrated circuit(s) cards with contacts - Part 3: Electronic signals and transmission protocols - Amendment 1: Protocol type T=1, asynchronous half duplex block transmission protocol".

[11] I-ETS 300 045-1: "European digital cellular telecommunications system (phase 1): Subscriber Identity Module - Module Equipment (SIM - ME) interface specification; Part 1: Generic (GSM 11.11)".

[12] ISO 639 (1988): "Code for the representation of names of languages".

[13] ISO/IEC 646 (1991): "Information processing - ISO 7-bit coded character set for information interchange".

[14] ISO 3166 (1988): "Codes for the representation of names of countries".

[15] ISO/IEC 7816-4: "Information technology - Identification cards - Integrated circuit(s) cards with contacts - Part 4: Interindustry commands for interchange".

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

[17] ISO 8859-1 (1987): "Information processing - 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet N°1".

[18] Void.

[19] Void.

[20] Void.

[21] Void.

[22] Void.

[23] Void.

[24] Void.

[25] Void.

[26] Void.

[27] Void.

[28] Void.

[29] Void.

[30] CCITT Recommendation E.118 (1988): "Automated international telephone credit card system".

[31] CCITT Recommendation T.50 (1988): "International alphabet n°5".

[32] TS 101 206-4: "Identification card systems; Telecommunications IC cards and terminals; Part 4: Application independent card related terminal requirements".

3 Definitions, abbreviations, symbols and notations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

Access Conditions (AC): a set of security attributes associated to a file.

allocable memory: portion of memory contained in a file but not presently allocated.

application: an application consists of a set of security mechanisms, files, data, protocols (excluding transmission protocols) which are located and used in the IC card and outside the IC card (external application).

application session: a link between the card application part and the external world application part of the same application.

Application Specific Command set (ASC): to a Dedicated File (DF) can be associated an optional an Application Specific Command set and/or an application specific program (ASC-set). This means that when selecting this application, the general command set is extended or modified by this specific command set. The ASC is valid for the whole subtree of this application unless there are other ASCs defined at the lower levels of this application.

Application Provider (AP): the entity which is responsible for the application after its allocation. One AP may have several applications in one card. The files allocated in the card corresponding to one application are called a card-application. There may exist several applications on a given card from the same Application Provider (AP).

card: a multi-application card can be considered as a set of files, some of them shared by the different application providers and/or the card issuer, other files owned exclusively by the different application providers or the card issuer. Files can, e.g. be read, written or executed.

card session: a link between the card and the external world starting with the Answer To Reset (ATR) and ending with a subsequent reset or a de-activation of the card.

current directory: the latest directory (Master File (MF) or Dedicated File (DF)) selected in the card.

current EF: the latest Elementary File (EF) selected in the card.

current file: the latest file (MF, DF or EF) selected in the card.

Dedicated File (DF): a file containing Access Conditions (AC) and allocable memory. It may be the parent of Elementary Files (EF) and/or Dedicated Files (DF).

directory: general name for MF or DF.

Elementary File (EF): a file containing Access Conditions (AC), data or program. It can not be the parent of another file.

$\mathbf{EF_{CHV}}$ is an elementary file containing the Card Holder Verification (CHV) information.

$\mathbf{EF_{DIR}}$ is an elementary file at the MF or at DF level, which contains a list of all, or part of, available applications in the card (see also ISO 7816-5) [16].

$\mathbf{EF_{ID}}$ is an elementary file at the MF level, containing the identification number of the card.

$\mathbf{EF_{IC}}$ is an elementary file at the MF level, containing general information concerning the Integrated Circuit (IC).

$\mathbf{EF_{ICC}}$ is an elementary file at the MF level, containing general information concerning the ICC.

$\mathbf{EF_{KEY\_OP}}$ is an elementary file containing operational keys.

$\mathbf{EF_{KEY\_MAN}}$ is an elementary file containing management keys.

$\mathbf{EF_{LANG}}$ is an elementary file at the MF level, containing the language preferences.

EF$_{NAME}$ is an elementary file at the MF level, containing the card holder name.

file IDentifier (ID): each file (MF, DF, EF) has a file identifier consisting of 2 bytes.

file qualifier: first byte of the file identifier.

keyfile version: indicates the absolute version number of the keyfile (coded in BCD).

Master File (MF): the mandatory unique file representing the root of the file structure and containing Access Conditions (AC) and allocable memory. It may be the parent of elementary files and/or dedicated files.

operating system: that which is required to manage the logical resources of a system, including process scheduling and file management.

padding: one or more bits appended to a message in order to cause the message to contain the required number of bits or bytes.

path: concatenation of file IDentifiers (ID) without delimitation.

pattern: is a string of bytes.

record: a string of bytes handled as a whole by the card and referenced by a record number or a record pointer.

record number: is sequential and unique within an EF. It is managed by the card.

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AC	Access Condition
ALW	ALWays
AP	Application Provider
APDU	Application Protocol Data Unit
ASC	Application Specific Command set
ATR	Answer To Reset
AUT	AUThenticated
BCD	Binary Coded Decimal
CEN	Comité Européen de Normalisation
CHV	Card Holder Verification
DECT	Digital Enhanced Cordless Telecommunications
DF	Dedicated File
EF	Elementary File
etu	elementary time unit
IC	Integrated Circuit
ID	IDentifier
MF	Master File
MII	Major Industry Identifier
NEV	NEVer
PIN	Personal Identification Number
PRO	Protected
PTS	Protocol Type Select (response to the ATR)
RFU	Reserved for Future Use
STC	ETSI Technical Sub-Committee
TC	Technical Committee
TE9	Terminal Equipment 9 (Technical Sub-Committee, STC, of ETSI)

3.3 Symbols

For the purposes of the present document the following symbols apply:

nAs	nano Ampere per second
ns	nano second
mA	milli Ampere
V_{CC}	Supply Voltage
V_{PP}	Programming Voltage

3.4 Notations

For the purposes of the present document the following notations apply:

"0" to "9" and "A" to "F": the sixteen hexadecimal digits.

4 Physical characteristics of the card

The physical characteristics for ID-1 cards shall be in accordance with EN 27816-1 [7] and EN 27816-2 [8].

Nevertheless, EN 726 is not limited to ID-1 size card but shall also cover other card formats as defined in ENV 1375-1 [1] and ENV 1375-2 [2].

The following additional requirements shall be applied in order to ensure proper operation in portable battery and line powered operated equipment as well as in stationary equipment.

4.1 Layout

The position of the contacts for cards used in Europe shall be as follows:

a) if no embossing and no magnetic stripe, contacts on any side;

b) if embossing, contacts on same side;

c) if magnetic stripe, contacts on other side;

d) if embossing & magnetic stripe, contacts on embossing side.

The IDentification number and the card sequence number (if it exists) as defined in EF_{ID} (see clause 10) may be present on the outside of the card (if embossed, then in accordance with EN 27811 [4] and [5], ISO 639 [12]).

4.2 Temperature range for card operation

The temperature range for full operational use shall be between -25°C and +65°C with occasional peaks of up to +70°C.

"Occasional" means not more than 4 hours each time and not over 100 times during the life time of the card.

For multi-application cards, which may also be used in portable battery operated equipment, the temperature range for operational use shall be between -25°C and +70°C with occasional peaks of up to +85°C.

"Occasional" means not more than 4 hours each time and not over 100 times during the life time of the card.

5 Electronic signals and transmission protocols

Electronic signals and transmission protocols shall be in accordance with EN 27816-3 [9] and [10]. The card shall support at least one of the protocols defined in that standard.

The following additional requirements shall be applied in order to have simplified terminals and to ensure a proper operation in mobile equipment, including portable battery operated, line powered equipment as well as in stationary equipment.

5.1 Supply voltage

The card shall be operated with a supply voltage V_{CC} of 5 V ± 10 %.

5.2 Supply current

The current consumption shall not exceed 20 mA at any frequency accepted by the card. If the card is expected to be used in mobile equipment including portable battery operated as well as line powered equipment, the current consumption shall not exceed 10 mA at any frequency.

Due to technology, spikes in the supply current can occur, the amplitude of which can be several times the average current. Under no circumstances, during operation, shall the card draw spike charges of over 40 nAs with no more than 400 ns duration and an amplitude of at most 200 mA.

5.3 Programming voltage

A programming voltage different from V_{CC} shall be generated internally.

NOTE: For the European telecommunication cards, the terminal is required to supply the V_{PP} contact with the same voltage as the V_{CC} contact (see TS 101 206-4 [32]).

Special conditions may apply to mobile equipment (see relevant specifications I-ETS 300 045-1 [11] and DECT specifications).

5.4 Duty cycle

Duty cycle for asynchronous operation shall be between 40 % and 60 % of the period during stable operation.

5.5 Guard time

For the transmission between the card and the terminal there are two extra guard times with regard to the time duration of the character. The character duration shall be fixed to 12 elementary time units (etu). The extra guard time for the transmission from the terminal to the card shall be fixed by TC1, parameter N in the Answer To Reset (ATR), which shall have a value of 255 (except for T = 0), 0 or 4.

5.6 Low consumption mode

Cards may have a low power consumption mode, especially for mobile equipment, indicated in the "clockstop" byte in EF_{ICC}.

6 Logical model for IC cards

The logical organization of data in a card, which defines the memory management of the card is shown in figure 1.

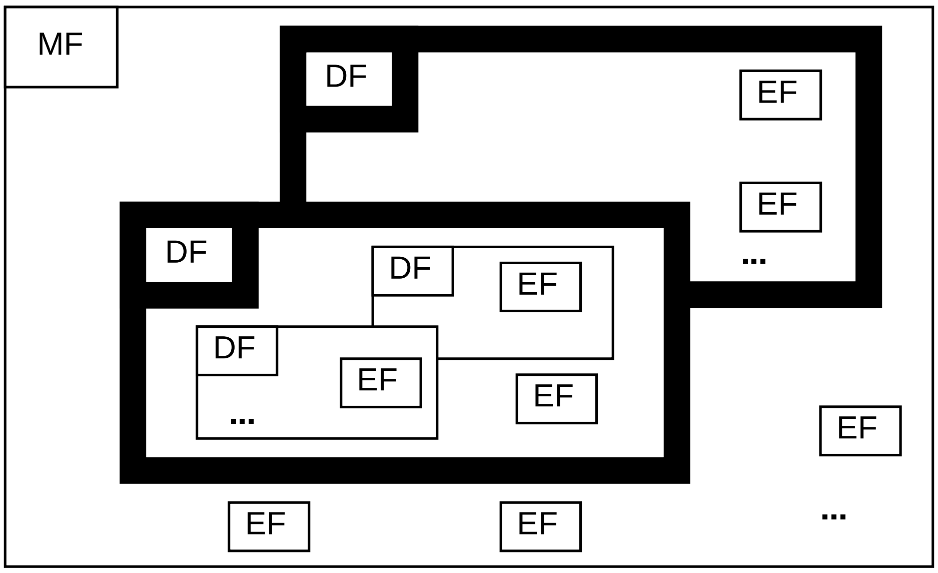


Figure 1: Data structure model for an IC card

6.1 File identifier

See ISO/IEC 7816-4 [15].

The file ID shall be chosen at the creation of the concerned file and shall be different for two files under the same parent. It is up to the operating system to ensure this requirement.

6.2 Elementary File (EF) structures

Based on ISO/IEC 7816-4 [15], the following different Elementary File (EF) structures are defined.

6.2.1 Transparent EF

An EF with a transparent structure consists of a sequence of bytes. A sequence of bytes to be read, written or updated are referenced by a relative address (offset) and a length indication (in bytes). The first byte of an EF with transparent structure has the relative address "00". The total data length of the EF is indicated in the header of the EF.

6.2.2 EF containing programs

If a transparent EF containing a program is created, the card issuer shall take all necessary steps to ensure that there is no possibility of interference between applications

For the set of commands, two possibilities appear:

a) the general set of commands is sufficient and there is no need to associate this DF with an ASC-set;

b) the general set of commands is not sufficient and there is a need to associate an ASC-set with this DF. This ASC-set shall be defined and agreed upon.

Programs contained in an EF are more related to the application with regard to some user-specific characteristics.

EXAMPLE 1: In some applications such a program could be a calculation of the card holders credit limit based on specific user characteristics.

Programs controlled by the specific ASC-set are more related to the application only.

EXAMPLE 2: A specific cryptographic algorithm used to authenticate the application.

6.2.3 EF with linear fixed structure

An EF of this structure consists of a sequence of records with fixed length. The first record in this EF is defined as record #1. The following are indicated in the header of this structure:

a) the total data length;

b) number of records created;

c) length of record.

There are three methods to access records within an EF of this type:

a) using the record number;

b) when positioned on the current record (known by the operating system), it shall be possible to perform an action on the current, the next (except for last record), the previous (except for the first record), the first or the last record existing in the EF;

c) by pattern seek starting from the beginning forward, from the end backward, from the next record forward and from the previous record backward.

It is not possible to create more than 254 records in one file.

6.2.4 EF with linear variable structure

An EF of this structure consists of a sequence of records with variable length. The first record in this EF is defined as record #1.

The following items are indicated in the header of this structure:

a) the total data length;

b) the number of records created.

Each record has it is own length indication, which cannot be changed after creation of the record.

The access to this type of EF is the same as for EFs with a linear fixed structure.

It is not possible to create more than 254 records in one file.

6.2.5 Cyclic EF

An EF of this structure consists of a sequence of records with identical length, organized as a ring.

In each file of cyclic structure the card maintains a reference to the last written record. In an EF of cyclic structure, the record number one (#1) is the last written record. The oldest written record, has the highest record number.

For writing operations, the only way of addressing a record is PREVIOUS. Only the record with the highest record number can be overwritten.

For increase and decrease operations the value to be modified is in the current record number 1, and the result of the computation shall be written in the oldest record. After an increase or decrease operation, the updated record shall become the current record number 1.

For reading operations, the ways of addressing a record are FIRST, LAST, NEXT, PREVIOUS, CURRENT or record number. FIRST, LAST, NEXT, PREVIOUS and CURRENT are used as described in ISO/IEC 7816-4 [15] subclause 5.1.4.1.

After the selection of a cyclic EF, the record pointer is automatically set on the last written record.

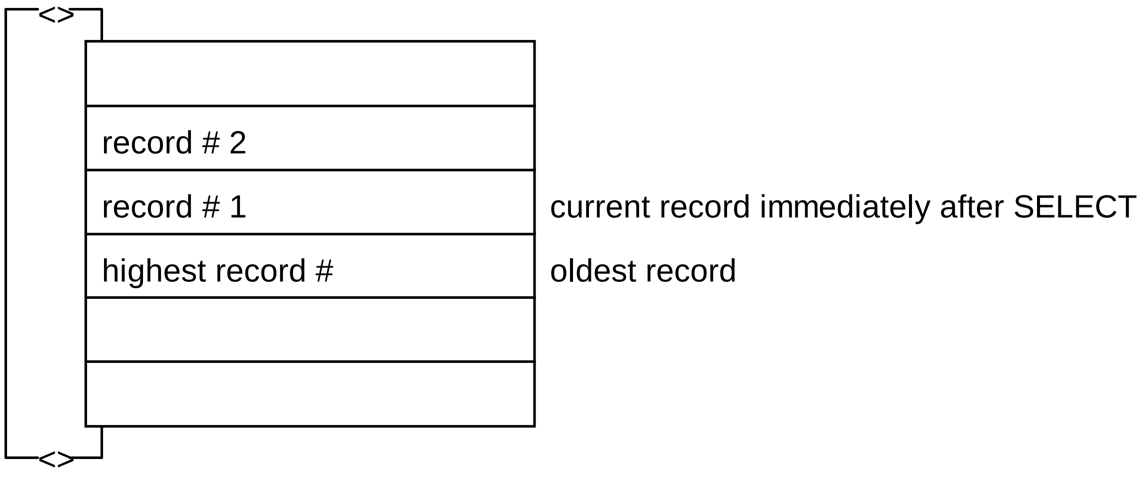


Figure 2: Cyclic EF organization

It is not possible to create more than 254 records in one file. Record "FF" is RFU.

6.2.6 EF containing ASC-set

An EF containing an Application Specific Command (ASC) set is a kind of filter which redirects commands and/or starts programs using procedures not defined in the general command set. An ASC can extend or modify the meaning of the general command set.

An ASC can only be associated to a given DF containing an application and becomes available when working on its sub-tree. When creating a DF, an information shall be given to the card indicating whether an EF containing an ASC-set may be available or not.

Only one ASC-set can be associated to a DF.

6.3 Contents of EF_{DIR}

EF_{DIR} may exist on any level. EF_{DIR} may have cross-references to DFs at any level according to business agreements.

The EF_{DIR} at the MF-level, contains for each non-hidden application the following information:

a) application IDentifier (ID);

b) verbal description (application label) of the application coded in ISO 8859-1 [17] (maximum 16 characters);

c) path (discretionary data) (sequence of 2 bytes file-IDs).

The coding of the contents of EF_{DIR} shall be in accordance with ISO/IEC 7816-5 [16].

6.4 Methods for selecting a file

See ISO/IEC 7816-4 [15].

When the channel-mechanism shall be used to select files, the operating system shall have to remember the Current File, the Current Directory, the Current EF and the status of the access conditions for each channel that is used.

With this channel mechanism, an exclusive select of the file (for one channel) shall be introduced. This is to prevent different applications to access data at the same time.

6.5 Application Specific Command (ASC) set

The procedure for adding an ASC-set to the card and guaranteeing that the added ASC-set shall not interfere with the existing functions and software is outside the scope of the present document and is under the responsibility of the application provider and the card issuer.

6.6 Invalidation/rehabilitation of a file

A file can be invalidated. As a result, this file is no longer available except for the functions SELECT, STATUS, DELETE and REHABILITATE (the REHABILITATE function is the reverse function of INVALIDATE).

An invalidated EF may still be read, if this possibility is indicated in the file status.

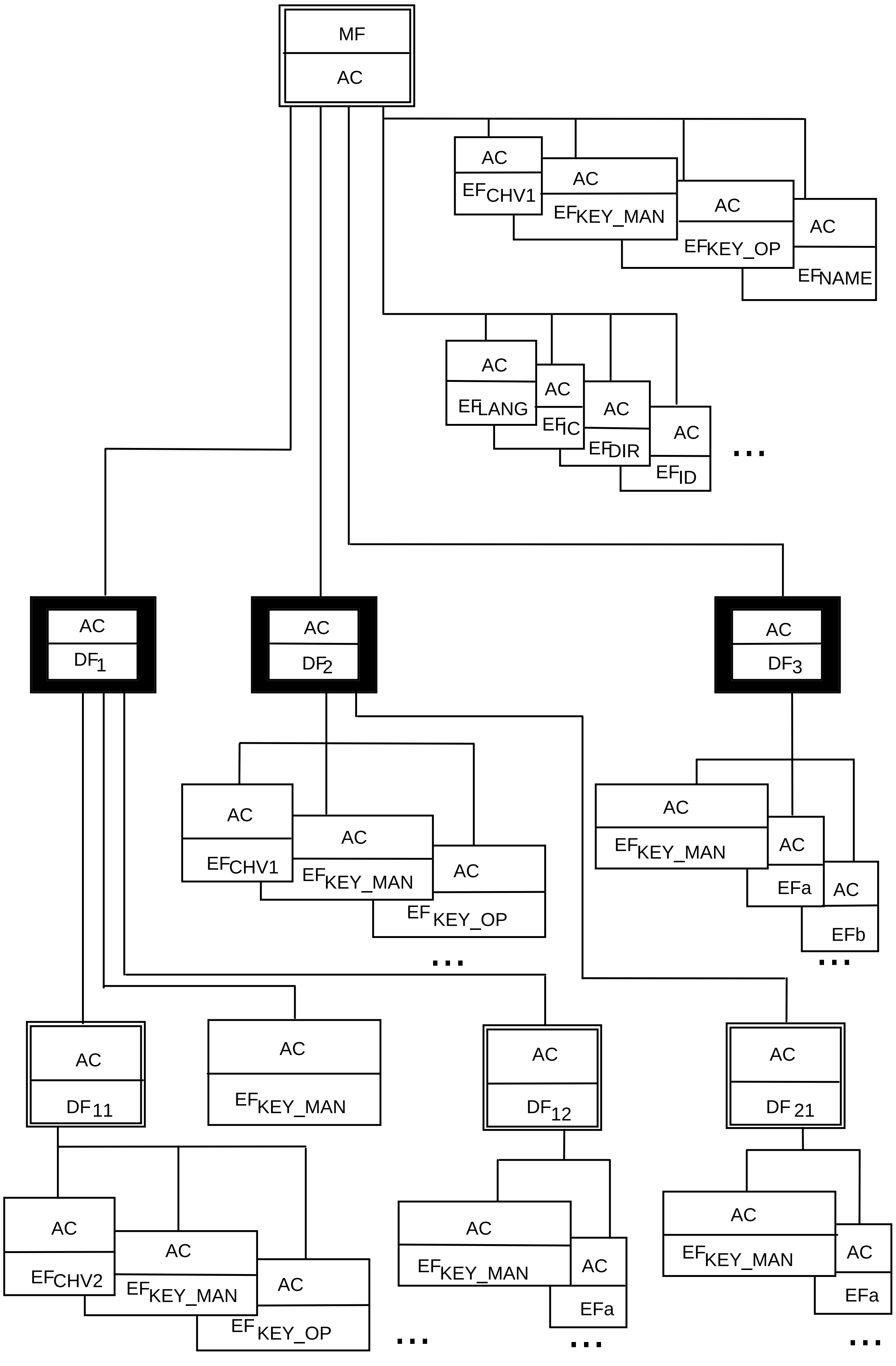


Figure 3: Example of the general structure with access control

7 Security facilities for the cards

7.1 Access Conditions (AC)

At each level of the file structure in the card, AC are defined. The AC are defined for groups of functions acting upon files with the same access requirements. The AC of the Current File shall be fulfilled before the file can be accessed through the functions.

EXAMPLE: If, in figure 3, EF_a below DF_{12} is accessed, only the AC of EF_a shall be fulfilled. The AC defined for DF_{12} and DF_1 are not relevant for the access to this EF_a.

7.1.1 Basic conditions and condition combinations

Basic conditions are:

ALW: the action shall **always** be possible. It can be performed without any restriction;

NEV: the action shall **never** be possible. It can only be performed internally;

CHV1: **Card Holder Verification** the action shall only be possible after a correct CHV presentation and/or biometric verification. CHV and/or biometric information shall be stored in the relevant EF_{CHV}. The term "relevant" means that for the Current File the EF_{CHV} can be a son file (a file one level lower), or if the EF_{CHV} does not exist there, or when it is invalidated, it shall be the relevant EF_{CHV} of the parent file, which may be at a higher level. In case of CHV larger than 8 bytes (e.g. a biometric template), the path to the EF containing the CHV is indicated in $EF_{CHV;}$

CHV2: see CHV 1, but CHV and/or biometric information shall be stored in EF_{CVH2} which may be located on a different level from EF_{CHV1};

PRO: the action shall only be possible in a **protected** way (allowing authentication of the origin and data integrity, see also subclause 7.6);

AUT: the action shall only be possible after an external authentication (i.e. authentication by the card of the terminal/application).

NOTE 1: Example of use of CHV1/CHV2 for different purposes: An issuer is only allowed to update the fixed dialling list (list of fixed numbers which are allowed to be used) in the card using CHV2, while the card user needs to enter CHV1 to read the fixed dialling list.

NOTE 2: An enciphered mode (ENC) is optional (according to ISO/IEC 7816-4 [15]).

Possible combinations of basic conditions give the following set of conditions:

ALW
CHV1
CHV2
PRO
AUT
CHV1/PRO
CHV2/PRO
CHV1/AUT
CHV2/AUT
NEV

Figure 4: Possible Access Conditions (AC)

7.1.2 EF containing the secret KEYS

There exist 2 types of functions, management functions and operational functions (see subclause 7.1.4). For each of these functions, it might be necessary to fulfil one of the following 2 access conditions: Protected (PRO) or AUThenticated (AUT). The necessary secret keys together with their respective keyfile version, keylength and algorithm ID shall be stored in the relevant keyfiles EF_{KEY_MAN} and EF_{KEY_OP}.

For the MF or a DF, the relevant keyfile is always the EF_{KEY_MAN}, which is a son file of the respective MF or DF.

For the current EF, depending on the function used, the keys to be used can be found in EF_{KEY_MAN} or in EF_{KEY_OP}. EF_{KE_OP} can only be created and loaded with keys under the responsibility of the DF it belongs to, using its key-file EF_{KEY_MAN}.

When creating an EF_{KEY_MAN} under a DF, there is no EF_{KEY_MAN} attached to this DF. Therefore in that case the relevant keyfile shall be found on the next upper level.

After creation, EF_{KEY_MAN} is empty (keylength of the first key is "00"). In this case, for writing the temporary keys, the relevant EF_{KEY_MAN} shall be found on the next upper level.

EF_{KEY_MAN} creation and loading with temporary keys at the MF level shall be performed under the same security steps as MF creation, i.e. under the responsibility of the card manufacturer.

EF_{KEY_OP} can be invalidated. In this case it is impossible to use the EF_{KEY_OP} of a higher level.

If there exists no EF_{KEY_OP} at the selected level, the EF_{KEY_OP} of a higher level shall be used.

7.1.3 EF containing CHV

For the functions which need to fulfil the condition CHV, the respective CHVs, shall be stored in the relevant EF_{CHV}.

This EF_{CHV} also contains:

a) the type of user identification that shall be used (see subclause 10.1);

b) the way (enciphered or not) in which the CHV and the UNBLOCK CHV shall be presented to the card;

c) the relevant key number if the way of presentation is enciphered;

d) the CHV attempts preset value N;

e) the remaining CHV attempts counter, is used to register consecutive bad CHV presentations. This decrementing counter shall be reset each time a correct CHV is entered. When it reaches zero, the CHV shall be blocked;

f) the UNBLOCK CHV (a kind of CHV allowing to preset with N the "remaining CHV attempts counter" and to unblock the CHV);

g) the remaining UNBLOCK CHV counter which is decremented each time the unblocking mechanism has not been used correctly. After a correct use of the unblocking mechanism it shall be preset again. When this counter reaches the value 0, then the corresponding application shall be blocked;

h) the number of remaining UNBLOCK mechanism use, which is a counter indicating the remaining number of times that the unblocking mechanism can still be used. This is also a counter which shall be decremented each time the unblocking mechanism is used and it shall never be reset.

7.1.4 File access conditions

Management functions, using the ACs of the MF, DF or a keyfile are:

a) DELETE FILE;

b) LOCK;

c) CREATE FILE; EXTEND FILE;

d) UPDATE BINARY; LOAD KEY FILE; (if the file to be updated is a keyfile);

e) INVALIDATE; REHABILITATE (if the file selected is the MF, a DF or a keyfile).

The following groups of management functions with the same AC requirements and having the same access conditions, are defined:

Table 1: Functions applicable on a DF

LOCK	RFU
DELETE FILE	CREATE FILE EXTEND FILE
REHABILITATE	INVALIDATE

Table 2: Functions applicable on a keyfile

LOAD KEY FILE	UPDATE
RFU	RFU
REHABILITATE	INVALIDATE

The keys used to fulfil these ACs are to be obtained from EF_{KEY_MAN}.

The coding of the functions are defined in subclause 9.3.

Operational functions, acting on an EF which is not a keyfile, are:

a) READ; SEEK;

b) UPDATE; WRITE;

c) INCREASE; DECREASE;

d) EXECUTE; CREATE RECORD;

e) INVALIDATE; REHABILITATE.

The following groups of operational functions with the same AC requirements and having the same access conditions, are defined:

Table 3: Functions applicable on an EF which is not a keyfile

READ/SEEK	UPDATE DECREASE
WRITE INCREASE	CREATE RECORD EXECUTE
REHABILITATE	INVALIDATE

The keys used to fulfil these ACs are to be obtained from EF_{KEY_OP}.

The coding of the functions are defined in subclause 9.3.

The WRITE (RECORD and BINARY) functions and the INCREASE function are located in the same group of Access Conditions (AC). However, for a given file, only one of the functions is valid (see subclause 9.3.1).

The UPDATE (RECORD and BINARY) functions and the DECREASE function are located in the same group of Access Conditions (AC). However, for a given file, only one of the functions is valid (see subclause 9.3.1).

For other functions no ACs are specified.

Table 4: Possible functions on files

Function	Current File				
	MF	DF	EF	Keyfile	EF_{CHV}
CLOSE APPLICATION	X	X			
CREATE FILE	X	X			
CREATE RECORD			X		
DECREASE			X		
DECREASE STAMPED			X		
DELETE FILE	X	X			
EXECUTE			X		
EXTEND	X	X			
INCREASE			X		
INCREASE STAMPED			X		
INVALIDATE	X	X	X	X	X
LOAD KEY FILE				X	
LOCK	X	X	X	X	X
READ BINARY			X		
READ BINARY STAMPED			X		
READ RECORD			X		
READ RECORD STAMPED			X		
REHABILITATE	X	X	X	X	X
SEEK			X		
SELECT	X	X	X	X	X
STATUS	X	X	X	X	X
UPDATE BINARY			X	X	X
UPDATE RECORD			X		
WRITE BINARY			X		
WRITE RECORD			X		

Table 4 explains which functions are possible when the current selected file is the MF, a DF, an EF, a keyfile (EF_{KEY_MAN} or EF_{KEY_OP}) or an EF_{CHV}.

The actions CREATE FILE, DELETE FILE, LOCK and EXTEND can only act upon son files of the Current Directory. When these actions are performed on a particular file, the ACs are obtained from the parent DF or MF.

The functions VERIFY CHV, CHANGE CHV, DISABLE CHV, ENABLE CHV, UNBLOCK CHV, INTERNAL AUTHENTICATION, EXTERNAL AUTHENTICATION, ASK RANDOM and GIVE RANDOM are possible, independently from the current file.

7.2 Possible functions on different EF types

Table 5a: EF types

Functions	EXTEND	INVAL REHAB	CREATE RECORD	UPDATE RECORD	READ RECORD	SEEK	READ BIN.	UPDATE BIN.
Filetype								
Linear Fixed struc	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Linear Var. struc	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Transparent	Yes	Yes	No	No	No	No	Yes	Yes
Cyclic	No	Yes	Yes	Yes(1)	Yes	Yes	No	No

Table 5b: EF types

Functions / Filetype	WRITE RECORD	WRITE BIN.	LOCK	EXEC.	SELECT/ STATUS	INCREASE/ DECREASE
Linear Fixed struc	Yes	No	Yes	No	Yes	No
Linear Var.struc	Yes	No	Yes	No	Yes	No
Transparent	No	Yes	Yes	Yes	Yes	No
Cyclic	Yes(note)	No	Yes	No	Yes	Yes
NOTE: Only for the next record to be written.						

UPDATE RECORD means an erase operation followed by a write operation.

NOTE: There is no DELETE function provided on the EF-level, in order to simplify the operating system and the memory management. It is up to the application to use for example a DELETE file combined with a restoring of a part of the file in order to delete some records.

7.3 Channel support

See ISO/IEC 7816-4 [15].

7.4 The CLOSE mechanism

For some applications, it may be necessary to have the possibility to delete the remembering of the access conditions already fulfilled. This possibility is provided by the CLOSE APPLICATION function (see subclause 8.26).

EXAMPLE: On a home terminal, after a security sensitive session, the card can stay in the terminal, but the application itself should not remain available. This is done with the CLOSE APPLICATION function.

7.5 Security context

The operating system shall remember the access conditions CHV and AUT which have already been fulfilled, until the card session ends (reset or de-activation of the card) or the application is explicitly closed (using the CLOSE APPLICATION function).

The purpose is not to ask twice for the same user identification (CHV and/or biometric) and/or external authentication. When several logical channels are supported, the access condition AUT shall be fulfilled for each logical channel. The access condition CHV, which is user related, is considered to be a common AC (if the same EF_{CHV} is being used) and can therefore be fulfilled once for all the logical channels. If each interaction between the external world and the card has to be authenticated, then AC = PRO may be used (instead of external authentication).

NOTE: The operating system has to keep track for every CHV and every possible key, which is the CHV or the AUT status.

7.6 General description of security functions

From security point of view, the AC PRO is only defined for the following types of functions:

a) functions where the data contained in the command will change the data contents in the card;

b) functions, not containing input data, but changing the file-status.

Functions, that don't change the file-status, but only require data to be send out from the card, don't require the AC PRO in the card. In this case a stamped mode is used, where the external world imposes whether the data to be send out by the card is protected by a cryptogram. This shall be done, using special commands.

7.6.1 Security functions linked to AC

Several functions can be performed with AC = PRO. For these functions, a cryptogram is part of the data sent to the card.

A cryptogram is the result of a cryptographic process, based on the algorithm and the secret key which are implicitly linked to the respective function, for the current selected file, through the keynumber in the AC-coding (see subclause 9.3). The secret key and the algorithm-ID used, are specified in the relevant keyfile (EF_{KEY_MAN} or EF_{KEY_OP}) (see subclauses 10.6 and 10.7).

When sending a command in protected mode to the card, first the external world has to ask a random number from the card. Then the external world shall compute the cryptogram using the following inputs:

a) random number;

b) part of the header of the command (INS, P1, P2);

c) length of the data field and data or none.

The cryptogram shall be joined at the end of the data send to the card.

After the reception of the command, the card checks if the cryptogram is consistent with the inputs (random number, header, data if some). If they match the command is accepted and then executed. Otherwise the command is rejected.

7.6.2 Security functions linked to the stamped mode

The functions (READ BINARY, READ RECORD, INCREASE, DECREASE) can be performed in stamped mode. For these functions, a cryptogram is part of the response data given by the card.

The cryptogram to be sent out, is the result of a cryptographic process, based on the algorithm and the secret key which are implicitly linked to the respective function, for the current selected file, through the keynumber in the AC-coding (see subclause 9.3). The secret key and the algorithm-ID used, are specified in the relevant keyfile (EF_{KEY_MAN} or EF_{KEY_OP}), see subclauses 10.6 and 10.7.

In order to compute the cryptogram for the response data, the card has to use the previously given challenge (random number). The cryptogram is computed on the following inputs:

a) challenge (random number);

b) part of the header (INS, P1, P2) of the command performed;

c) data;

d) maximum length of data expected in the response.

The cryptogram shall be joined at the end of the data provided by the card.

For the modes of INCREASE STAMPED and DECREASE STAMPED refer to subclauses 9.2.31 and 9.2.33. For the modes of other cryptogram calculations refer to EN 726-2.

7.6.3 External authentication

This function shall be used by the terminal/application to be authenticated by the card. For this purpose, the card gives a random number (a challenge) to the terminal, which replies to the card with a cryptogram. This cryptogram shall be verified by the card running a cryptographic algorithm using the previously sent random number and a secret key. The secret key and the algorithm ID to be used for this process shall be in the relevant keyfile (see subclause 8.25). The key number is a parameter given by the terminal/application to the card. If the verification of the cryptogram is positive, the AUT condition is fulfilled, and shall be remembered by the card, for each possible key, until the end of the session (see subclause 7.5).

7.6.4 Internal authentication

This function shall be used by the terminal/application to authenticate the card. For this purpose, the terminal shall give a challenge to the card, which replies with a cryptogram to this challenge. This response shall be verified by the terminal, using a cryptographic algorithm, the previously sent challenge number and a secret key. The secret key and the algorithm ID to be used for this process shall be in the relevant keyfile (see subclause 8.22). The key number is a parameter given by the terminal/application to the card.

NOTE: The challenge should usually be a random number. However, instead of a random number, also another authentication parameter can be used (e.g. the time or a sequence number).

7.6.5 Algorithm ID

Table 6: Algorithm IDs

Algorithm	ID	AUTHENTICATION	PRO	STAMPED	KEY LOAD
DSAA	**"1"**	**X**			
COMP NAT	**"2"**	**X**			
USA4	**"3"**				
TESA-7	**"4"**	**X**	**X**	**X**	**X**
COMP 128	**"40"**	**X**			
Escape for Proprietary Algorithms	**"70"-"7F"**				
NOTE 1: Algorithms used for encipherment, and available for general use, are not defined in the present document. However, such algorithms might be incorporated for proprietary use. NOTE 2: DSAA means DECT Standard Authentication Algorithm.					

8 Description of the functions

The following general functions are defined:

1) SELECT;
2) STATUS;
3) CREATE FILE;
4) DELETE FILE;
5) EXTEND;
6) EXECUTE;
7) UPDATE BINARY;
8) UPDATE RECORD;
9) CREATE RECORD;
10) READ BINARY;
11) READ BINARY STAMPED;
12) READ RECORD;
13) READ RECORD STAMPED;
14) SEEK;
15) VERIFY CHV;
16) CHANGE CHV;

17) DISABLE CHV;

18) ENABLE CHV;

19) UNBLOCK CHV;

20) INVALIDATE;

21) REHABILITATE;

22) INTERNAL AUTHENTICATION;

23) ASK RANDOM;

24) GIVE RANDOM;

25) EXTERNAL AUTHENTICATION;

26) CLOSE APPLICATION;

27) WRITE BINARY;

28) WRITE RECORD;

29) LOCK;

30) DECREASE;

31) DECREASE STAMPED;

32) INCREASE;

33) INCREASE STAMPED;

34) LOAD KEY FILE.

It shall not be mandatory for all cards complying to the present document to support all the described functions of the present document. Therefore, card profiles are defined (see subclause 10.5). At minimum all cards shall support SELECT and READ BINARY.

The expected results of a function or the possible status conditions, are always returned by the card after sending the corresponding command. They are reflected in the status information bytes which are returned by the card after each command. In subclause 9.4 all the possible status information conditions and their corresponding coding are given for each possible command.

If an invalid command is received or the requirements for performing the function are not all fulfilled, the function shall not be performed and there shall be no change in the data contained in the file structure unless otherwise specified for the function (e.g. to record failed CHV attempts).

In all representations, the leftmost bit represents the most significant bit of the most significant byte while the rightmost bit represents the least significant bit of the least significant byte.

b8 b1							b8 b1
byte 1	2	3	4	5	6	7	byte 8

Figure 5: Notation for a string of bytes

8.1 SELECT

This function allows the selection of a file, according to the methods of file-selection, according to ISO/IEC 7816-4 [15].

Following movements are possible by using the file ID, the path or the application identifier:

a) downwards;

b) horizontally;

c) upwards;

d) from the MF.

After a successful selection of the MF or a DF, the selected file becomes the new Current Directory and there is no current EF. After a successful selection of an EF, the selected file becomes the new Current EF and the current DF is (or becomes) the directory containing the selected file, but the record pointer is not defined (except for cyclic files, according to subclause 6.2.5). To reach a record it shall be mandatory to set the record pointer by using the modes: next, previous, first or last for the functions: SEEK, READ, UPDATE or WRITE.

Input:	Refer to subclause 9.2.1 for a description of the input parameters.
Output:	Refer to subclause 9.2.1 for a description of the output data in case the selected file is a DF, an EF, an EF_{CHV} or a keyfile.

NOTE: For this function, no ACs are defined.

8.2 STATUS

This function returns information concerning the Current Directory. The Current File is not affected by the STATUS function.

Input:	None.
Output:	Refer to subclause 9.2.2 for a description of the output data.

NOTE: For this function, no ACs are defined.

8.3 CREATE FILE

This function allows the creation of a new file under the Current Directory. The AC for the CREATE FILE function of the Current Directory shall be fulfilled.

The file ID shall be included in the command.

It shall be the responsibility of the issuer to update EF_{DIR} in case of the creation of a new application.

When creating an EF with linear fixed or cyclic structure it may be possible either to just reserve the space allocated to the file without any record formatted or to directly create as many records as allowed by the requested file size. After creation of an EF of linear variable structure the file is empty (i.e. no record can be accessed).

During file creation it may be possible to initialize each byte of an EF with the same value (e.g. "00" or "FF"). The value "00" is required for creation of an EF_{CHV} as well as for creation of a keyfile.

It may also be possible to create a file without modifying the state of its allocated data space. In that case, care shall be taken by the application provider that this feature cannot be used for disclosing sensitive data which would not have been destroyed by a former DELETE command.

If an executable EF has to be created, the AC to be fulfilled shall be those valid for the MF, since the creation of an executable file resides under the responsibility of the card issuer. For an EF containing a program, the AC shall be forced to NEV or AUT for the READ functions and to PRO for the UPDATE functions. Other options shall not be possible.

If an invalidated file is to be created, and the card does not support the INVALIDATE / REHABILITATE functionality, the card may optionally reject the CREATE function.

If an EF_{CHV} is to be created with a disabled CHV, and the card does not support the ENABLE / DISABLE CHV functionality, the card may optionally reject the CREATE function.

If AC = PRO, the card authenticates the origin and the contents of the command, by verifying the cryptogram (input c) generated by the terminal from the input parameters (inputs a and b) and the random number previously given by the card after the ASK RANDOM function. The CREATE function shall not be performed by the card if the verification of the cryptogram is incorrect. The key which is used for this verification shall be the one indicated in the AC for the Current Directory, for the CREATE FILE function.

After the creation of a DF, the Current Directory shall be on the newly created file. In case of an EF creation, the Current EF shall be on the newly created file and the Current Directory is unchanged. After creation of an EF with linear structure, the record pointer is not defined. After creation of an EF with cyclic structure, the position of the record pointer depends on the type of creation: if the creation of the EF does not create any record the current record pointer is not defined: if the creation of the EF directly creates records, the current record pointer is on the last created record (record #1).

For the creation of the MF, the relevant security steps shall be defined by the card-manufacturer (see annex C).

The memory space allocated shall be definitively reserved.

Input:	a/ initialization value and type of creation.
	b/ the file-parameters.
	c/ a cryptogram.
	Refer to subclause 9.2.3 for a description of the input parameters in case the file to be created is a DF, an EF, an EF$_{CHV}$ or a keyfile.
Output:	None.

8.4 DELETE FILE

This function allows to delete a file under the Current Directory.

The AC for DELETE FILE, defined for the parent file of the file to be deleted, shall be fulfilled.

Deletion means that the file (header and data) shall be definitively destroyed such that it is not recoverable. The entire contents of the file shall be overwritten with a constant value. It shall not be possible to interrupt this process in such a way that the data can become recoverable.

It shall only be possible to delete a DF if it contains no further files.

NOTE: If a file having an application identifier is deleted, the issuer shall ensure that the corresponding entry in EF$_{DIR}$ is deleted.

If AC = PRO, the input consists of (a, b), where (b) is the cryptogram, calculated by the terminal from the input-parameter (a) and the random number previously given by the card after the ASK RANDOM function.

Input:	a/ file ID.
	b/ cryptogram.
	Refer to subclause 9.2.4 for a description of the input data.
Output:	None.

8.5 EXTEND

This function allows the extension of the memory space allocated to the file under the Current Directory. The EXTEND function shall only be performed if there is enough space available in the parent file in order to allow this extension.

This function shall not be allowed for a cyclic EF, for an EF containing a program, and for the MF. The AC for EXTEND FILE, defined for the parent file of the file to be extended, shall be fulfilled.

When extending an EF with linear fixed structure it may be possible to directly create as many records as allowed by the requested additional size if the whole file is already formatted. The extension of an EF with linear variable structure does not format any additional record.

It may be possible to initialize each byte of the extended part with the same value (e.g. "00" or "FF").

It may also be possible to extend a file without modifying the state of its newly allocated data space. In that case, care shall be taken by the application provider that this feature cannot be used for disclosing sensitive data which would not have been destroyed by a former DELETE command.

When extending an EF with linear structure, the record pointer is not affected.

The input shall only consist of input (a, b, c), except if AC = PRO (see below).

If AC = PRO, the input consists of inputs (a, b, c and d). The card shall authenticate the origin and the contents of the command, by verifying the cryptogram (input d), calculated by the terminal from the input-parameters (a, b and c) and the random number previously given by the card after the ASK RANDOM function. The EXTEND function shall not be performed by the card if the verification of the cryptogram is incorrect. For the EXTEND function the key used for this verification shall be the one indicated in the AC of the Current Directory.

Input:	a/ initialization value and type of extension.
	b/ file ID.
	c/ the number of bytes to be extended.
	d/ cryptogram.
	Refer to subclause 9.2.5 for a description of the input data.
Output:	None.

8.6 EXECUTE

This function allows to run a program stored in an EF after having selected this EF. The function shall only be allowed for an EF containing a program. The AC which shall be fulfilled are the ACs of the Current EF for the EXECUTE function.

The input shall consist only of input (a), and the output shall consist of output (a), except for the AC = PRO (see below).

If AC = PRO, the input shall consist of inputs (a, b). The card authenticates the origin and the contents of the command, by verifying the cryptogram (input b), calculated by the terminal from the input-parameter (a) and the random number previously given by the card after the ASK RANDOM function. The EXECUTE function shall not be performed by the card if the verification of the cryptogram is incorrect. The key which shall be used for this verification is the one indicated in the AC of the Current EF, for the EXECUTE function.

Input:	a/ none or data.
	b/ cryptogram.
	Refer to subclause 9.2.6 for a description of the input data.
Output:	a/ none or data.

8.7 UPDATE BINARY

This function allows the updating of the current EF (which shall be a transparent EF), with a string of bytes. The function can be performed according to the AC defined for this EF for the UPDATE function. For updating an EF containing a program (see note) (indicated in the type of file in the select response), the ACs shall be those defined for the CREATE FILE function at the MF level.

NOTE: The card issuer shall take all necessary steps to ensure that there is no possibility of interference between applications.

An update can be considered as a "replacement" (erase, followed by a write) of the bits already present in the card memory by the bits given in the update command.

The input shall only consist of inputs a, b, except if AC = PRO (see below).

If AC = PRO, the input consists of inputs a, b and c. The card shall authenticate the origin and the contents of the command, by verifying the cryptogram (input c) calculated by the terminal/application from the input-parameters (a, b) and the random number previously given by the card after an ASK RANDOM function. The UPDATE function shall not be performed by the card if the verification of the cryptogram is incorrect. The key which is used for this verification shall be the one indicated in the AC of the Current EF, for the UPDATE functions.

Input:	a/ relative address (offset) from the beginning of the EF and the length (in bytes) of the string to be updated.
	b/ data to be updated.
	c/ cryptogram.
	Refer to subclause 9.2.7 for a description of the input data.
Output:	none.

8.8 UPDATE RECORD

This function allows the updating of one record in the Current EF. The EF shall be of the linear or the cyclic type. If the EF is a linear type, the record to be updated can be the current, the next or the previous one (this refers to the previously performed READ, UPDATE or SEEK function), the first, the last, or the record indicated by its record number. If the EF is a cyclic type, only the previous record before the last written record (which is the oldest written record) shall be updated, using the PREVIOUS mode.

The UPDATE can be considered as a "replacement" (erase, followed by a write) of the bytes already present in the card memory by the bytes given in the command.

Six modes are defined:

a) CURRENT and ABSOLUTE modes: the record pointer is not affected by this function;

b) NEXT mode: the record pointer is incremented before the UPDATE RECORD function is performed. If the record pointer has not been previously set within the selected EF, then UPDATE (next) shall update the first record in the EF and set the record pointer to this record. If the record pointer is already situated at the last record in the EF, UPDATE (next) shall not cause the record pointer to be shifted and no record data shall be updated;

c) PREVIOUS mode: the record pointer is decremented before the UPDATE RECORD function is performed. If the record pointer has not been previously set within the selected EF, then UPDATE (previous) shall update the last record in the EF and set the record pointer to this record. If the record pointer is already situated at the first record in the EF, UPDATE (previous) shall not cause the record pointer to be shifted, and no record data shall be updated (except for a cyclic file);

d) FIRST and LAST modes: the record pointer becomes the first or the last record before the UPDATE RECORD function is performed.

The function shall only be performed if the number of bytes to be written is consistent with the record length and file structure. For fixed length records the function shall only be performed if the number of bytes to be written is the same as the record length. For variable length records the card may also allow this function to be performed if the number of bytes to be written is different from the current length of the requested record, in which case the record length shall be changed according to the supplied data.

The function is performed according to the AC defined for the UPDATE function for this EF.

If the UPDATE function is allowed for an EF, the DECREASE function shall not be valid for this EF (see subclauses 7.1.4 and 9.3.1).

The input shall only consist of inputs a, b except if AC = PRO (see below).

If AC = PRO, the input shall consist of inputs a, b and c. The card shall authenticate the origin and the contents of the command, by verifying the cryptogram (input c) calculated by the terminal/application from the input-parameters (a, b) and the random number previously given by the card after an ASK RANDOM function. The UPDATE function shall not be performed by the card if the verification of the cryptogram is incorrect. The key used for this verification shall be the one indicated in the AC of the Current EF, for the UPDATE functions.

Input:	a/ indication first/ last/ next/ previous/ current or absolute (record number) and the length (in bytes) of the data to be updated.
	b/ the data of the record to be updated.
	c/ cryptogram.
	Refer to subclause 9.2.8 for a description of the input data.
Output:	None.

8.9 CREATE RECORD

This function allows to append a record (and fill it) at the logical end of an EF of the linear or cyclic type, inside the memory space allocated during file creation.

The function shall only be performed if the number of bytes to be written is consistent with the record length and file structure. For fixed length records the function shall only be performed if the number of bytes to be written is the same as the record length.

This function can be performed according to the AC defined for this EF for the CREATE RECORD function.

It shall only be possible to create a new record in an EF with cyclic structure if the file is empty (i.e. no record already formatted) or if the last created record is also the last written.

The input shall only consist of input (a) except if AC = PRO (see below).

If AC = PRO, the input shall consist of inputs a, b. The card shall authenticate the origin and the contents of the command, by verifying the cryptogram (input b) calculated by the terminal/application from the input-parameter (a) and the random number previously given by the card after an ASK RANDOM function. The CREATE RECORD function shall not be performed by the card if the verification of the cryptogram is incorrect. The key used for this verification shall be the one indicated in the AC of the Current EF, for the CREATE RECORD function.

The record pointer will indicate the newly created record in the Current EF.

Input:	a/ record contents.
	b/ cryptogram.
	Refer to subclause 9.2.9 for a description of the input data.
Output:	None.

NOTE: It is possible with this function to have linear files which can be written only once (like an EPROM memory). This can be accomplished by putting the AC for WRITE or UPDATE to NEVER. As a consequence, the only possibility to write in such an EF is to perform the CREATE RECORD function which appends to a file a new record.

8.10 READ BINARY

This function allows to read out a string of bytes from the Current EF (which has to be a transparent EF).

The function can be performed according to the AC defined for this EF for the READ functions.

Input:	Relative address (offset) from the beginning of the EF and the length (in bytes) of the string to be read.

Output: Data.

Refer to subclause 9.2.10 for a description of the input and output data.

NOTE: AC=PRO is not applicable for this function.

8.11 READ BINARY STAMPED

This function has the same functionality as the previous described READ BINARY which is defined in subclause 8.10. In addition the READ BINARY STAMPED function adds a cryptogram to the output data.

In order to calculate the cryptogram to be sent out by the card, a challenge shall be given before to the card, using the GIVE RANDOM function.

The input consists of (a) and the output shall consist of (a, b). The relevant key to be used is defined in the AC for the READ function.

Input: a/ relative address (offset) from the beginning of the EF and the length (in bytes) of the string to be read.

Output: a/ data.

b/ cryptogram.

Refer to subclause 9.2.11 for a description of the input and output data.

NOTE: AC=PRO is not applicable for this function.

8.12 READ RECORD

This function allows to read out one record from the Current EF. The EF shall be of the linear or the cyclic type. The record to be read can be the current, the next or the previous one. This refers to the previously performed READ, UPDATE or SEEK function. It can also be the first, last, or the record indicated by its record number.

Six modes are defined:

a) CURRENT and ABSOLUTE modes: the record pointer is not affected by this function;

b) NEXT mode: the record pointer is incremented before the READ RECORD function is performed. If the record pointer has not been previously set within the selected EF, then READ (next) shall read the first record in the EF and set the record pointer to this record. If the record pointer is already situated at the last record in the EF, READ (next) shall not cause the record pointer to be shifted and no record data shall be returned (except for cyclic file);

c) PREVIOUS mode: the record pointer is decremented before the READ RECORD function is performed. If the record pointer has not been previously set within the selected EF, then READ (previous) shall read the last record in the EF and set the record pointer to this record. If the record pointer is already situated at the first record in the EF, READ (previous) shall not cause the record pointer to be shifted, and no record data shall be returned (except for cyclic file);

d) FIRST and LAST modes: the record pointer becomes the first or the last record before the READ RECORD function is performed.

When the number of bytes requested from the record equals its length, the card shall return the complete record.

When the number of bytes requested from the record is not equal to its length, the card may either respond with incorrect parameter P3, or return record data according to the following:

a) if the number of bytes requested is less than the record length, the card may return the requested number of bytes (0 bytes are requested if Le is empty);

b) if the number of bytes requested is greater than the record length, the card may return the complete record. Only data of the requested record shall be returned;

c) if Le = 0 the function may either return the data of the requested record only, or may return all of the data starting from the requested record until the end of the file has been reached or until the maximum length of response has been given. For a cyclic file, the end of the file is considered to be reached when all records up to the oldest record have been read.

The function can be performed according to the AC defined for this EF for the READ functions.

Input: Indication current/ previous/ next/ first/ last, or absolute (record number).

Output: The data of the record read.

Refer to subclause 9.2.12 for a description of the input and output data.

NOTE: AC=PRO is not applicable for this function.

8.13 READ RECORD STAMPED

This function has the same functionality as the previous described READ RECORD which is defined in subclause 8.12. In addition the READ RECORD STAMPED function adds a cryptogram to the output data.

In order to calculate the cryptogram to be sent out by the card, a challenge shall be given before to the card, using the GIVE RANDOM function.

The input consists of (a) and the output shall consist of (a) and (b). The relevant key to be used is defined in the AC for the READ function.

Input: a/ indication current/ previous/ next/ first/ last, or absolute (record number).

Output: a/ the data of the record read.

b/ cryptogram.

Refer to subclause 9.2.13 for a description of the input and output data.

NOTE: AC=PRO is not applicable for this function.

8.14 SEEK

This function is used to locate a record containing a certain pattern inside the Current EF. The function shall be valid for all types of EFs containing records.

After a successful SEEK, the record found will be indicated by the record pointer. After an unsuccessful SEEK the record pointer shall remain unchanged.

Input: a/ offset, type of response, seek-mode, pattern-length.

b/ pattern.

Refer to subclause 9.2.14 for a description of the input data.

Output: None or record number;

Refer to subclause 9.2.14 for a description of the eventual output data.

The SEEK is performed with the length of the pattern. The pattern is compared to the indicated number of bytes starting from the given offset from the start of the record data. Only one comparison is made per record. If this pattern does not match, another record is tested until a match is found or all records have been compared. If no conditions match, record not found is reported (record shorter than pattern length = no match).

SEEK from the next-record forward: if the record pointer has not been previously set within the selected file, the search begins on the first record of the selected file.

SEEK from the previous-record backward: if the record pointer has not been previously set within the selected file, the search begins on the last record of the selected file.

If the record pointer is already situated at the last record in the EF, SEEK with the option start from next location forward shall not cause the record pointer to be shifted and no record number shall be returned (except for a cyclic file).

If the record pointer is already situated at the first record in the EF, SEEK with the option start from previous location backward shall not cause the record pointer to be shifted and no record number shall be returned (except for a cyclic file).

NOTE: AC = PRO is not applicable for this function.

8.15 VERIFY CHV

This function allows verification of the CHV, which is defined in the relevant EF_{CHV} for the Current File (see subclause 7.1).

If the CHV status is blocked or disabled, the function shall end unsuccessfully. If the CHV status is not blocked and is enabled, the presented CHV shall be checked. If the presented CHV is false, the "remaining CHV attempts counter" shall be decremented. If the presented CHV is correct, the "remaining CHV attempts counter" shall be preset to N and the relevant CHV1 or CHV2 access condition shall be fulfilled.

After N consecutive false CHV presentations, the respective CHV shall be blocked and the AC (CHV1 or CHV2) can never be fulfilled until the UNBLOCK CHV function has been successfully performed.

If the presented CHV is false, the current status of the relevant CHV1 or CHV2 access condition shall be maintained, unless the CHV status becomes blocked. When the incorrect CHV causes the CHV to become blocked, the relevant CHV1 or CHV2 access condition shall be lost.

The CHV can be presented in clear text or enciphered, according to the type of presentation which has been defined for this CHV and which is indicated in the relevant EF_{CHV}. It is up to the operating system to manage the way to present the CHV in accordance with EF_{CHV}.

If the relevant EF_{CHV} indicates that the CHV shall be presented enciphered, the input shall consist of a and b where inputs are enciphered. The enciphering shall be performed by the terminal/application before the CHV is presented to the card. The encipherment process includes the random number previously given by the card after an ASK RANDOM function while the relevant key for the enciphered CHV presentation shall be defined in the relevant EF_{CHV}.

Input: a/ indication CHV1/CHV2 (P2 of the header).

b/ the CHV (enciphered or not), shall be 8 bytes long (In case of a biometric CHV, the length may be different). If the CHV does not contain 8 bytes, it shall be up to the terminal for padding the remaining bytes of the CHV to the right with binary ones (see EN 726-4 subclause 10.3.3).

Refer to subclause 9.2.15 for a description of the input data.

Output: None.

NOTE: For this function, no ACs are defined.

8.16 CHANGE CHV

This function allows the change of the CHV which protects the Current File, but only if the following conditions are fulfilled:

a) indication CHV change allowed or not (given in the SELECT response etc.) is positive;

b) the CHV is not disabled;

c) the CHV is not blocked.

The old and the new CHV can be presented in clear text or enciphered, according to the type of presentation which has been defined for this CHV and which is indicated in the relevant EF_{CHV} for the Current File (see subclause 7.1). The relevant key number for enciphered CHV presentation is also defined in this EF_{CHV}.

Successful completion of this function shall fulfil the relevant CHV1 or CHV2 access condition and preset the "remaining CHV attempts counter" to N.

If the presented CHV is incorrect, then the "remaining CHV attempts counter" shall be decremented. After N consecutive false CHV presentations, the respective CHV shall be blocked and the AC (CHV1 or CHV2) can never be fulfilled until the UNBLOCK CHV function has been successfully performed.

If the presented CHV is false, the current status of the relevant CHV1 or CHV2 access condition shall be maintained, unless the CHV status becomes blocked. When the incorrect CHV causes the CHV to become blocked, the relevant CHV1 or CHV2 access condition shall be lost.

If the relevant EF_{CHV} indicates that the CHVs shall be presented enciphered, the enciphering shall be performed by the terminal/application before the CHVs is presented to the card. The encipherment process includes the random number previously given by the card after an ASK RANDOM function, while the relevant key for the enciphered CHV presentation shall be defined in the relevant EF_{CHV}.

Input:	a/ indication CHV1/CHV2 (P2 of the header).
	b/ Old CHV, new CHV (both enciphered or not), shall be 8 bytes long (In case of a biometric CHV, the length may be different). If the CHVs does not contain 8 bytes, it shall be up to the terminal for padding the remaining bytes of the CHVs to the right with binary ones (see EN 726-4 subclause 10.3.3).
	Refer to subclause 9.2.16 for a description of the input data.
Output:	None.

NOTE: For this function, no ACs are defined.

8.17 DISABLE CHV

This function directly acts on the relevant EF_{CHV}. Its purpose is to disable the verification of the relevant CHV which means that the successful CHV verification is no longer mandatory in order to fulfil AC = CHV.

This function shall only be possible if DISABLE/ENABLE CHV are allowed for the relevant CHV (see SELECT response in case of EF_{CHV}). The CHV disabling remains effective until it has been enabled with the function ENABLE CHV.

Successful completion of this function shall fulfil the relevant CHV1 or CHV2 access condition and preset the "remaining CHV attempts counter" to N.

If the presented CHV is false, the current status of the relevant CHV1 or CHV2 access condition shall be maintained, unless the CHV status becomes blocked. When the incorrect CHV causes the CHV to become blocked, the relevant CHV1 or CHV2 access condition shall be lost.

If the relevant EF_{CHV} indicates that the CHV shall be presented enciphered, the enciphering shall be performed by the terminal/application before the CHV is presented to the card. The encipherment process includes the random number previously given by the card after an ASK RANDOM function, while the relevant key for the enciphered CHV presentation shall be defined in the relevant EF_{CHV}.

The function DISABLE CHV shall not be executed when the CHV is already disabled or when the CHV is blocked. If the presented CHV is incorrect, then the CHV attempt counter shall be decremented. After N consecutive false CHV presentations, the respective CHV shall be blocked and the AC (CHV1 or CHV2) can never be fulfilled until the UNBLOCK CHV function has been successfully performed.

Input:	a/ indication CHV1/CHV2 (P2 of the header).
	b/ CHV (enciphered or not).
	Refer to subclause 9.2.17 for a description of the input data.
Output:	None.

NOTE: For this function, no ACs are defined.

8.18 ENABLE CHV

This function directly acts on the relevant EF_{CHV}. Its purpose is to enable the verification of the relevant CHV which means that the successful CHV verification is mandatory to fulfil AC = CHV.

This function shall only be possible if DISABLE/ENABLE CHV are allowed for the relevant CHV (see SELECT response in case of EF_{CHV}). The CHV enabling remains effective until it has been disabled with the function DISABLE CHV.

Successful completion of this function shall fulfil the relevant CHV1 or CHV2 access condition and preset the "remaining CHV attempts counter" to N.

If the presented CHV is incorrect, then the CHV attempt counter shall be incremented. After N consecutive false CHV presentations, the respective CHV shall be blocked and the AC (CHV1 or CHV2) can never be fulfilled until the UNBLOCK CHV function has been performed.

If the presented CHV is false, the current status of the relevant CHV1 or CHV2 access condition shall be maintained, unless the CHV status becomes blocked. When the incorrect CHV causes the CHV to become blocked, the relevant CHV1 or CHV2 access condition shall be lost.

The function ENABLE CHV shall only be performed when the CHV is disabled and not blocked.

If the relevant EF_{CHV} indicates that the CHV shall be presented enciphered, the enciphering shall be performed by the terminal/application before the CHV is presented to the card. The encipherment process includes the random number previously given by the card after an ASK RANDOM function, while the relevant key for the enciphered CHV presentation shall be defined in the relevant EF_{CHV}.

Input:	a/ indication CHV1/CHV2 (P2 in the header).
	b/ CHV (enciphered or not).
	Refer to subclause 9.2.18 for a description of the input data.
Output:	None.

NOTE: For this function, no ACs are defined.

8.19 UNBLOCK CHV

This function allows the unblocking and presetting of a CHV which has been blocked after N (indicated in EF_{CHV}) consecutive wrong CHV presentations. This function may be performed whether or not the relevant CHV is blocked. The UNBLOCK CHV is a special kind of CHV.

Each time a blocked CHV is unblocked, the number of "USE OF THE UNBLOCK MECHANISM", as defined in the relevant EF_{CHV}, shall be decremented. The number shall not be decremented if the CHV was not previously blocked or if the unblock attempt was not successful. If this number becomes zero, the relevant CHV cannot be unblocked any more. If this number is set to "FF", this mechanism shall be allowed to be used an infinite number of times.

If the presented UNBLOCK CHV is incorrect, the "remaining UNBLOCK CHV attempts counter" shall be decremented. When the value 0 is reached the UNBLOCK CHV is blocked. A false UNBLOCK CHV shall have no effect on the status of the respective CHV itself.

If the presented UNBLOCK CHV is correct, the corresponding "remaining CHV attempts counter" in the relevant EF_{CHV} shall be preset with N, the CHV preset with the value of the new CHV and the "remaining UNBLOCK CHV attempts counter" preset to 10. After a successful unblocking attempt the CHV is enabled and the relevant access condition level is satisfied.

The UNBLOCK CHV can be presented in clear text or enciphered, according to the type of presentation which has been defined for this kind of CHV and which is indicated in the relevant EF_{CHV}. The relevant key for enciphered UNBLOCK CHV presentation is also defined in this EF_{CHV} and shall be the same as for the enciphered CHV presentation.

If the relevant EF_{CHV} indicates that the CHV/UNBLOCK CHV shall be presented enciphered, the enciphering shall be performed by the terminal/application before the CHV/UNBLOCK CHV is presented to the card. The encipherment process includes the random number previously given by the card after an ASK RANDOM function, while the relevant key for the enciphered CHV presentation shall be defined in the relevant EF_{CHV}.

Successful unblocking is equivalent to a successful CHV-presentation.

Input:	a/ indication CHV1/CHV2.
	b/ the UNBLOCK CHV and the new CHV (both enciphered or not).
	Refer to subclause 9.2.19 for a description of the input data.
Output:	None.

NOTE: For this function, no ACs are defined.

8.20 INVALIDATE

This function only allows to invalidate the Current File. After an INVALIDATE function, the bit 1 of byte 12 (file status, see subclause 9.2.1) shall be zero for the concerned file. The ACs which shall be fulfilled are the ACs for the Current File, for the INVALIDATE function.

A file which is invalidated is not available any more, within the application, for any function, except for the SELECT, STATUS, DELETE and REHABILITATE functions. The function READ may be performed when the indication "readable when invalidated" (see file status, subclause 9.2.3), was given at the creation of the file.

After the invalidation of EF_{KEY_OP}, a function requiring a key from this EF_{KEY_OP} can not be performed anymore.

It shall not be possible to invalidate EF_{KEY_MAN}.

Input:	None or cryptogram.
	Refer to subclause 9.2.20 for a description of the input data.
Output:	None.

8.21 REHABILITATE

This function allows the rehabilitation of the invalidated Current File. After a REHABILITATE function, the bit 1 of byte 12 (file status, see subclause 9.2.1) shall be set to one for the concerned file. The AC which shall be fulfilled are the AC for the Current File, for the REHABILITATE function.

Input:	None or cryptogram.
	Refer to subclause 9.2.21 for a description of the input data.
Output:	None.

8.22 INTERNAL AUTHENTICATION

This function allows the external world to authenticate an application in the card. For this purpose, the card has to calculate and send out a cryptogram, using the relevant keyfile, and a challenge and the key both given to the card as an input parameter of the INTERNAL AUTHENTICATION function. The relevant keyfile means:

a) EF_{KEY_MAN} if the current selected file is a DF or a keyfile;

b) EF_{KEY_OP} if the current selected file is an EF.

INTERNAL AUTHENTICATION shall not be accepted before CHV1 has been successfully presented, if this is required in the current directory (bit b8 of byte 8, see subclause 9.3.2).

Input: a/ challenge.

b/ key number.

Output: a/ cryptogram.

Refer to subclause 9.2.22 for a description of the input and output data.

NOTE 1: For this function, no ACs are defined.

NOTE 2: Keynumber valid or not for internal authentication, is indicated in the algorithm ID in the relevant keyfile (EF_{KEY_MAN} or EF_{KEY_OP}).

8.23 ASK RANDOM

This function allows the external world to ask the card for a random number. This random number shall be valid at least for the next function carried out. It shall be deleted within the card at least with use of the random.

The ASK RANDOM function shall always be performed once before any function requiring a cryptogram or enciphered data as input parameters (functions performed with AC = PRO).

Input: Length of the random number (L_e parameter).

Refer to subclause 9.2.23 for a description of the input data.

Output: Random number.

Refer to subclause 9.2.23 for a description of the output data.

NOTE: For this function, no ACs are defined.

8.24 GIVE RANDOM

This function allows to submit a random number to the card.

The GIVE RANDOM function shall always be performed once before any function requiring a computation of a cryptogram which has to be send out by the card.

Input: Random number.

Refer to subclause 9.2.24 for a description of the input data.

Output: None.

NOTE: For this function, no ACs are defined.

8.25 EXTERNAL AUTHENTICATION

This function allows the authentication of the external world by the card. (In order to fulfil the access condition AUT for the selected current file). This function shall follow immediately after the ASK RANDOM function. The card then verifies the cryptogram given by the external world to the card, using the relevant keyfile and the key defined as an input parameter. The relevant keyfile means:

a) EF_{KEY_MAN} if the current selected file is a DF or a keyfile;

b) EF_{KEY_OP} if the current selected file is an EF.

The access condition AUT shall be fulfilled if the cryptogram verification by the card is positive.

Input: a/ key number.

b/ cryptogram.

Refer to subclause 9.2.25 for a description of the input data.

Output: None.

NOTE: For this function, no ACs are defined.

8.26 CLOSE APPLICATION

This function gives the ability, before leaving an application, to delete in the card, the recollection of the ACs which were fulfilled for the current directory.

Input: DF ID.

Refer to subclause 9.2.26 for a description of the input data.

Output: None.

NOTE: For this function, no ACs are defined.

8.27 WRITE BINARY

This function allows the writing in the Current EF (which has to be a transparent EF), with a string of bytes. The function can be performed according to the AC defined for this EF for the WRITE function. A write can be considered as an "OR-ing" of the bits already present in the card memory by the bits given in the write command.

The input shall consist of inputs (a and b) except if the AC = PRO (see below).

If AC = PRO, the input shall consist of inputs (a, b, and c). The card authenticates the origin and the contents of the command, by verifying the cryptogram (input c) calculated by the terminal/application from the input-parameters (a, b) and the random number previously given by the card after an ASK RANDOM function. The WRITE function shall not be performed by the card if the verification of the cryptogram is incorrect. The key used for this verification shall be the one indicated in the AC of the Current EF, for the WRITE functions.

Input: a/ relative address (offset) to the beginning of the EF and the length (in bytes) of the string to be written (P1, P2 in the header and L_c).

b/ data to be written.

c/ cryptogram.

Refer to subclause 9.2.27 for a description of the input data.

Output: None.

8.28 WRITE RECORD

This function allows the writing of one record in the Current EF. The EF shall be of the linear or the cyclic type. For a linear file, the record to be written can be the current, the next or the previous one (this refers to the previously performed READ, WRITE, UPDATE or SEEK function). It can also be the first or the last, or the record indicated by its record number. For cyclic files, there is no choice. If the EF is a cyclic type, only the previous record before the last written record (which is the oldest written record) shall be written, using the PREVIOUS mode.

The WRITE can be considered as an "OR-ing" of the bits already present in the card memory with the bits given in the command.

Six modes are defined:

a) CURRENT and ABSOLUTE modes: the record pointer is not affected by this function;

b) NEXT mode: the record pointer is incremented before the WRITE RECORD function is performed. If the record pointer has not been previously set within the selected EF, then WRITE (next) shall write the first record in the

EF and set the record pointer to this record. If the record pointer is already situated at the last record in the EF, WRITE (next) shall not cause the record pointer to be shifted and no record data shall be written;

c) PREVIOUS mode: the record pointer is decremented before the WRITE RECORD function is performed. If the record pointer has not been previously set within the selected EF, then WRITE (previous) shall write the last record in the EF and set the record pointer to this record. If the record pointer is already situated at the first record in the EF, WRITE (previous) shall not cause the record pointer to be shifted, and no record data shall be written (except for a cyclic file);

d) FIRST and LAST modes: the record pointer becomes the first or the last record before the WRITE RECORD function is performed.

The function shall only be performed if the number of bytes to be written is the same as the length of the requested record.

The function can be performed according to the AC defined for the WRITE function for this EF.

If the WRITE function is allowed for an EF, the INCREASE function shall not be valid for this EF (see subclauses 7.1.4 and 9.3).

The input shall consist of inputs (a and b) except if AC = PRO (see below).

If AC = PRO, the input shall consist of inputs a, b and c. The card authenticates the origin and the contents of the command, by verifying the cryptogram (input c) calculated by the terminal/application from the input-parameters a and b and the random number previously given by the card after an ASK RANDOM function. The WRITE function shall not be performed by the card if the verification of the cryptogram is incorrect. The key which is used for this verification shall be the one indicated in the AC of the Current EF, for the WRITE functions.

Input:	a/ indication current/ previous/ next/ first/ last or absolute (record number).
	b/ the data of the record.
	c/ cryptogram.
	Refer to subclause 9.2.28 for a description of the input data.
Output:	None.

8.29 LOCK

This function allows to set to NEV the AC of one or several specific groups of functions with the same AC requirements of the current selected file. This function does not act on the group READ for which the initial AC can not be modified.

The AC for LOCK, defined for the parent file shall be fulfilled.

If AC=PRO the input shall consist of inputs a, b and c. The card shall authenticate the origin and contents of the command by verifying the cryptogram (input c) calculated by the terminal from the input parameters a and b and the random number previously given by the card after an ASK RANDOM function. The LOCK function shall not be performed by the card if the verification of the cryptogram is incorrect. The key used for this verification shall be the one indicated in the AC for the Current Directory for the LOCK function.

Input:	a/ indication of the group to be locked.
	b/ the file ID. The card shall check that this file ID is consistent with the one of the selected file before executing the command.
	c/ cryptogram.
	Refer to subclause 9.2.29 for a description of the input data.
Output:	None.

8.30 DECREASE

This function allows to decrease the contents of the last written record (record number 1), in the current selected cyclic EF. The new value shall then be stored in the oldest record (previous record before the last written record) in the same way as defined for the UPDATE RECORD function.

The IC card shall not perform the DECREASE function if the minimum value is to be exceeded. The minimum value is the value for which all bytes of the record are zero. If the minimum value is to be exceeded, the IC card shall return an indication (given in the status bytes) that the DECREASE function has not been performed.

The AC which shall be fulfilled, are the AC for the current EF. If the DECREASE function is allowed for an EF, the UPDATE function shall not be valid for this EF (see subclauses 7.1.4 and 9.3.1).

The input shall consist only of input (a), and the output shall consist of outputs (a, b), except for the AC = PRO (see below).

If AC = PRO, the input shall consist of inputs a and b. The card authenticates the origin and the contents of the command, by verifying the cryptogram (input b) calculated by the terminal/application from the input-parameter (a) the random number previously given by the card after an ASK RANDOM function. The DECREASE function shall not be performed by the card if the verification of the cryptogram is incorrect. The key which is used for this verification shall be the one indicated in the AC of the Current EF, for the DECREASE function.

Input:	a/ Value to be deducted (in 3 bytes).
	b/ Cryptogram.
	Refer to subclause 9.2.30 for a description of the input data.
Output:	a/ New value.
	b/ Value deducted.
	Refer to subclause 9.2.30 for a description of the output data.

8.31 DECREASE STAMPED

This function has the same functionality as the function DECREASE which is defined in subclause 8.30. In addition the DECREASE STAMPED function adds a cryptogram to the output data. In order to calculate the outgoing cryptogram, a challenge shall be given before to the card, using the GIVE RANDOM function.

The input shall consist only of input (a), and the output shall consist of outputs (a, b, c), except for the AC = PRO (see below).

If AC = PRO, the input shall consist of inputs a and b. The card authenticates the origin and the contents of the command, by verifying the cryptogram (input b) calculated by the terminal/application from the input-parameter (a) the random number previously given by the card after an ASK RANDOM function. The DECREASE function shall not be performed by the card if the verification of the cryptogram is incorrect. The key which is used for this verification shall be the one indicated in the AC of the Current EF, for the DECREASE function.

The relevant key to be used is defined in the AC for DECREASE function.

Input:	a/ Value to be deducted (in 3 bytes).
	b/ Cryptogram.
	Refer to subclause 9.2.31 for a description of the input data.
Output:	a/ New value.
	b/ Value deducted.
	c/ Cryptogram".
	Refer to subclause 9.2.31 for a description of the output data.

8.32 INCREASE

This function allows to increase the contents of the last written record (record number 1), in the current selected cyclic EF. The new value shall then be stored in the oldest record (previous record before the last written record) in the same way as defined for the UPDATE RECORD function.

The IC card shall not perform the INCREASE function if the maximum value is to be exceeded. The maximum value is the value for which all bytes of the record are "FF". If the maximum value is to be exceeded, the IC card shall return an indication (given in the status bytes) that the INCREASE function has not been performed.

The AC which shall be fulfilled, are the AC for the current EF. If the INCREASE function is allowed for an EF, the WRITE function shall not be valid for this EF (see subclauses 7.1.4 and 9.3.1).

The input shall consist only of input (a), and the output shall consist of outputs (a, b), except for the AC = PRO (see below).

If AC = PRO, the input shall consist of inputs a and b. The card authenticates the origin and the contents of the command, by verifying the cryptogram (input b) calculated by the terminal/application from the input-parameter (a) the random number previously given by the card after an ASK RANDOM function. The INCREASE function shall not be performed by the card if the verification of the cryptogram is incorrect. The key which is used for this verification shall be the one indicated in the AC of the Current EF, for the INCREASE function.

Input:	a/ Value to be added (in 3 bytes).
	b/ Cryptogram.
Output:	a/ New value.
	b/ Value added.
	Refer to subclause 9.2.32 for a description of the output data.

8.33 INCREASE STAMPED

This function has the same functionality as the function INCREASE which is defined in subclause 8.32. In addition the INCREASE STAMPED function adds a cryptogram to the output data.

In order to calculate the outgoing cryptogram, a challenge shall be given before to the card, using the GIVE RANDOM function.

The input shall consist only of input a, and the output shall consist of outputs (a, b, c), except for the AC = PRO (see below).

If AC = PRO, the input shall consist of inputs (a and b). The card authenticates the origin and the contents of the command, by verifying the cryptogram (input b) calculated by the terminal/application from the input-parameter (a) the random number previously given by the card after an ASK RANDOM function. The INCREASE STAMPED function shall not be performed by the card if the verification of the given cryptogram is incorrect. The key which is used for this verification shall be the one indicated in the AC of the Current EF, for the INCREASE function.

The relevant key to be used is defined in the AC for INCREASE function.

Input:	a/ Value to be added (in 3 bytes).
	b/ Cryptogram.
	Refer to subclause 9.2.33 for a description of the input data.
Output:	a/ New value.
	b/ Value added.
	c/ Cryptogram".
	Refer to subclause 9.2.33 for a description of the output data.

8.34 LOAD KEY FILE

This function can only be performed on keyfiles (EF_{KEY_MAN} and EF_{KEY_OP}).

It allows the loading of the keys into the currently selected keyfile.

Only the access conditions PRO, CHV/PRO or NEV are applicable for this function.

The LOAD KEY FILE function is linked to the SAGE algorithm TESA-7.

The input parameters consist of (a, b, c, d and e). The card shall authenticate the origin and the contents of the command by verifying the cryptogram (input e) calculated by the terminal /application from the input parameters (a, b, c, d) and the random number previously given by the card using the ASK RANDOM function.

Input parameter (d) will be enciphered by the external world. For deciphering the card shall use secret key as used for checking the cryptogram.

The relevant key to fulfil the AC for LOAD KEY FILE is located in the next upper level EF_{KEY_MAN}, if EF_{KEY_MAN} is not filled yet, otherwise the relevant key is in the addressed EF_{KEY_MAN}.

Input:	a/ EF_{KEY} type (MAN or OP) (P1).
	b/ key number (P2).
	c/ key file version, key length, algorithm ID (in clear).
	d/ enciphered key.
	e/ cryptogram.
	Refer to subclause 9.2.34 for a description of the input data.
Output:	None.

9 Description of the commands

9.1 Mapping principles

See ISO/IEC 7816-4 [15] subclause 5.3 for the description of the message structure for the functions described in clause 8 of the present document.

The general command set description, the Application Specific Command (ASC) set description and the status conditions returned by the card shall be independent from the transmission protocol used. For example, byte or block protocol.

9.1.1 Command APDU

See ISO/IEC 7816-4 [15] subclause 5.3.1.

9.1.2 Response APDU

See ISO/IEC 7816-4 [15] subclause 5.3.3.

9.1.3 Command APDU conventions

See ISO/IEC 7816-4 [15] subclauses 5.3.2 and 5.4.

Lc codes the number of data bytes present in the data field of the command, from 1 to 255 (or 65 535 for a card supporting the use of extended Lc and Le fields). If no data bytes are present, Lc shall not be sent (note that the use of Lc=0 is not defined in ISO/IEC 7816-4) [15].

Le codes the maximum number of data bytes expected in the data field of the response, from 1 to 256 (or 65536 for a card supporting extended Lc and Le fields). When Le = 0, this corresponds to the maximum value of Le, 256 (or 65536). If the requested number of bytes are not available, the card may optionally return the number of bytes that are available, or return no data and an appropriate error response. If the Le field is empty, this corresponds to requesting 0 bytes of data.

The transport of the Application Protocol Data Units (APDU) of a command-response pair by the transmission protocols defined in ISO/IEC 7816-3 is specified in:

a) ISO/IEC 7816-4 [15] annex A for T=0;

b) ISO/IEC 7816-4 [15] annex B , for T=1.

When a command APDU is transported in the T=0 protocol according to ISO/IEC 7816-4 [15] Annex A, when no Lc or Le field is sent, a P3 byte of 0 is sent. This would appear identical to the card as a command APDU with an Le value of 0. In this case the card may either interpret the APDU as having an Le value of 0 or an empty Le field, dependent on the context (e.g. the command sent).

9.2 Coding of the commands

Table 7: Coding of the commands

Command	INS	P1	P2
SELECT	"A4"	** (note 1)	type
STATUS	"F2"	"00"	"00"
CREATE-FILE	"E0"	Initialization value	Type of creation
DELETE-FILE	"E4"	"00"	"00"
EXTEND	"D4"	Initialization value	Type of extension
EXECUTE	"AE"	"00"	"00"
UPDATE BINARY	"D6"	offset	offset
UPDATE RECORD	"DC"	rec.no	mode
CREATE RECORD	"E2"	"00"	"00"
READ BINARY	"B0"	offset	offset
READ BINARY STAMPED	"B4"	offset	offset
READ RECORD	"B2"	rec.no	mode
READ RECORD STAMPED	"B6"	rec.no	mode
SEEK	"A2"	00/offset	type/mode
VERIFY CHV	"20"	"00"	CHV
CHANGE CHV	"24"	"00"	CHV
DISABLE CHV	"26"	"00"	CHV
ENABLE CHV	"28"	"00"	CHV
UNBLOCK CHV	"2C"	"00"	CHV
INVALIDATE	"04"	"00"	"00"
REHABILITATE	"44"	"00"	"00"
INTERNAL AUTHENTICATION	"88"	"00"	"00"
ASK RANDOM	"84"	"00"	"00"
GIVE RANDOM	"86"	"00"	"00"
EXTERNAL AUTHENTICATION	"82"	"00"	"00"
CLOSE APPLICATION	"AC"	ID	ID
WRITE BINARY	"D0	offset	offset
WRITE RECORD	"D2"	rec.no	mode
LOCK	"76"	group	"00"
DECREASE	"30"	"00"	"00"
DECREASE STAMPED	"34"	“01”	"00"
INCREASE	"32"	"00"	"00"
INCREASE STAMPED	"36"	“02”	"00"
LOAD KEY FILE	"D8"	EF_{KEY} type	Key number
GET RESPONSE	"C0"	"00"	"00"
ENVELOPE PUT	"C2"	"00"	"00"

NOTE 1: ** means selection control byte.

NOTE 2: For the commands in this clause, it has to be noted, unless otherwise specified, that all the numbers are in HEX mode except for the L_c/L_e-parameter in the commands, which are in decimal mode for clarity.

NOTE 3: The length "X", which is most likely to be used in the L_c/L_e-parameter and in the responses, denotes the variable length of the input (which can be a challenge) and output parameters of the different algorithms used in the card.

The CLA-bytes AX should be used for telecommunication cards, with X coded according to ISO/IEC 7816-4 [15] subclause 5.4.1. It is highly recommended to support A0.

Unless otherwise specified, data fields are left justified and padded with 1s. All the bytes and bits in the response to commands specified RFU are set to zero.

Where bytes P1 or P2 are listed as "00", other values of these bytes are RFU. In these cases the card shall only accept the value "00" and reject any other values as incorrect.

9.2.1 SELECT

Table 8: Coding of the SELECT command

CLA	As defined in subclause 9.2
INS	"A4"
P1	Selection control, see table 9
P2	Type of selection, see table 10
L_c field	Not present or length of the data field
Data field	If present, according to P1, - file ID - path from MF - path from current DF - application identifier
L_e field	Maximum length of data expected in response

Table 9: Coding of the selection control P1

Bit Assignment of P1			**Meaning of the bits**
b8 b7 b6 b5	**b4 b3**	**b2 b1**	
	Selection	**by File ID**	
RFU filled with 0	0 0	0 0	Select according to file qualifier Data = File ID
	0 0	0 1	Select son DF Data = DF ID
	0 0	1 0	Select EF under Current Directory Data = EF ID
	0 0	1 1	Select parent directory of current DF Data = none
	0 1	0 0	Absolute selection of DF Data = application identifier
	0 1	0 1	RFU
	0 1	1 0	RFU
	0 1	1 1	RFU
	Selection	by path	
	1 0	0 0	Select EF or DF from the MF Data = path (MF non included)
	1 0	0 1	Select EF or DF downward from Current Directory Data = path (current level not included)
	1 0	1 0	RFU
	1 0	1 1	RFU
Any other value			RFU

NOTE: The type(s) of selection supported by the card is given in EF_{ICC}.

When selecting according to file qualifier, the indicated qualifier shall be checked in priority order:

1) immediate children of the current DF;
2) parent DF;
3) immediate children of the parent DF.

If in "SELECT by path from MF" the path is empty, the MF shall be selected.

If in "SELECT by path from current directory" the path is empty, the current directory shall be selected.

Table 10: Type of the selection P2

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
X	X	X	0	X	X	X	X	Non exclusive select
X	X	X	1	X	X	X	X	Exclusive select

For cards which do not support multi-session mechanism P2 = "00" (non exclusive selection).

Table 11: Coding of the data field of SELECT command (in case L_c is present)

Bytes	Description	Length
1 - 2n	File ID or path	2n bytes

Table 12: Coding of the data field of SELECT command in case of absolute selection

Bytes	Description	Length
1 - X	Application identifier	1 - 16 bytes

NOTE: In case of SELECT parent directory, non data is needed.

Table 13: Coding of the SELECT response in case of an MF or DF

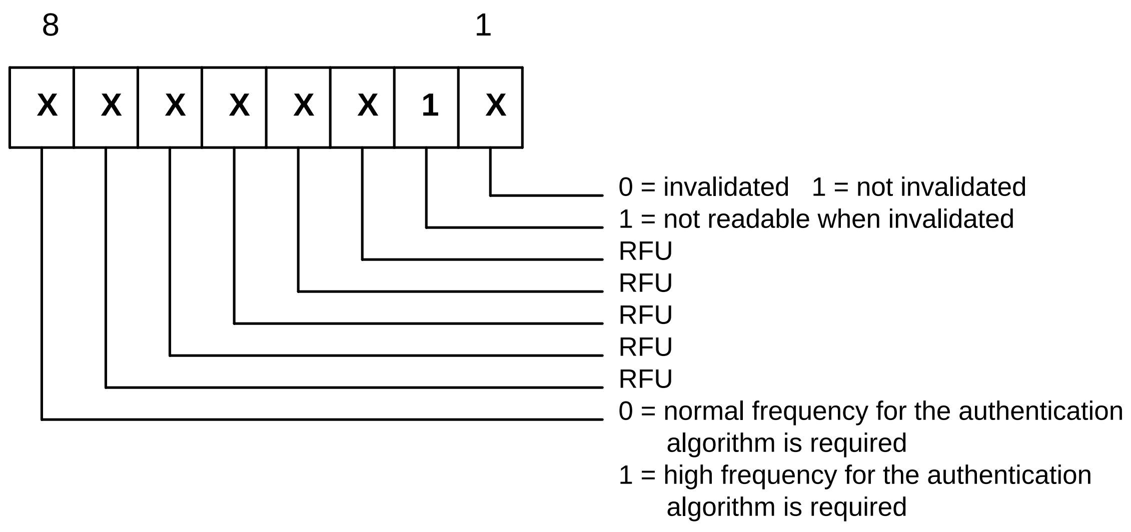
Bytes	Description	Length
1	Tag = "85" (proprietary)	1 byte
2	Length (byte 3 to the end)	1 byte
3 - 4	Total amount of memory of the Current Directory which has not been allocated to any of the DFs or EFs under the Current Directory	2 bytes
5 - 6	File ID	2 bytes
7	Type of file	1 byte
8 - 11	Access conditions (AC) (see subclause 9.3)	4 bytes
12	File status	1byte
13	Length of the following data (byte 14 to the end)	1 byte
14	Current Directory characteristics	1 byte
15	Number of direct son DFs under the Current Directory	1 byte
16	Number of direct son EFs under the Current Directory	1 byte
17	Number of secret codes	1 byte
18	RFU	1 byte
19	CHV1 status	1 byte
20	UNBLOCK CHV1 status	1 byte
21	CHV2 status	1 byte
22	UNBLOCK CHV2 status	1 byte

The presence of bytes 19 to 22 is optional depending on the existence of CHV1 and CHV2.

Byte 7: Type of file:

- "01" Master File (MF);
- "02" Dedicated File (DF);
- "03" Dedicated File (DF) with ASC.

Byte 12: File status:


8
1
X X X X X X 1 X
0 = invalidated 1 = not invalidated
1 = not readable when invalidated
RFU
RFU
RFU
RFU
RFU
0 = normal frequency for the authentication algorithm is required
1 = high frequency for the authentication algorithm is required

Byte 13: Length of the following data (byte 14 to the end).

This byte contains the value:

- "05" if the DF has no relevant EF_{CHV1} or EF_{CHV2};
- "07" if the DF has a relevant EF_{CHV1} and no relevant EF_{CHV2};
- "09" if the DF has both relevant EF_{CHV1} and EF_{CHV2}.

Byte 14: DF characteristics:

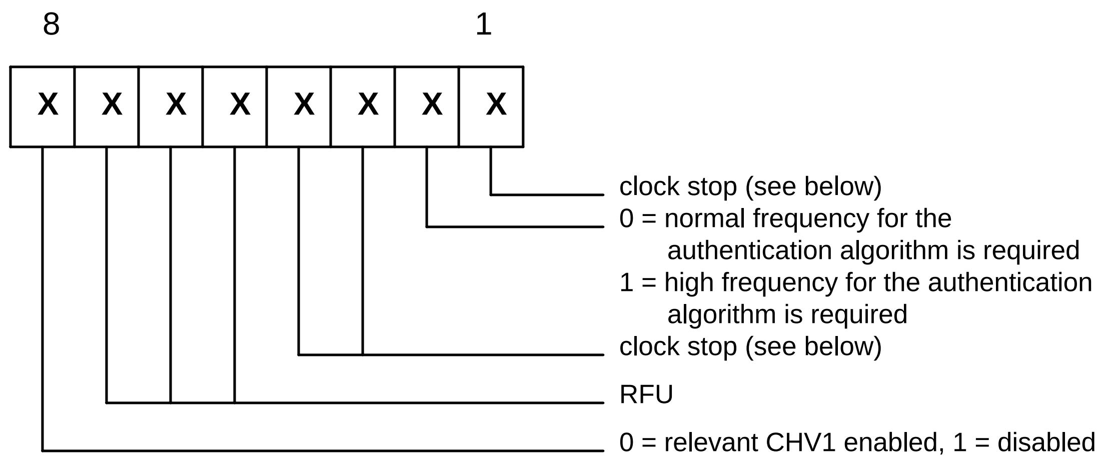

8
1
X X X X X X X X
clock stop (see below)
0 = normal frequency for the authentication algorithm is required
1 = high frequency for the authentication algorithm is required
clock stop (see below)
RFU
0 = relevant CHV1 enabled, 1 = disabled

Table 14: Clockstop

bit1	bit3	bit4	Description
1	0	0	Clockstop allowed, no preferred level
1	1	0	Clockstop allowed, high level preferred
1	0	1	Clockstop allowed, low level preferred
0	0	0	Clockstop not allowed
0	1	0	Clockstop only allowed on high level
0	0	1	Clockstop only allowed on low level

Table 14 gives the coding of the conditions for stopping the clock (note that stopping the clock is an optional feature).

If bit b1 is coded "1" stopping the clock is allowed at high or low level. In this case bits b3 and b4 give information about the preferred level (high or low, resp.) at which the clock may be stopped.

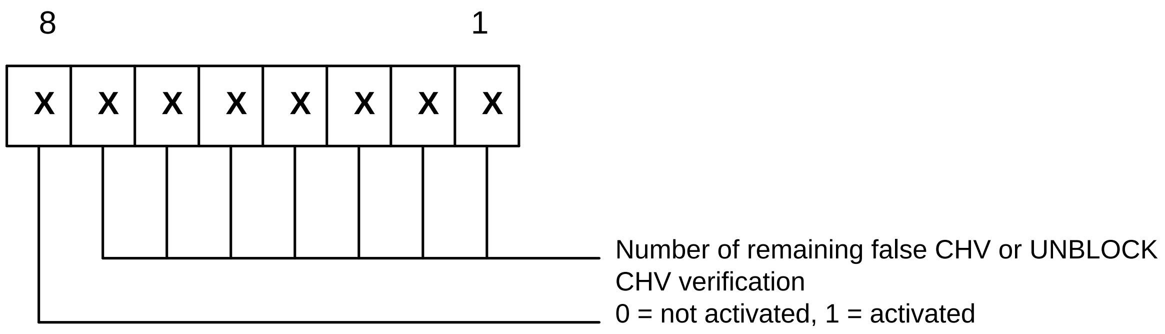
If bit b1 is coded "0", the clock may be stopped only if the mandatory condition in bits b3, b4 (b3=1, i.e. stop at high level or b4=1, i.e. stop at low level) is fulfilled. If all 3 bits are coded "0", then the clock shall not be stopped.

Byte 17: Number of secret codes

This byte consists of the total number of both CHVs and UNBLOCK CHVs. Assuming that a CHV is always associated with an UNBLOCK CHV (both are located in the same file EF_{CHV}), the value of this byte may be:

- "00" if the DF has no relevant EF_{CHV1} or EF_{CHV2};
- "02" if the DF has a relevant EF_{CHV1} and no relevant EF_{CHV2};
- "04" if the DF has both relevant EF_{CHV1} and EF_{CHV2};
- byte 19, 20, 21 and 22: CHV or UNBLOCK CHV status byte.

Each of those bytes is coded as follows:


8
1
X X X X X X X X
Number of remaining false CHV or UNBLOCK CHV verification
0 = not activated, 1 = activated

oth CHV and UNBLOCK CHV are initialized when bit 1 of CHV ACTIVATION byte is set to 1 in an EF_{CHV}.

Table 15: Coding of the SELECT response in case of an EF

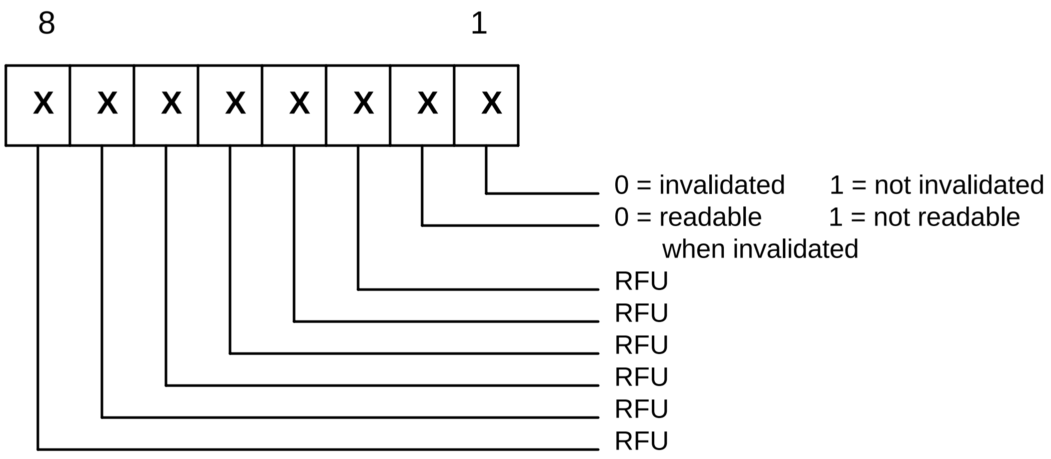
Bytes	Description	Length
1	Tag = "85" (proprietary)	1 byte
2	Length (byte 3 to the end)	1 byte
3 - 4	File size	2 bytes
5 - 6	File ID	2 bytes
7	Type of file	1 byte
8 - 11	Access Conditions (AC) (see subclause 9.3)	4 bytes
12	File status	1 byte
13	Length of the following data (byte 14 to the end)	1 byte
14	Type of EF	1 byte
15	Length of records (if fixed structure)	1 byte

Bytes 3 to 4: File size indicates in case of an EF with linear structure the actual number of records multiplied with their respective length in bytes. For a transparent file the file size indicates the number of bytes allocated for the body of the file.

Byte 7: Type of file:

- "04" Elementary file (EF).

Byte 12: File status:


8
1
X X X X X X X X
0 = invalidated 1 = not invalidated
0 = readable when invalidated 1 = not readable
RFU
RFU
RFU
RFU
RFU
RFU

Byte 13: Length of the following data (byte 14 to the end).

The value of this byte may be:

- "01" if the selected EF is not structured with records (transparent, program or ASC type) or if it is structured with records of variable length (linear type with variable structure);
- "02" if the selected EF is structured with records of fixed length (cyclic or linear fixed structure).

Byte 14: Type of EF:

- "00" Transparent (EF);
- "01" Linear with fixed structure (EF);
- "02" Linear with variable structure (EF);
- "03" Cyclic (EF);
- "04" Program (EF);
- "05" ASC (EF).

Table 16: Coding of the SELECT response in case of an EF_{CHV}

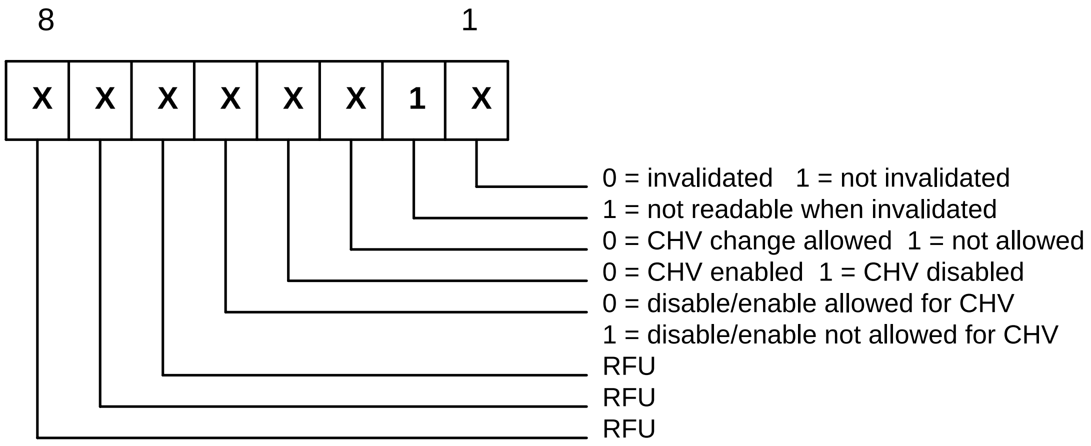
Bytes	Description	Length
1	Tag = "85" (proprietary)	1 byte
2	Length (byte 3 to the end)	1 byte
3 - 4	File size	2 bytes
5 - 6	File ID	2 bytes
7	Type of file	1 byte
8 - 11	Access Conditions (AC) (see subclause 9.3)	4 bytes
12	File status	1 byte
13	Length of the following data (byte 14 to the end)	1 byte
14	Type of EF	1 byte
15	Number of remaining CHV attempts	1 byte
16	EF_{CHV} activation byte	1 byte
17	Way to present the CHV (see 10.1)	1 byte
18	Key number in the relevant EF_{KEY_OP}	1 byte
19	Number of remaining UNBLOCK CHV attempts	1 byte
20	Number of remaining UNBLOCK CHV mechanisms	1 byte

Bytes 3-4: For a transparent file the file size indicates the number of bytes allocated for the body of the file.

Byte 7: Type of file:

- "04" Elementary file (EF).

Byte 12: File status:


8
1
X X X X X X 1 X
0 = invalidated 1 = not invalidated
1 = not readable when invalidated
0 = CHV change allowed 1 = not allowed
0 = CHV enabled 1 = CHV disabled
0 = disable/enable allowed for CHV
1 = disable/enable not allowed for CHV
RFU
RFU
RFU

Byte 13: Length of the following data (byte 14 to the end).

The value of this byte is always "07".

Table 17: Coding of the SELECT response in case of a keyfile (EF$_{KEY\_MAN}$ or EF$_{KEY\_OP}$)

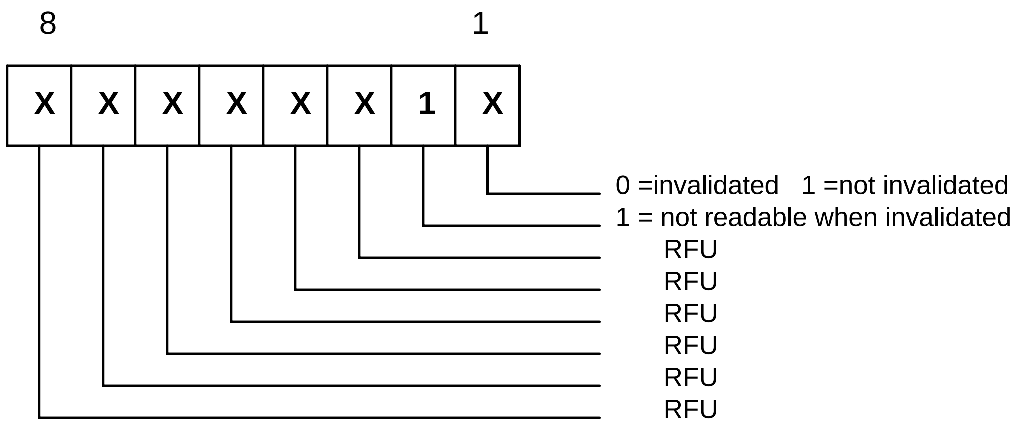
Bytes	Description	Length
1	Tag = "85" (proprietary)	1 byte
2	Length (byte 3 to the end)	1 byte
3 - 4	File size	2 bytes
5 - 6	File ID	2 bytes
7	Type of file	1 byte
8 - 11	Access conditions (AC) (see subclause 9.3)	4 bytes
12	File status	1 byte
13	Length of the following data (byte 14 to the end)	1 byte
14	Type of EF	1 byte
15	Key file version	1 byte
16	Keylength of key 0	1 byte
17	Algorithm ID for key 0	1 byte
18	Keylength of key 1	1 byte
19	Algorithm ID for key 1	1 byte
20	...	1 byte

Bytes 3 to 4: For a transparent file, the file size indicates the number of bytes allocated for the body of the file.

Byte 7: Type of file:

- "04" Elementary file (EF).

Byte 12: File status:


8
1
X X X X X X 1 X
0 =invalidated 1 =not invalidated
1 = not readable when invalidated
RFU
RFU
RFU
RFU
RFU
RFU

Byte 13: Length of the following data (byte 14 to the end).

The value of this byte is variable and depends on the number of keys contained in the respective keyfile (EF_{KEY_MAN} or EF_{KEY_OP}).

9.2.2 STATUS

Table 18: Coding of the STATUS command

CLA	As defined in subclause 9.2
INS	"F2"
P1	"00" (other values are RFU)
P2	"00" (other values RFU)
L_c field	Not present
Data field	Not present
L_e field	Maximum length of data expected in response

Table 19: Coding of the STATUS response

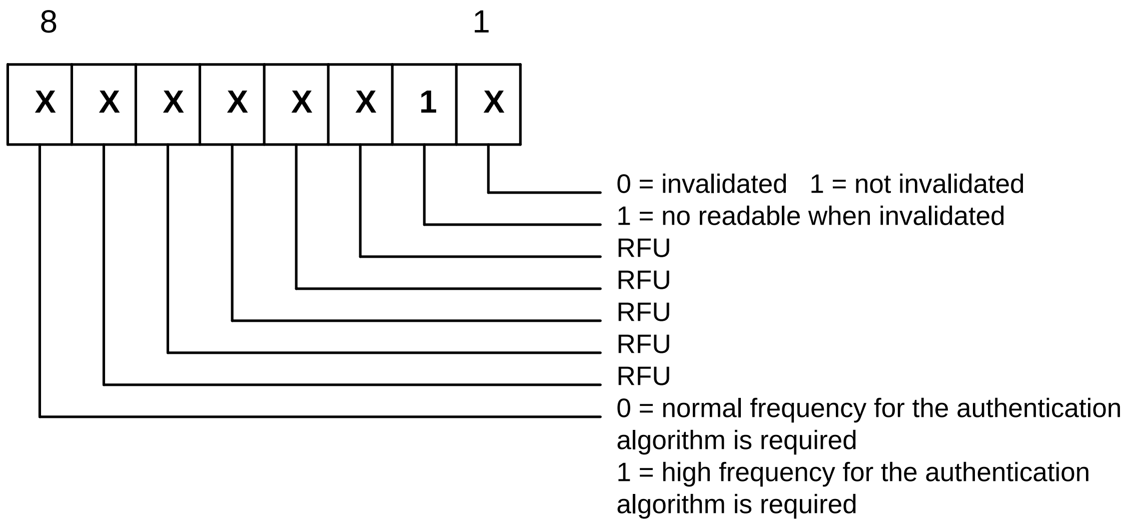
Bytes	Description	Length
1	Tag = "85" (proprietary)	1 byte
2	Length (byte 3 to the end)	1 byte
3 - 4	Total amount of memory of the Current Directory which has not been allocated to any of the directories of EFs under the Current Directory	2 bytes
5 - 6	File ID	2 bytes
7	Type of file	1 byte
8 - 11	Access conditions (AC) (see subclause 9.3)	4 bytes
12	File status	1 byte
13	Length of the following data (byte 14 to the end)	1 byte
14	Current Directory characteristics	1 byte
15	Number of direct son DFs under the Current Directory	1 byte
16	Number of direct son EFs under the Current Directory	1 byte
17	Number of secret codes	1 byte
18	RFU	1 byte
19	CHV1 status	1 byte
20	UNBLOCK CHV1 status	1 byte
21	CHV2 status	1 byte
22	UNBLOCK CHV2 status	1 byte

Byte 7: Type of file:

- "01" Master File (MF);
- "02" Dedicated File (DF);

- "03" Dedicated File (DF) with ASC.

Byte 12: File status:

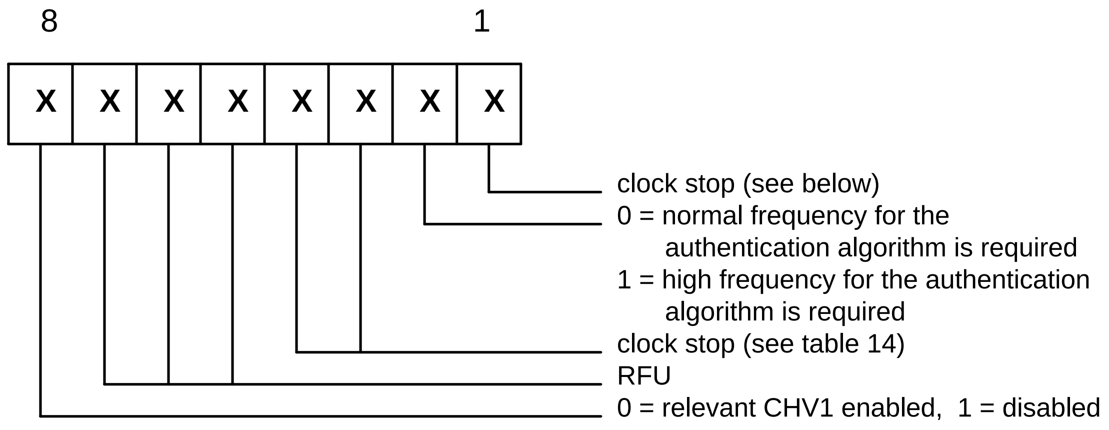

8
1
X X X X X X 1 X
0 = invalidated 1 = not invalidated
1 = no readable when invalidated
RFU
RFU
RFU
RFU
RFU
0 = normal frequency for the authentication algorithm is required
1 = high frequency for the authentication algorithm is required

Byte 13: Length of the following data (byte 14 to the end).

This byte contains the value:

- "05" if the DF has no relevant EF_{CHV1} or EF_{CHV2};
- "07" if the DF has a relevant EF_{CHV1} and no relevant EF_{CHV2};
- "09" if the DF has both relevant EF_{CHV1} and EF_{CHV2}.

Byte 14: DF characteristics:


8
1
X X X X X X X X
clock stop (see below)
0 = normal frequency for the authentication algorithm is required
1 = high frequency for the authentication algorithm is required
clock stop (see table 14)
RFU
0 = relevant CHV1 enabled, 1 = disabled

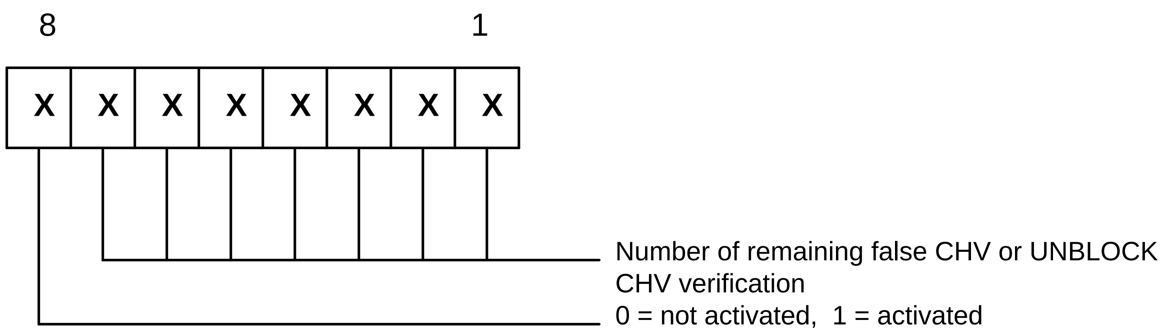
Byte 17: Number of secret codes.

This byte consists of the total number of both CHVs and UNBLOCK CHVs. Assuming that a CHV is always associated with an UNBLOCK CHV (both are located in the same file EF_{CHV}), the value of this byte may be:

- "00" if the DF has no relevant EF_{CHV1} or EF_{CHV2};
- "02" if the DF has a relevant EF_{CHV1} and no relevant EF_{CHV2};
- "04" if the DF has both relevant EF_{CHV1} and EF_{CHV2}.

Byte 19, 20, 21 and 22: CHV or UNBLOCK CHV status byte.

Each of those bytes is coded as follows:


8
1
X X X X X X X X
Number of remaining false CHV or UNBLOCK CHV verification
0 = not activated, 1 = activated

Both CHV and UNBLOCK CHV are initialized when bit 1 of CHV ACTIVATION byte is set to 1 in an EF_{CHV}.

9.2.3 CREATE FILE

Table 20: Coding of the CREATE FILE command

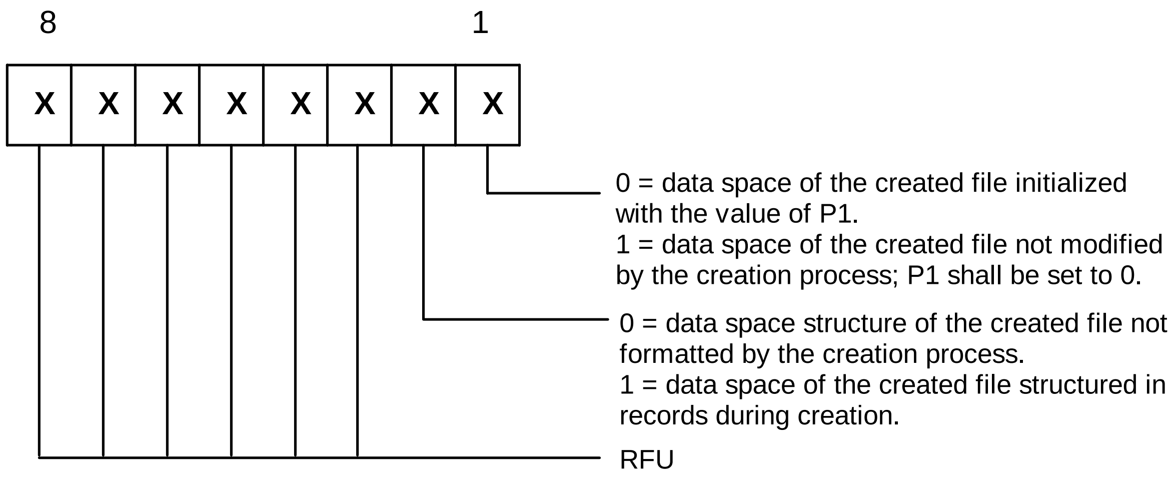
CLA	As defined in subclause 9.2
INS	"E0"
P1	Initialization value
P2	Type of creation
L_c field	Number of data bytes (+ X)
Data field	Data sent to the card (+ cryptogram)
L_e field	Not present

In L_c the (+X) is indicating the length of the cryptogram if AC = PRO.

Coding of P1 and P2:

When creating a DF, P1 and P2 shall be set to 0 (other values are RFU).

When creating an EF, parameter P2 indicates the type of creation:


8
1
X X X X X X X X
0 = data space of the created file initialized with the value of P1.
1 = data space of the created file not modified by the creation process; P1 shall be set to 0.
0 = data space structure of the created file not formatted by the creation process.
1 = data space of the created file structured in records during creation.
RFU

Creation of a transparent EF:

P2	P1	State of the EF after creation
"00"	"00" to "FF"	Data space initialized with the value of P1; whole data space accessible
"01"	"00" (other values are RFU)	Data space not initialized; whole data space accessible.

Other values of P2 are RFU.

Creation of an EF with linear fixed or cyclic structure:

P2	P1	State of the EF after creation
"00"	"00" to "FF"	Data space initialized with the value of P1; no record created
"01"	"00" (other values are RFU)	Data space not initialized; no record created
"02"	"00" to "FF"	Data space initialized with the value of P1; EF filled with records
"03"	"00" (other values are RFU)	Data space not initialized; EF filled with records

Other values of P2 are RFU.

Creation of an EF with linear variable structure:

P2	P1	State of the EF after creation
"00"	"00" to "FF"	Data space initialized with the value of P1; no record created
"01"	"00" (other values are RFU)	Data space not initialized; no record created

Other values of P2 are RFU.

Table 21: Coding of the data field of the CREATE FILE command (in case of creation of a DF)

Bytes	Description	Length
1	Tag "85" (proprietary)	1 byte
2	Length (byte 3 to the end)	1 byte
3 - 4	Amount of memory to be allocated to the DF	2 bytes
5 - 6	File ID	2 bytes
7	Type of file	1 byte
8 - 11	Access conditions (AC) (see subclause 9.3)	4 bytes
12	File status (see subclause 9.2.1)	1 byte
13	Length status (see subclause 9.2.1)	1 byte
14 - 16	Key number related to AC (see subclause 9.3)	3 bytes
17	Application identifier length	1 byte
18 - (17 + Y)	Application identifier	Y bytes

Where an application ID is provided, it shall be between 1 and 16 bytes.

If AC = PRO, the cryptogram shall be given starting at byte 18 or (18+Y) with a length of X bytes for a DF containing an application.

Byte 7: Type of file.

"10" Dedicated File (DF).

"11" Dedicated File (DF) with ASC.

Table 22: Coding of the data field of the CREATE FILE command (in case of the creation of an EF)

Bytes	Description	Length
1	Tag "85" (proprietary)	1 byte
2	Length (byte 3 to the end)	1 byte
3 - 4	File size	2 bytes
5 - 6	File ID	2 bytes
7	Type of file	1 byte
8 - 11	Access conditions (AC) (see subclause 9.3)	4 bytes
12	File status (see subclause 9.2.1 for an EF)	1 byte
13	Length of the following data (byte 14 to the end)	1 byte
14 - 16	Key number related to AC (see subclause 9.3)	3 bytes
17	Length of records (if fixed structure)	1 byte

If AC = PRO, the cryptogram shall be given starting at byte 17 or 18 with a length of X bytes.

In the case of an EF_{CHV}, it shall be filled by using an UPDATE command. The operating system shall manage the fact that some data are not valid before this UPDATE command.

Bytes 3 to 4: File size indicates the number of bytes allocated for the body of the file. In the case of an EF with linear structure the maximum number of records multiplied with their respective length in bytes.

Byte 7: Type of file.

"00"	Transparent (EF);
"01"	Linear with fixed structure (EF);
"02"	Linear with variable structure (EF);
"03"	Cyclic (EF);
"04"	Program (EF);
"05"	ASC (EF).

9.2.4 DELETE FILE

Table 23: Coding of the DELETE FILE command

CLA	As defined in subclause 9.2
INS	"E4"
P1	"00" (other values are RFU)
P2	"00" (other values are RFU)
L_c field	Number of data bytes (+ X)
Data field	Data sent to the card
L_e field	Not present

In L_c the (+X) is indicating the length of the cryptogram if AC = PRO.

Table 24: Coding of the data field of the DELETE FILE command

Bytes	Description	Length
1 - 2	File ID	2 bytes
3 - X+2	Cryptogram (if AC = PRO)	X bytes

9.2.5 EXTEND

Table 25: Coding of the EXTEND command

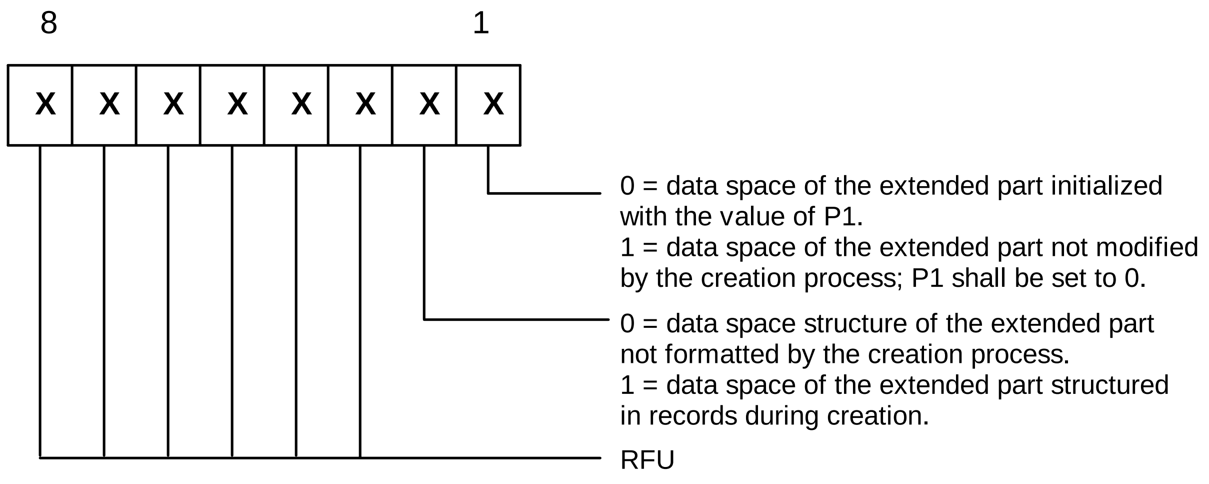
CLA	As defined in subclause 9.2
INS	"D4"
P1	Initialization value
P2	Type of extension
L_c field	Number of data bytes (+ X)
Data field	Data sent to the card
L_e field	Not present

In L_c the (+X) is indicating the length of the cryptogram if AC = PRO.

Coding of P1 and P2:

When extending a DF, P1 and P2 shall be set to 0 (other values are RFU).

When extending an EF, parameter P2 indicates the type of creation:


8
1
X X X X X X X X
0 = data space of the extended part initialized with the value of P1.
1 = data space of the extended part not modified by the creation process; P1 shall be set to 0.
0 = data space structure of the extended part not formatted by the creation process.
1 = data space of the extended part structured in records during creation.
RFU

Extension of a transparent EF:

P2	P1	State of the EF after creation
"00"	"00" to "FF"	Additional data space initialized with the value of P1; whole data space accessible
"01"	"00" (other values are RFU)	Additional data space not initialized; whole data space accessible.

Other values of P2 are RFU.

Extension of an EF with linear fixed structure:

P2	P1	State of the EF after creation
"00"	"00" to "FF"	Additional data space initialized with the value of P1; no record created
"01"	"00" (other values are RFU)	Additional data space not initialized; no record created
"02"	"00" to "FF"	Additional data space initialized with the value of P1; EF filled with records (only if all the initial data space was formatted)
"03"	"00" (other values are RFU)	Additional data space not initialized; EF filled with records (only if all the initial data space was formatted)

Other values of P2 are RFU.

Extension of an EF with linear variable structure:

P2	P1	State of the EF after creation
"00"	"00" to "FF"	Additional data space initialized with the value of P1; no record created
"01"	"00" (other values are RFU)	Additional data space not initialized; no record created

Other values of P2 are RFU.

Table 26: Coding of the data field of the EXTEND command

Bytes	Description	Length
1 - 2	File ID	2 bytes
3	Number of bytes to extend	1 byte
4 - 3+X	Cryptogram (if AC = PRO)	X bytes

9.2.6 EXECUTE

Table 27: Coding of the EXECUTE command

CLA	As defined in subclause 9.2
INS	"AE"
P1	"00" (other values are RFU)
P2	"00" (other values are RFU)
L_c field	Number of data bytes (+ X)
Data field	Data sent to the card (+ cryptogram)
L_e field	If present, maximum length of data expected in response

In L_c the (+X) is indicating the length of the cryptogram if AC = PRO.

This is a command which can have command data and/or expected response data (the data length is application dependent and can also be zero).

9.2.7 UPDATE BINARY

Table 28: Coding of the UPDATE BINARY command

CLA	As defined in subclause 9.2
INS	"D6"
P1	Offset
P2	Offset
L_c field	Length of the subsequent data field (+ X)
Data field	Data to be updated (+ cryptogram)
L_e field	Not present

In L_c the (+X) is indicating the length of the cryptogram if AC = PRO.

Offset is coded in 2 bytes, right justified, i.e., "00 00" means the 1st byte of the EF, "00 01" means the 2nd byte etc.

Where the command is allowed, but the data field does not contain any data to be updated in the current EF, no data shall be updated and a success response shall be returned.

9.2.8 UPDATE RECORD

Table 29: Coding of the UPDATE RECORD command

CLA	As defined in subclause 9.2
INS	"DC"
P1	Record no.
P2	Mode
L_c field	Length of the subsequent data field (+ X)
Data field	Data to be updated (+ cryptogram)
L_e field	Not present

In L_c the (+X) is indicating the length of the subsequent cryptogram if AC = PRO.

The mode in parameter P2 indicates:

- "00" first record;
- "01" last record;
- "02" next record;
- "03" previous record;
- "04" the record no. in P1 (absolute mode) and current mode (P1 = "00" denotes the current record).

9.2.9 CREATE RECORD

Table 30: Coding of the CREATE RECORD command

CLA	As defined in subclause 9.2
INS	"E2"
P1	"00" (other values are RFU)
P2	"00" (other values are RFU)
L_c field	Length of the subsequent data field (+ X)
Data field	Data sent to the card (+ cryptogram)
L_e field	Not present

In L_c the (+X) is indicating the length of the subsequent cryptogram if AC = PRO.

Table 31: Coding of the data field of the CREATE RECORD command

Bytes	Description	Length
1 - n	Record contents	n bytes
n+1, n+X	Cryptogram (if AC = PRO)	X bytes

9.2.10 READ BINARY

Table 32: Coding of the READ BINARY command

CLA	As defined in subclause 9.2
INS	"B0"
P1	Offset
P2	Offset
L_c field	Not present
Data field	Not present
L_e field	Maximum number of bytes to be read

Offset is coded in 2 bytes, right justified, i.e., "00 00" means the 1st byte of the EF, "00 01" means the 2nd byte etc.

Table 33: Coding of the READ BINARY response

Bytes	Description	Length
1 - n	Data to be read	n bytes

9.2.11 READ BINARY STAMPED

Table 34: Coding of the READ BINARY STAMPED command

CLA	As defined in subclause 9.2
INS	"B4"
P1	Offset
P2	Offset
L_c field	Not present
Data field	Not present
L_e field	Maximum number of bytes to be read (+ X)

In L_e the (+X) is indicating the length of the subsequent cryptogram.

Offset is coded in 2 bytes, right justified, i.e., "00 00" means the 1st byte of the EF, "00 01" means the 2nd byte etc.

Table 35: Coding of the READ BINARY STAMPED response

Bytes	Description	Length
1 - n	Data to be read	n bytes
n+1, n+X	Cryptogram	X bytes

9.2.12 READ RECORD

Table 36: Coding of the READ RECORD command

CLA	As defined in subclause 9.2
INS	"B2"
P1	Record no.
P2	Mode
L_c field	Not present
Data field	Not present
L_e field	Maximum number of bytes to be read

If the L_e field consist of "00", then all the records until the end of the file should be read.

The mode in parameter P2 indicates:

- "00" first record;
- "01" last record;
- "02" next record;
- "03" previous record;
- "04" the record no. in P1 (absolute mode) and current mode (P1 = "00" denotes the current record).

Table 37: Coding of the READ RECORD response

Bytes	Description	Length
1 - n	The data of the record	n bytes

9.2.13 READ RECORD STAMPED

Table 38: Coding of the READ RECORD STAMPED command

CLA	As defined in subclause 9.2
INS	"B6"
P1	Record no.
P2	Mode
L_c field	Not present
Data field	Not present
L_e field	Maximum number of bytes to be read (+X)

In L_e the (+X) is indicating the length of the subsequent cryptogram.

If the L_e field consist of "00", then all the records until the end of the file should be read.

The mode in parameter P2 indicates:

- "00" first record;
- "01" last record;

- "02" next record;
- "03" previous record;
- "04" the record no. in P1 (absolute mode) and current mode (P1 = "00" denotes the current record).

Table 39: Coding of the READ RECORD STAMPED response

Bytes	Description	Length
1 - n	The data of the record	n bytes
n+1, n+X	Cryptogram	X bytes

9.2.14 SEEK

Table 40: Coding of the SEEK command

CLA	As defined in subclause 9.2
INS	"A2"
P1	Offset
P2	Type/Mode
L_c field	Length of the subsequent data field
Data field	Data sent to the card
L_e field	Not present or 1 byte in case of type = 2

The parameter P1 indicates the offset (in bytes) inside the record:

- "00" no offset.

The type in parameter P2 indicates:

- X X X X 0 0 0 0 from the beginning forward;
- X X X X 0 0 0 1 from the end backward;
- X X X X 0 0 1 0 from the next location forward;
- X X X X 0 0 1 1 from the previous location backward;
- 0 0 0 0 X X X X type 1, no response data;
- 0 0 0 1 X X X X type 2, returns as response data the absolute number of the record found by the SEEK command.

Table 41: Coding of the data field of the SEEK command

Bytes	Description	Length
1 - n	Pattern	n bytes

Table 42: Coding of the SEEK response in case of type = 2

Bytes	Description	Length
1	Absolute record number found by the SEEK command	1 byte

9.2.15 VERIFY CHV

Table 43: Coding of the VERIFY CHV command

CLA	As defined in subclause 9.2
INS	"20"
P1	"00" (other values are RFU)
P2	CHV
L_c field	Number of data bytes
Data field	Data sent to the card
L_e field	Not present

CHV in parameter P2 indicates: "01" = CHV1, "02" = CHV2.

Table 44: Coding of the data field of the VERIFY CHV command

Bytes	Description	Length
1 - 8	CHV	8 bytes

9.2.16 CHANGE CHV

Table 45: Coding of the CHANGE CHV command

CLA	As defined in subclause 9.2
INS	"24"
P1	"00" (other values are RFU)
P2	CHV
L_c field	Number of data bytes
Data field	Data sent to the card
L_e field	Not present

CHV in parameter P2 indicates: "01" = CHV1, "02" = CHV2.

Table 46: Coding of the data field of the CHANGE CHV command

Bytes	Description	Length
1 - 8	Old CHV	8 bytes
9 - 16	New CHV	8 bytes

9.2.17 DISABLE CHV

Table 47: Coding of the DISABLE CHV command

CLA	As defined in subclause 9.2
INS	"26"
P1	"00" (other values are RFU)
P2	CHV
L_c field	Number of data bytes
Data field	Data sent to the card
L_e field	Not present

CHV in parameter P2 indicates: "01" = CHV1, "02" = CHV2.

Table 48: Coding of the data field of the DISABLE CHV command

Bytes	Description	Length
1 - 8	CHV	8 bytes

9.2.18 ENABLE CHV

Table 49: Coding of the ENABLE CHV command

CLA	As defined in subclause 9.2
INS	"28"
P1	"00" (other values are RFU)
P2	CHV
L_c field	Number of data bytes
Data field	Data sent to the card
L_e field	Not present

CHV in parameter P2 indicates: "01" = CHV1, "02" = CHV2.

Table 50: Coding of the data field of the ENABLE CHV command

Bytes	Description	Length
1 - 8	CHV	8 bytes

9.2.19 UNBLOCK CHV

Table 51: Coding of the UNBLOCK CHV command

CLA	As defined in subclause 9.2
INS	"2C"
P1	"00" (other values are RFU)
P2	CHV
L_c field	Number of data bytes
Data field	Data sent to the card
L_e field	Not present

CHV in parameter P2 indicates: "00"/"01" = CHV1, "02" = CHV2.

Table 52: Coding of the data field of the UNBLOCK CHV command

Bytes	Description	Length
1 - 8	UNBLOCK CHV	8 bytes
9 - 16	New CHV	8 bytes

9.2.20 INVALIDATE

Table 53: Coding of the INVALIDATE command

CLA	As defined in subclause 9.2
INS	"04"
P1	"00" (other values are RFU)
P2	"00" (other values are RFU)
L_c field	Not present or (+ X)
Data field	Not present or cryptogram
L_e field	Not present

In L_c the (+X) is indicating the length of the cryptogram if AC = PRO.

Table 54: Coding of the data field of the INVALIDATE command

Bytes	Description	Length
1 - X	Cryptogram (if AC = PRO)	X bytes

9.2.21 REHABILITATE

Table 55: Coding of the REHABILITATE command

CLA	As defined in subclause 9.2
INS	"44"
P1	"00" (other values are RFU)
P2	"00" (other values are RFU)
L_c field	Not present or (+ X)
Data field	Not present or cryptogram
L_e field	Not present

In L_c the (+X) is indicating the length of the cryptogram if AC = PRO.

Table 56: Coding of the data field of the REHABILITATE command

Bytes	Description	Length
1 - X	Cryptogram (if AC = PRO)	X bytes

9.2.22 INTERNAL AUTHENTICATION

Table 57: Coding of the INTERNAL AUTHENTICATION command

CLA	As defined in subclause 9.2
INS	"88"
P1	"00" (other values are RFU)
P2	Key number (0 - 15) (note)
L_c field	Number of data bytes
Data field	Data sent to the card
L_e field	Maximum length of data expected in response
NOTE: Key numbers valid for INTERNAL AUTHENTICATION are indicated in the relevant Algorithm ID byte, using bit 8, in the relevant keyfile (EF_{KEY_MAN} or EF_{KEY_OP}).	

Table 58: Coding of the data field of the INTERNAL AUTHENTICATION command

Bytes	Description	Length
1 - X	Challenge	X bytes

Table 59: Coding of the INTERNAL AUTHENTICATION response

Bytes	Description	Length
1 - X	Cryptogram	X bytes

9.2.23 ASK RANDOM

Table 60: Coding of the ASK RANDOM command

CLA	As defined in subclause 9.2
INS	"84"
P1	"00" (other values are RFU)
P2	"00" (other values are RFU)
L_c field	Not present
Data field	Not present
L_e field	Maximum length of data expected in response

Table 61: Coding of the ASK RANDOM response

Bytes	Description	Length
1 - X	Random number	X bytes

9.2.24 GIVE RANDOM

Table 62: Coding of the GIVE RANDOM command

CLA	As defined in subclause 9.2
INS	"86"
P1	"00" (other values are RFU)
P2	"00" (other values are RFU)
L_c field	Number of data bytes
Data field	Data sent to the card
L_e field	Not present

Table 63: Coding of the data field of the GIVE RANDOM command

Bytes	Description	Length
1 - X	Random number	X bytes

9.2.25 EXTERNAL AUTHENTICATION

Table 64: Coding of the EXTERNAL AUTHENTICATION command

CLA	As defined in subclause 9.2
INS	"82"
P1	"00" (other values are RFU)
P2	"00" (other values are RFU)
L_c field	Number of data bytes (+ X)
Data field	Data sent to the card
L_e field	Not present

Table 65: Coding of the data field of the EXTERNAL AUTHENTICATION command

Bytes	Description	Length
1	Key number	1 byte
2 - 1 + X	Cryptogram	X bytes

9.2.26 CLOSE APPLICATION

Table 66: Coding of the CLOSE APPLICATION command

CLA	As defined in subclause 9.2
INS	"AC"
P1	File ID
P2	File ID
L_c field	Not present
Data field	Not present
L_e field	Not present

File ID is coded in 2 bytes, right justified.

9.2.27 WRITE BINARY

Table 67: Coding of the WRITE BINARY command

CLA	As defined in subclause 9.2
INS	"D0"
P1	Offset
P2	Offset
L_c field	Length of the subsequent data field (+ X)
Data field	Data to be written (+ cryptogram)
L_e field	Not present

In L_c the (+X) is indicating the length of the subsequent cryptogram if AC = PRO.

Offset is coded in 2 bytes, right justified, i.e., "00 00" means the 1st byte of the EF, "00 01" means the 2nd byte etc.

Where the command is allowed, but the data field does not contain any data to be updated in the current EF, no data shall be updated and a success response shall be returned.

Table 68: Coding of the data field of the WRITE BINARY command

Bytes	Description	Length
1 - n	Data to be written	Length
n + 1, n + X	Cryptogram (if AC = PRO)	X bytes

9.2.28 WRITE RECORD

Table 69: Coding of the WRITE RECORD command

CLA	As defined in subclause 9.2
INS	"D2"
P1	Record no.
P2	Mode
L_c field	Length of the subsequent data field (+ X)
Data field	Data to be written (+ cryptogram)
L_e field	Not present

In L_c the (+X) is indicating the length of the subsequent cryptogram if AC = PRO.

The mode in parameter P2 indicates:

- "00" first record;

- "01" last record;
- "02" next record;
- "03" previous record;
- "04" the record no. in P1 (absolute mode) and current mode (P1 = "00" denotes the current record).

Table 70: Coding of the data field of the WRITE RECORD command

Bytes	Description	Length
1 - n	Data to be written	n bytes
n + 1, n + X	Cryptogram (if AC = PRO)	X bytes

9.2.29 LOCK

Table 71: Coding of the LOCK command

CLA	As defined in subclause 9.2
INS	"76"
P1	Group
P2	"00" (other values are RFU)
L_c field	Number of data bytes (+ X)
Data field	Data sent to the card (+ cryptogram)
L_e field	Not present

In L_c the (+X) is indicating the length of the subsequent cryptogram if AC = PRO.

P1 contains an indication of the group of functions for which the AC should be locked (set to NEV). The contents of P1 is as follows:

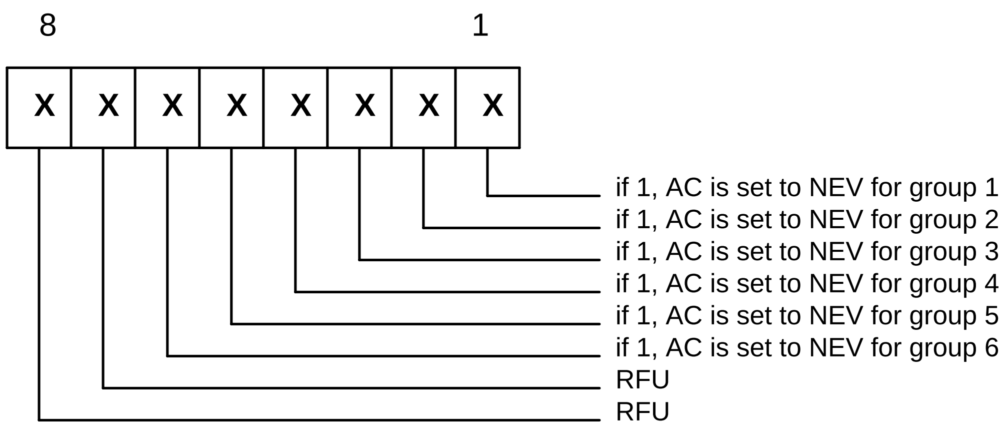

8
1
X X X X X X X X
if 1, AC is set to NEV for group 1
if 1, AC is set to NEV for group 2
if 1, AC is set to NEV for group 3
if 1, AC is set to NEV for group 4
if 1, AC is set to NEV for group 5
if 1, AC is set to NEV for group 6
RFU
RFU

Table 72: Group of functions with same AC requirements

Group	File	Function
1	EF	READ/SEEK (The LOCK function has no effect and the AC remains unchanged)
	EF_{KEY}	LOAD KEY FILE
	MF/DF	LOCK
2	EF	UPDATE or DECREASE
	EF_{KEY}	UPDATE
	MF/DF	RFU (not used)
3	EF	WRITE or INCREASE
	EF_{KEY}	RFU (not used)
	MF/DF	DELETE FILE
4	EF	CREATE RECORD/EXECUTE
	EF_{KEY}	RFU (not used)
	MF/DF	CREATE FILE/EXTEND
5	EF	REHABILITATE
	EF_{KEY}	REHABILITATE
	MF/DF	REHABILITATE
6	EF	INVALIDATE
	EF_{KEY}	INVALIDATE
	MF/DF	INVALIDATE

Table 73: Coding of the data field of the LOCK command

Bytes	Description	Length
1 - 2	File ID	2 bytes

9.2.30 DECREASE

Table 74: Coding of the DECREASE command

CLA	As defined in subclause 9.2
INS	"30"
P1	"00" (other values are RFU)
P2	"00" (other values are RFU)
L_c field	Number of data bytes (+ X)
Data field	Data sent to the card (+ cryptogram)
L_e field	Maximum length of data expected in response

In L_c the (+X) is indicating the length of the subsequent cryptogram if AC = PRO.

Table 75: Coding of the data field of the DECREASE command

Bytes	Description	Length
1 - 3	Value to be deducted	3 bytes
4 - (3+X)	Cryptogram (if AC = PRO)	X bytes

Table 76: Coding of the DECREASE response

Bytes	Description	Length
1 - Y	New value	Y bytes
(Y+1) - (Y+3)	Value deducted	3 bytes
Y is indicating the length of the record.		

9.2.31 DECREASE STAMPED

Table 77: Coding of the DECREASE STAMPED command

CLA	As defined in subclause 9.2
INS	"34"
P1	"01" (other values are RFU)
P2	"00" (other values are RFU)
L_c field	Number of data bytes (+ X)
Data field	Data sent to the card (+ cryptogram)
L_e field	Maximum length of data expected in response

In L_c the (+X) is indicating the length of the subsequent cryptogram.

Table 78: Coding of the data field of the DECREASE STAMPED command

Bytes	Description	Length
1 - 3	Value to be deducted	3 bytes
4 - (3+X)	Cryptogram (if AC = PRO)	X bytes

Table 79: Coding of the DECREASE STAMPED response

Bytes	Description	Length
1 - Y	New value	Y bytes
(Y+1) - (Y+3)	Value deducted	3 bytes
(Y+4) - (Y+3+X)	Cryptogram"	X bytes

Y is indicating the length of the record.

9.2.32 INCREASE

Table 80: Coding of the INCREASE command

CLA	As defined in subclause 9.2
INS	"32"
P1	"00" (other values are RFU)
P2	"00" (other values are RFU)
$L_{c \text{ field}}$	Number of data bytes (+ X)
Data field	Data sent to the card (+ cryptogram)
L_e field	Maximum length of data expected in response

In L_c the (+X) is indicating the length of the subsequent cryptogram if AC = PRO.

Table 81: Coding of the data field of the INCREASE command

Bytes	Description	Length
1 - 3	Value to be added	3 bytes
4 - (3+X)	Cryptogram (if AC = PRO)	X bytes

Table 82: Coding of the INCREASE response

Bytes	Description	Length
1 - Y	New value	Y bytes
(Y+1) - (Y+3)	Value added	3 bytes

Y is indicating the length of the record.

9.2.33 INCREASE STAMPED

Table 83: Coding of the INCREASE STAMPED command

CLA	As defined in subclause 9.2
INS	"36"
P1	"02" (other values are RFU)
P2	"00" (other values are RFU)
L_c field	Number of data bytes (+ X)
Data field	Data sent to the card (+ cryptogram)
L_e field	Maximum length of data expected in response

In L_c the (+X) is indicating the length of the subsequent cryptogram (If AC=PRO).

Table 84: Coding of the data field of the INCREASE STAMPED command

Bytes	Description	Length
1 - 3	Value to be added	3 bytes
4 - (3+X)	Cryptogram (if AC = PRO)	X bytes

Table 85: Coding of the INCREASE STAMPED response

Bytes	Description	Length
1 - Y	New value	Y bytes
(Y+1) - (Y+3)	Value added	3 bytes
(Y+4) - (Y+3+X)	Cryptogram	X bytes

Y is indicating the length of the record.

9.2.34 LOAD KEY FILE

Table 86: Coding of the LOAD KEY FILE command

CLA	As defined in subclause 9.2
INS	"D8"
P1	EF_{KEY} type
P2	Key number
L_c field	Length of the subsequent data field(+ X)
Data field	Data to be loaded (+ cryptogram)
L_e field	Not present

EF_{KEY} type:

P1 = 00 -> EF_{KEY_MAN};

P1 = 01 -> EF_{KEY_OP};

Table 87: Coding of the data field of the LOAD KEY FILE command

Bytes	Description	Length
1	Key file version (only present in case of key 0)	1 byte
2	Key length	1 byte
3	Algorithm ID	1 byte
4 - 19	Key enciphered	16 bytes
20 - 27	Cryptogram	8 bytes

NOTE: The random number used has a length of 8 bytes.

9.2.35 GET RESPONSE

Table 88: Coding of the GET RESPONSE command

CLA	As defined in subclause 9.2
INS	"C0"
P1	"00" (other values are RFU)
P2	"00" (other values are RFU)
L_c field	Not present
Data field	Not present
L_e field	Maximum length of data expected in response

The GET RESPONSE command is associated with the T=0 transmission protocol. With this protocol it is not possible to send both Lc and Le with a single command APDU, allowing data to be both sent to and received from the card. Where command data is sent to the card, and there is response data to receive, the terminal may send a GET RESPONSE command to obtain the response data from the previous command.

The GET RESPONSE command may follow the SELECT, INCREASE, INCREASE STAMPED, DECREASE, DECREASE STAMPED, SEEK and EXECUTE commands or immediately after the ATR.

The response data depends on the previous commands.

Since the MF is implicitly selected after the ATR, the GET RESPONSE command may be used at this time to obtain the response data for selection of the MF.

If the command GET RESPONSE is executed, it is required that it is executed immediately after the command it is related to (no other command shall come between the command/response pair and the command GET RESPONSE). If the sequence is not respected, the card shall send the status response "technical response with no diagnostic given" as a reaction to the GET RESPONSE.

The GET RESPONSE command may also be used if the card supports the use of extended Lc and Le fields (as defined in ISO/IEC 7816-4 [15]) together with the T=0 protocol. In the case where Le > 256, the GET RESPONSE command may be used to obtain further response bytes up to Le. This command may be sent where the previous command sent P3 = 0, and the card returned 256 data bytes with an indication that further response bytes are available.

9.2.36 ENVELOPE PUT

Table 89: Coding of the ENVELOPE PUT command

CLA	As defined in subclause 9.2
INS	"C2"
P1	"00" (other values are RFU)
P2	"00" (other values are RFU)
L_c field	Number of data bytes
Data field	Data sent to the card
L_e field	Not present

The commands in subclauses 9.2.35 and 9.2.36 shall only be used as a transport service for the commands of the byte protocol (T=0).

The ENVELOPE PUT command is associated with the T=0 transmission protocol. It is only required if the card supports the use of extended Lc and Le fields (as defined in ISO/IEC 7816-4 [15]). It is used for the transmission of command APDUs to the card where Lc > 255. In this case the command APDU is split into segments with lengths < 256 bytes which are transmitted to the card in the command data of successive ENVELOPE PUT commands.

9.3 Access Condition (AC) coding

At creation time, for a given file, bytes 9 to 11 of the data of the CREATE FILE command are used to indicate which AC is related to each group of functions while 3 further bytes are used to indicate which of the 16 possible keys (0 to 15) is related to each group of functions.

For bytes 9 to 11, all the possible AC are coded on 4 bits as defined in table 90.

Table 90

ALW	0	
CHV1	1	
CHV2	2	
PRO	3	
AUT	4	
RFU	5	
CHV1/PRO	6	
CHV2/PRO	7	
CHV1/AUT	8	
CHV2/AUT	9	
RFU	10	
RFU	11	
RFU	12	
RFU	13	
RFU	14	- Reserved for future
NEV	15	extension of the AC.

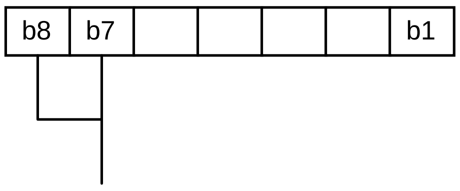
9.3.1 Creation of an EF

At creation time, for a given file, byte 8 of the data of the CREATE FILE command is used with the following purpose:

a) only one of the functions WRITE or INCREASE can be used, depending on the coding of bit 7 of byte 8;

b) only one of the functions UPDATE or DECREASE can be used, depending on the coding of bit 8 of byte 8.

Byte 8 :



Coding : see below

Coding of byte 8, b7 & b8	
b8 b7	**Valid actions**
0 0	UPDATE and WRITE
0 1	UPDATE and INCREASE
1 0	DECREASE and WRITE
1 1	DECREASE and INCREASE

Table 91: Access Condition (AC) coding

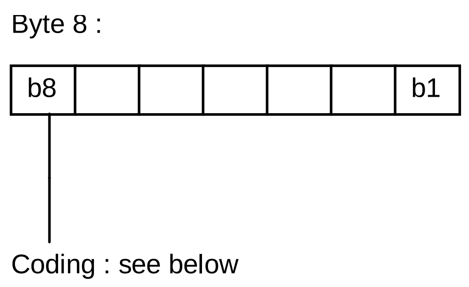
READ/SEEK	UPDATE or DECREASE	byte 9
WRITE or INCREASE	CREATE RECORD/EXECUTE	byte 10
REHABILITATE	INVALIDATE	byte 11

Table 92: Keynumber coding

READ/SEEK	UPDATE or DECREASE	byte 14
WRITE or INCREASE	CREATE RECORD/EXECUTE	byte 15
REHABILITATE	INVALIDATE	byte 16

9.3.2 Creation of a DF

The coding of byte 8 of the data of the CREATE FILE command for a DF shall identify whether CHV1 is required to be verified before INTERNAL AUTHENTICATION is permitted.



Coding of byte 8, b8	
b8	**Valid actions**
0 1	CHV1 has to be verified before INTERNAL AUTHENTICATION No CHV1 verification is required before INTERNAL AUTHENTICATION

Table 93: Access Condition (AC) coding

LOCK	RFU (Not used)	byte 9
DELETE FILE	CREATE /EXTEND FILE	byte 10
REHABILITATE	INVALIDATE	byte 11

Table 94: Keynumber coding

LOCK	RFU (Not used)	byte 14
DELETE FILE	CREATE /EXTEND FILE	byte 15
REHABILITATE	INVALIDATE	byte 16

9.3.3 Creation of a keyfile (EF_{KEY_MAN} or EF_{KEY_OP})

At creation time of a keyfile, byte 8 of the data of the CREATE_FILE command is not used (RFU).

Table 95: Access condition coding

LOAD KEY FILE	UPDATE	byte 9
RFU (Not used)	RFU (Not used)	byte 10
REHABILITATE	INVALIDATE	byte 11

Table 96: Keynumber coding

LOAD KEY FILE	UPDATE	byte 14
RFU (Not used)	RFU (Not used)	byte 15
REHABILITATE	INVALIDATE	byte 16

9.4 Status conditions returned by the card

According to EN 27816-3 [9], [10] two status bytes, SW1 and SW2, are returned after each command.

9.4.1 Security management

Table 97: Status bytes SW1 SW2

SW1	SW2	Error description
98	02	- No CHV and/or key defined - There were no AC to be fulfilled for this DF - Key invalid for function
98	04	- AC no fulfilled - Wrong cryptogram verification - Unsuccessful CHV verification but verify CHV mechanism still possible (number of false consecutive verification < N) - Unsuccessful UNBLOCK CHV verification but UNBLOCK CHV mechanism still possible (number of false consecutive verifications <10)
98	08	- In contradiction with CHV status
98	10	- In contradiction with the invalidation status
98	35	- No ASK RANDOM/GIVE RANDOM before
98	40	- Unsuccessful CHV verification, verify CHV mechanism no longer possible (number of false consecutive verifications ≥ N) - Unsuccessful UNBLOCK CHV verification, UNBLOCK CHV mechanism no longer possible (number of false consecutive verifications ≥ 10) - Limit of successful usage of the UNBLOCK mechanism has been reached
98	50	- Increase/Decrease cannot be performed (Maximum/minimum value reached)

9.4.2 Memory management

Table 98: Status bytes SW1 SW2

SW1	SW2	Error description
92	0X	- Update successful but after using an internal retry routine X times
92	10	- Insufficient memory space available
92	20	- File ID is already existing in this parent file (MF, DF)
92	40	- Memory problem

9.4.3 Referencing management

Table 99: Status bytes SW1 SW2

SW1	SW2	Error description
94	00	- No EF selected as current - EF no selected
94	02	- Out of range (invalid address)
94	04	- File ID not found - Record not found (see note) - Pattern not found
94	08	- Current file is inconsistent with the command
NOTE: The use of 94 04 to indicate record not found is permitted, but it is recommended that for future implementations the status response 94 02 is used if the referenced record is not found (out of range).		

9.4.4 Application independent errors

Table 100: Status bytes SW1 SW2

SW1	SW2	Error description
6E	XX	- Wrong instruction class given in the command
6D	XX	- Unknown instruction code given in the command
6F	XX	- Technical problem with no diagnostic given (command aborted)
6B	XX	- Incorrect parameters P1 or P2 (see NOTE)
67	XX	- Incorrect parameter P3

NOTE: The use of 6B XX to indicate an addressed record being out of range is permitted, but it is recommended that for future implementations the status response 94 02 is used in this case.

9.4.5 Responses to commands which are correctly executed or supporting chaining mechanism

Table 101: Status bytes SW1 SW2

SW1	SW2	Error description
90	00	- Normal ending (ACK) of the command
9F	XX	- Length "XX" of the response data

9.4.6 Commands versus possible status responses

The following tables show for each command the possible status conditions returned (marked *). For each command supported at least one response indicating a failure to complete the command successfully shall be supported. These responses shall be in accordance with subclauses 9.4.1 to 9.4.5.

Table 102: Status responses

	Security Status								**Memory Status**			
Commands	98 02	98 04	98 08	98 10	98 35	98 40	98 50		92 0X	92 10	92 20	92 40
ASK RANDOM												*
CHANGE CHV	*	*	*	*	*	*			*			*
CLOSE APPLICATION												*
CREATE FILE	*	*		*	*				*	*	*	*
CREATE RECORD	*	*		*	*				*	*		*
DECREASE	*	*		*	*		*		*			*
DECREASE STAMPED	*	*		*	*		*		*			*
DELETE FILE	*	*		*	*				*			*
DISABLE CHV	*	*	*	*	*	*			*			*
ENABLE CHV	*	*	*	*	*	*			*			*
ENVELOPE PUT												*
EXECUTE	*	*		*	*							*
EXTEND	*	*		*	*				*	*		*
EXTERNAL AUTHENTICATION	*	*		*	*				*			*
GET RESPONSE												*
GIVE RANDOM												*
INCREASE	*	*		*	*		*		*			*
INCREASE STAMPED	*	*		*	*		*		*			*
INTERNAL AUTHENTICATION	*	*		*								*
INVALIDATE	*	*		*	*				*			*
LOCK	*	*		*	*				*			*
LOAD KEY FILE	*	*		*	*				*			*
READ BINARY		*		*								*
READ BINARY STAMPED	*	*		*	*							*
READ RECORD		*		*								*
READ RECORD STAMPED	*	*		*	*							*
REHABILITATE	*	*		*	*				*			*
SEEK		*		*								*
SELECT												*
STATUS												*
UNBLOCK CHV	*	*	*	*	*	*			*			*
UPDATE BINARY	*	*		*	*				*			*
UPDATE RECORD	*	*		*	*				*			*
VERIFY CHV	*	*	*	*	*	*			*			*
WRITE BINARY	*	*		*	*				*			*
WRITE RECORD	*	*		*	*				*			*

Table 103: Status responses

	Ref. Status					Applic. Indepen. Errors						OK	
Commands	9400	9402	9404	9408		6EXX	6DXX	6FXX	6BXX	67XX		9000	9FXX
ASK RANDOM						*	*	*	*	*		*	
CHANGE CHV						*	*	*	*	*		*	
CLOSE APPLICATION			*	*		*	*	*	*	*		*	
CREATE FILE						*	*	*	*	*		*	
CREATE RECORD	*	*		*		*	*	*	*	*		*	
DECREASE	*			*		*	*	*	*	*		*	*
DECREASE STAMPED	*			*		*	*	*	*	*		*	*
DELETE FILE			*			*	*	*	*	*		*	
DISABLE CHV						*	*	*	*	*		*	
ENABLE CHV						*	*	*	*	*		*	
ENVELOPE PUT						*	*	*	*	*		*	
EXECUTE	*			*		*	*	*	*	*		*	*
EXTEND				*		*	*	*	*	*		*	
EXTERNAL AUTHENTICATION						*	*	*	*	*		*	
GET RESPONSE						*	*	*	*	*		*	*
GIVE RANDOM						*	*	*	*	*		*	
INCREASE	*			*		*	*	*	*	*		*	*
INCREASE STAMPED	*			*		*	*	*	*	*		*	*
INTERNAL AUTHENTICATION						*	*	*	*	*		*	*
INVALIDATE				*		*	*	*	*	*		*	
LOAD KEY FILE	*			*		*	*	*	*	*		*	
LOCK			*			*	*	*	*	*		*	
READ BINARY	*	*1		*		*	*	*	*	*		*	
READ BINARY STAMPED	*	*1		*		*	*	*	*	*		*	
READ RECORD	*	*	*2	*		*	*	*	*	*		*	
READ RECORD STAMPED	*	*	*2	*		*	*	*	*	*		*	
REHABILITATE						*	*	*	*	*		*	
SEEK	*		*	*		*	*	*	*	*		*	*
SELECT			*			*	*	*	*	*		*	*
STATUS						*	*	*	*	*		*	
UNBLOCK CHV						*	*	*	*	*		*	
UPDATE BINARY	*	*1		*		*	*	*	*	*		*	
UPDATE RECORD	*	*	*2	*		*	*	*	*	*		*	
VERIFY CHV						*	*	*	*	*		*	
WRITE BINARY	*	*1		*		*	*	*	*	*		*	
WRITE RECORD	*	*	*2	*		*	*	*	*	*		*	

*1 Use of the status response is permitted, but it is recommended that for future implementations the status response 6B XX is used if the offset given in P1/P2 is out of range.

*2 Use of the status response is permitted, but it is recommended that for future implementations the status response 94 02 is used if the referenced record is not found (out of range).

10 Contents of special EF

The coding is in accordance with ISO 8859-1 [17], except when specified otherwise. The parity bit (bit 8) in ASCII characters set to 0 indicates no parity.

Due to the Access Conditions (AC), the information of the MF or DF belong to the different EF in the way described below. Some of these data are optional (O) and some are mandatory (M). If optional data are not required, optional data at the end of a file may be deleted. If optional data are not at the end of a file, they shall be set to "FF".

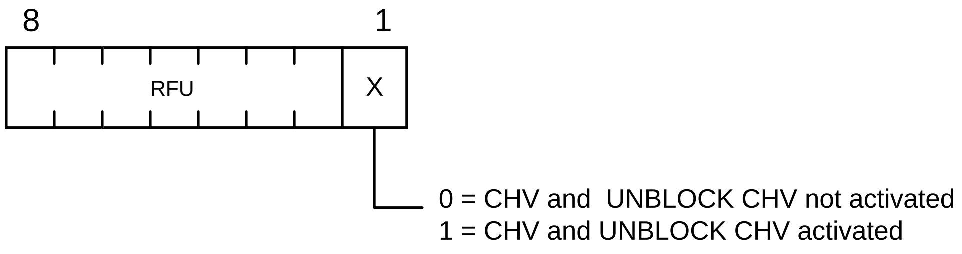
10.1 EF_{CHV}

EF_{CHV} is a transparent file (MF, DF level).

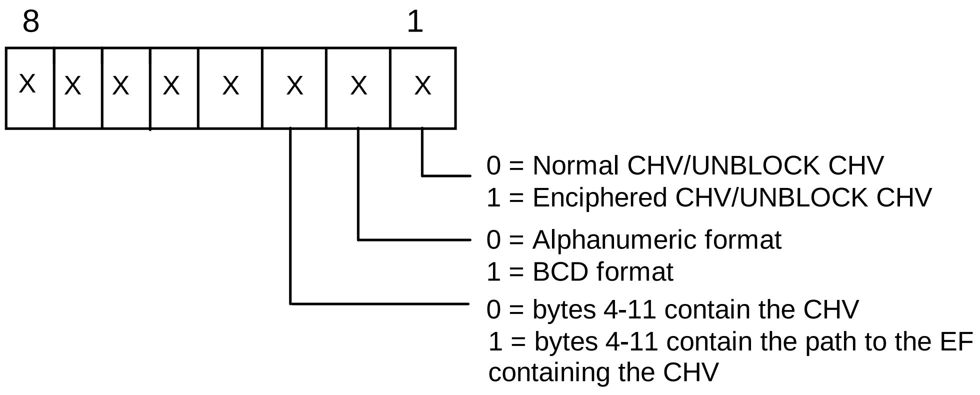
Table 104: EF_{CHV}

File ID: "0000" (CHV1) or "0100" (CHV2)			Optional
AC READ NEV EXECUTE NEV UPDATE Application provider WRITE NEV INVALIDATE Application provider REHABILITATE Application provider			
Bytes	**Description**	**M/O**	**Length**
1	EFCHV activation byte	M	1 byte
2	Way to present the CHV/UNB.CHV	M	1 byte
3	KEY nbr in the relevant EFKEY_OP	M	1 byte
4 - 11	CHV	M	8 bytes
12	CHV attempts Preset value N	M	1 byte
13	Remaining CHV attempt counter	M	1 byte
14 - 21	UNBLOCK CHV	M	8 bytes
22	Remaining UNBLOCK CHV attempt counter	M	1 byte
23	Number of remaining UNB.mech.use	M	1 byte

Coding of byte 1: this byte shall express the fact whether the EF can be used for CHV verification.


8
1
RFU
X
0 = CHV and UNBLOCK CHV not activated
1 = CHV and UNBLOCK CHV activated

Coding of byte 2: Way to present the CHV\UNBLOCK CHV


8
1
X X X X X X X X
0 = Normal CHV/UNBLOCK CHV
1 = Enciphered CHV/UNBLOCK CHV
0 = Alphanumeric format
1 = BCD format
0 = bytes 4-11 contain the CHV
1 = bytes 4-11 contain the path to the EF containing the CHV

Bytes 4 to 11: The CHV is coded according to CCITT Recommendation T.50 [31]/ISO/IEC 646 [13] for telecommunication applications (8 numbers), but using BCD coding the CHV length can be extended to 16 digits. In the case of a CHV larger than 8 bytes (e.g. by biometric means), bytes 4 - 11 indicate the path from the MF to the EF (the MF-ID is not included in the path) containing the CHV (e.g. biometric template).

Byte 23: When byte 23 has the value "FF", this mechanism is allowed to be used an infinite number of times.

10.2 EF_{DIR}

EF_{DIR} is a transparent file. This file is under the responsibility of the issuer at the MF level, but can be under other responsibilities, when situated at a lower level.

Table 105: EF_{DIR} at MF-level

File ID: "2F00"		Optional	
AC: READ issuer/application provider CREATE...EXECUTE NEV UPDATE issuer/application provider WRITE issuer/application provider INVALIDATE issuer/application provider REHABILITATE issuer/application provider			
Bytes	**Description**	**M/O**	**Length**
1	Application identifier tag "4F"	M	1 byte
2	Application identifier length	M	1 byte
3	Application identifier	M	1-16 bytes
	Application label tag "50"	M	1 byte
	Application label length	M	1 byte
	Application label (Verbal description)	M	0-16 bytes
	Path tag "51"	M	1 byte
	Path length	M	1 byte
	Path	M	X bytes
	...		
	Second application information		

10.3 EF_{IC}

EF_{IC} is an optional transparent file at the MF level.

Table 106: EF_{IC}

File ID: "0005"		Optional	
AC: READ ALW CREATE...EXECUTE NEV UPDATE NEV WRITE NEV INVALIDATE NEV REHABILITATE NEV			
Bytes	**Description**	**M/O**	**Length**
1 - 4	IC serial number	M	4 bytes
5 - 8	IC manufacturing references	M	4 bytes
Bytes 1 to 4: IC serial number Contents: IC serial number, binary coded. Purpose: to identify the chip. Bytes 5 to 8: IC manufacturing references Contents: IC manufacturer identifier and fabrication elements, binary coded. Purpose: to identify the chip manufacturer and related information (date and site of fabrication). The AC in table 106, were put to NEVER using the LOCK function, after initializing the contents of the file.			

10.4 EF_{ICC}

EF_{ICC} is a transparent file at the MF level.

Table 107: EF_{ICC}

File ID: "0002"			Mandatory
AC: READ ALW CREATE...EXECUTE NEV UPDATE NEV WRITE NEV INVALIDATE NEV REHABILITATE NEV			
Bytes	**Description**	**M/O**	**Length**
1	Clockstop	M	1 byte
2 - 5	IC card serial number	M	4 bytes
6 - 9	IC card manufacturing references	M	4 bytes
10	Card personalizer ID	M	1 byte
11 - 15	Embedder/IC assembler ID	M	5 bytes
16 - 17	IC identifier	O	2 bytes
18	Card profile	O	1 byte
19	Type of selection	O	1 byte

The AC in table 107, were put to NEVER using the LOCK function, after initializing the contents of the file.

Byte 1: Clockstop

Table 108: Clockstop

	High level	**Low level**	
bit3	**bit2**	**bit1**	
1	0	0	Clockstop allowed, no preferred level
1	1	0	Clockstop allowed, high level preferred
1	0	1	Clockstop allowed, low level preferred
0	0	0	Clockstop not allowed,
0	1	0	Clockstop only allowed on high level
0	0	1	Clockstop only allowed on low level

Table 108 gives the coding of the conditions for stopping the clock (stopping the clock is an optional feature).

If bit b3 is coded "1", stopping the clock is allowed at high or low level. In this case bits b2 and b1 give information about the preferred level (high or low, resp.) at which the clock may be stopped.

If bit b3 is coded "0", the clock may be stopped only if the mandatory condition in bits b2, b1 (b2=1, i.e. stop at high level or b1=1, i.e. stop at low level) is fulfilled. If all 3 bits are coded "0", then the clock shall not be stopped.

Bytes 2 to 5: IC card serial number

Contents: IC card serial number, binary coded.

Purpose: To uniquely identify the card.

Bytes 6 to 9: IC card manufacturing references

Contents: IC card manufacturer ID and fabrication elements, binary coded.

Purpose: To identify the card manufacturer and related information (date and site of fabrication).

Byte 10: Card personalizer ID

Contents: Card personalizer ID as defined by the card issuer.

Purpose: To identify the personalizer of the card.

Bytes 11 to 15: Embedder/IC assembler identifier

Contents: 5 bytes in the form CCEEA. CC = 2 alphabetic country code of the embedder as defined in ISO 3166 [14], EE = 2 alphanumeric characters based on the name of the embedder (there should be a registry at the national level) and A = 1 alphanumeric character for other purposes, e.g. to identify the IC assembler.

Purpose: To identify the organization which combines the IC assembly and the plastic cards.

Bytes 16 to 17: IC identifier

Contents: IC and IC manufacturer identifiers.

Purpose: To identify the IC on the card.

Byte 18: Card Profile

Contents: Profile level (0, 1, 2, 3, 4 or 99). Coded in BCD.

Purpose: For implementation reasons, it is possible to distinguish between five different card profiles (see table 109). It is also possible (for mono-application cards) by the use of profile 99 to indicate that the profile of the card is not covered by the existing five profiles.

For many applications, it shall be essential that the complete system is as simple and inexpensive as possible. Therefore, it shall be necessary to distinguish between several profiles in the card according to the complexity of the following features:

a) channel mechanism / multi-session support;

b) executable EFs;

c) linear variable length structured files;

d) AC = PRO.

A card with a more complex profile can always be used as a card with a less complex profile.

Card profile 0 covers the I-ETS 300 045-1 [11] and the functions for the management phase of the card. However, it does not contain any of the functions described above. It is a multi-application card, but the loading of the applications shall be done at personalization time in a secure environment. The card has very restricted capabilities on network security.

Card profile 1 is more complete with regard to the functions (management, authentication and AC = PRO), but remains a simple card profile.

Card profile 2 contains commands with variable length records. It is a multi-application card that covers most of the security needs, required for telecommunication use, and that provides a better file organization.

Card profile 3 adds one executable files, and the ENVELOPE PUT command to the second one. It is a complete multi-application card with the possibility of adaptation to specific needs of applications, due to executable files.

Card profile 4 contains all the features mentioned above, plus multi-session support.

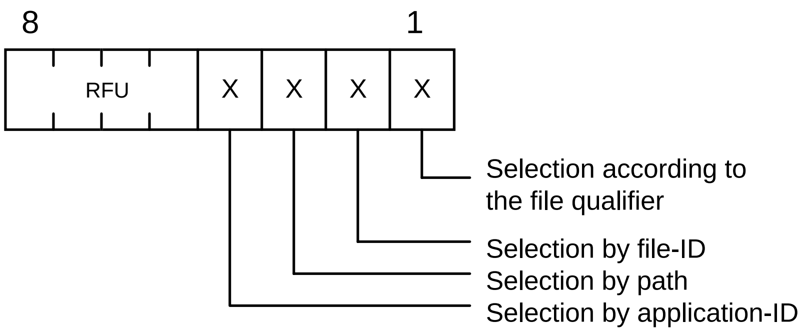
Table 109: Card profiles

Function	0	1	2	3	4
ASK RANDOM		X	X	X	X
CHANGE CHV	X	X	X	X	X
CLOSE APPLICATION					X
CREATE FILE		X	X	X	X
CREATE RECORD		X	X	X	X
DECREASE		X	X	X	X
DECREASE STAMPED		X	X	X	X
DELETE FILE				X	X
DISABLE CHV	X	X	X	X	X
ENABLE CHV	X	X	X	X	X
ENVELOPE PUT (see below)				X	X
EXECUTE				X	X
EXTEND				X	X
EXTERNAL AUTHENTICATION		X	X	X	X
GET RESPONSE (see below)	X	X	X	X	X
GIVE RANDOM		X	X	X	X
INCREASE		X	X	X	X
INCREASE STAMPED		X	X	X	X
INTERNAL AUTHENTICATION	X	X	X	X	X
INVALIDATE		X	X	X	X
LOCK			X	X	X
READ BINARY	X	X	X	X	X
READ BINARY STAMPED		X	X	X	X
READ RECORD	X	X	X	X	X
READ RECORD STAMPED		X	X	X	X
REHABILITATE		X	X	X	X
SEEK	X	X	X	X	X
SELECT	X	X	X	X	X
STATUS	X	X	X	X	X
UNBLOCK CHV	X	X	X	X	X
UPDATE BINARY	X	X	X	X	X
UPDATE RECORD	X	X	X	X	X
VERIFY CHV	X	X	X	X	X
WRITE BINARY		X	X	X	X
WRITE RECORD		X	X	X	X
AC = PRO		X	X	X	X
AC = CHV2				X	X
MULTI SESSION					X
LINEAR VARIABLE STRUCTURE			X	X	X
EF CONTAINING PROGRAM				X	X
CYCLIC FILE MANAGEMENT		X	X	X	X

The ENVELOPE PUT and GET RESPONSE commands shall only be used as a transport service for commands of the byte protocol (T=0). If this protocol is not supported, it is not required that the card supports these commands.

The LOAD KEY FILE is an optional command for all card profiles.

Byte 19: Type of selection


8
1
RFU
X
X
X
X
Selection according to the file qualifier
Selection by file-ID
Selection by path
Selection by application-ID

Contents: X=0 means that this type of selection is not supported by the card. All combinations of these 4 types of selection are possible.

Purpose: To indicate the type of selection supported by the card.

10.5 EF_{ID}

EF_{ID} is a transparent file at MF level. It is initialized by the issuer:

Table 110: EF_{ID}

File ID: "0003"		**Mandatory**	
AC: READ ALW CREATE...EXECUTE NEV UPDATE NEV WRITE NEV INVALIDATE issuer REHABILITATE issuer			
Bytes	**Description**	**M/O**	**Length**
1 - 10	Identification number	M	10 bytes
11 - 13	Date of activation	O	3 bytes
14 - 16	Card expiry date	O	3 bytes
17	Card sequence number	O	1 byte
18 - 19	Country code	O	2 bytes

Bytes 1 to 10: Identification number

Contents: 19 numeric digits, coded according CCITT Recommendation E.118 [30], i.e. MII=89, country code up to 3 numbers, issuer identification number up to 4 digits and individual account number. It is supposed that the individual account number rather identifies the user than the account. Coded in BCD, left justified.

Purpose: identification number.

Bytes 11 to 13: Date of activation of the MF.

Contents: 6 numeric digits, YYMMDD. Coded in BCD.

Purpose: to define the date of the activation.

Bytes 14 to 16: Card expiry date.

Contents: YYMMDD. Coded in BCD.

Purpose: year, month and day.

Byte 17: Card sequence number.

Contents: sequence number. Coded in BCD.

Purpose: the sequence number is needed if more than one card with the same account number is used by a user or if a card is replaced by a new one.

Bytes 18 to 19: Country code

Contents: 3 numeric digits, country code.

Purpose: the country code of the issuer already appears in the identification number. Only present for being compatible with banking cards (see ISO 3166 [14]).

The AC in table 110, were put to NEVER using the LOCK function, after initializing the contents of the file.

10.6 EF_{KEY_MAN}

EF_{KEY_MAN} is a transparent file containing management keys. EF_{KEY_MAN} is mandatory at the MF-level and at each DF-level.

Table 111: EF_{KEY_MAN}

File ID: "0011"		Mandatory
AC: UPDATE NEV LOAD KEY FILE Application provider INVALIDATE Application provider REHABILITATE Application provider		
Bytes	**Description**	**Length**
1	Keyfile version	1byte
2	Keylength of key 0 (X)	1 byte
3	Algorithm ID for key 0	1 byte
4	KEY 0	X bytes
4+X	Keylength of key 1 (Y)	1 byte
5+X	Algorithm ID for key 1	1 byte
6+X	KEY 1	Y bytes
6+X+Y	Keylength of key 2	1 byte
6+X+Y+1	Keylength of key 3 (Z)	1 byte
	...	

A value "00" in the keylength field indicates that there is no more significant information following. As such, it is used as an End-Of-File (EOF) indication.

A value "01" in the keylength field indicates an empty keyfield. As a result the next byte indicates the keylength of the next key.

Algorithm IDs are defined in subclause 7.6.5, and are coded on 7 bits (bits 1 to 7). Bit 8 is reserved for the following purpose:

bit 8=0: the respective key is only valid for internal authentication;

bit 8=1: the respective key is valid for any purpose, except for internal authentication.

In order to prevent corruption of keys inside an EF_{KEY_MAN} during key-loading (e.g. by interrupting the power to the card during the load operation) the operating system may use a flag in EEPROM, indicating that the key-loading was not done correctly. This flag might be tested during the reset of the chip.

10.7 EF_{KEY_OP}

EF_{KEY_OP} is a transparent file containing operational keys. EF_{KEY_OP} is optional at MF- and at DF-level.

EF_{KEY_OP} has exactly the same file structure as EF_{KEY_MAN}.

Table 112: EF_{KEY_OP}

File ID: "0001"		Optional
AC: UPDATE NEV LOAD KEY FILE Application provider INVALIDATE Application provider REHABILITATE Application provider		
Bytes	**Description**	**Length**
1	Keyfile version	1 byte
2	Keylength of key 0 (X)	1 byte
3	Algorithm ID for key 0	1 byte
4	KEY 0	X bytes
4+X	Keylength of key 1 (Y)	1 byte
5+X	Algorithm ID for key 1	1 byte
6+X	KEY 1	Y bytes
6+X+Y	Keylength of key 2	1 byte
6+X+Y+1	Keylength of key 3 (Z)	1 byte
	...	

10.8 EF_{LANG}

EF_{LANG} is a transparent file at the MF level. EF_{LANG} is initialized by the issuer:

Table 113: EF_{LANG}

File ID: "2F05"			Optional
AC: READ ALW CREATE...EXECUTE NEV UPDATE User (note) WRITE issuer INVALIDATE issuer REHABILITATE issuer			
Bytes	**Description**	**M/O**	**Length**
1 - 2	First language preference	O	2 bytes
3 - 4	Second language preference	O	2 bytes
5 - 6	Third language preference	O	2 bytes
7 - 8	Fourth language preference	O	2 bytes
NOTE: User means that the AC conditions to be fulfilled can be: CHV1, PRO or ALW depending on the issuer			

Language preference:

Contents: Maximum four preferences, in order of priority, according to ISO 639 [12]. Coded according to ISO 8859-1 [17].

Purpose: To display messages in an optional language.

10.9 EF_{NAME}

EF_{NAME} is a transparent file at MF level. EF_{NAME} is initialized by the issuer:

Table 114: EF_{NAME}

File ID: "0004"			Optional
AC: READ AUT CREATE...EXECUTE NEV UPDATE issuer WRITE Issuer INVALIDATE issuer REHABILITATE issuer			
Bytes	**Description**	**M/O**	**Length**
1 - 2	Card holder name	O	X bytes

Card holder name:

Contents: Card holder name (coded according to ISO 8859-1 [17]).

Purpose: Personal print outs.

For the read function, only those applications that have fulfilled the AC AUT, have the right to read the name of the card holder.

11 Interoperability of IC cards

11.1 Standardized applications

In order to supply a standardized application in European countries, in a common way has to be defined:

a) the application identifier;

b) the tree structure;

c) the file IDs;

d) the AC related to these files;

e) the contents of EFs.

Specific algorithms and keys can be chosen by the respective application providers. However, the external characteristics of algorithms (format and size of parameters and keys) shall be chosen in a common way.

For authentication or other secure functions using keys, e.g. the following two solutions may appear:

a) service suppliers provide each acquiring institution with security modules;

b) each application provider provides a gateway to a national security centre of the application provider.

11.2 Non-standardized applications

Any non-standardized application in conformance with the general set of commands may be loaded in any card of any country, if required and accepted by the card issuer. In this case the application provider has the possibility to obtain from the card issuer the necessary security elements, including a cryptogram, for the loading of the DF.

12 Security aspects for card manufacturers, application providers and card issuers

This clause gives general requirements for the following stages of the life cycle of a card:

a) chip and card manufacturing process;

b) card preparation;

c) application preparation;

d) usage;

e) termination of use.

For each of these stages, security requirements are defined in the subsequent paragraphs.

12.1 Chip and card manufacturing process

The chip manufacturing process includes:

a) chip semiconductor design and software design (see subclause 12.1.1);

b) chip semiconductor manufacturing (see subclause 12.1.2);

c) chip assembling (see subclause 12.1.3);

d) chip embedding (see subclause 12.1.4).

12.1.1 Chip semiconductor design and software design

The semiconductor design shall prevent the possibility of reading out secured and protected data. The level of security may be increased by a secure memory structure using technologies which allow hiding coherent information in various sections of the masks.

The operating system shall ensure that:

a) no unauthorized access to files is possible;

b) all access conditions have to be fulfilled to get access;

c) loading of new files cannot disturb other files.

12.1.2 Chip semiconductor manufacturing

The manufacturing process shall be in a secure environment, which is protected by access control. The whole manufacturing process shall be logged (number of rejects, destruction of the rejects etc.).

Storage and transport of chips or IC cards shall be physically protected, cryptographic information concerning these chips shall be logically protected.

12.1.3 Chip assembling

The process shall be logged by the manufacturer. Destruction of rejects shall be logged.

12.1.4 Chip embedding

The process shall be logged by the card manufacturer. Destruction of rejects shall be logged.

Annex A (informative): Example of creating an application in the card

The card manufacturer creates the MF, some mandatory elementary files and the keyfile EF_{KEY_MAN} at the MF-level. This EF_{KEY_MAN} will be filled with temporary keys by the card manufacturer.

Then the cards are given to the card issuers, together with the temporary keys. The card issuers can then replace the temporary keys with theirs own set of management keys. This can be done using an UPDATE BINARY function, or a LOAD KEY FILE function (depending on the AC defined for EF_{KEY_MAN}).

Annex B (informative): Examples of certification mechanisms

For some applications, it may be necessary to provide a kind of seal on certain sensitive data. Depending of the nature of the data to be certified, two scenarios are given.

B.1 Certification of external data

Described here is a means by which external data, not necessarily memorized in the card, can be certified.

Figure B.1 describes the commands to be used. A special file EF_{CERT}, where the data to certify are temporarily written, is required. This file has access conditions CHV1 for WRITE and UPDATE.

The file EF_{CERT} is selected and an UPDATE is done with the data and the timestamp. A READ STAMPED done on this file gives to the external world the cryptogram made on the data and the timestamp.

In this case, the input data shall be certified if and only if CHV1 has been fulfilled.

NOTE: If required, EF_{CERT} can be deleted.

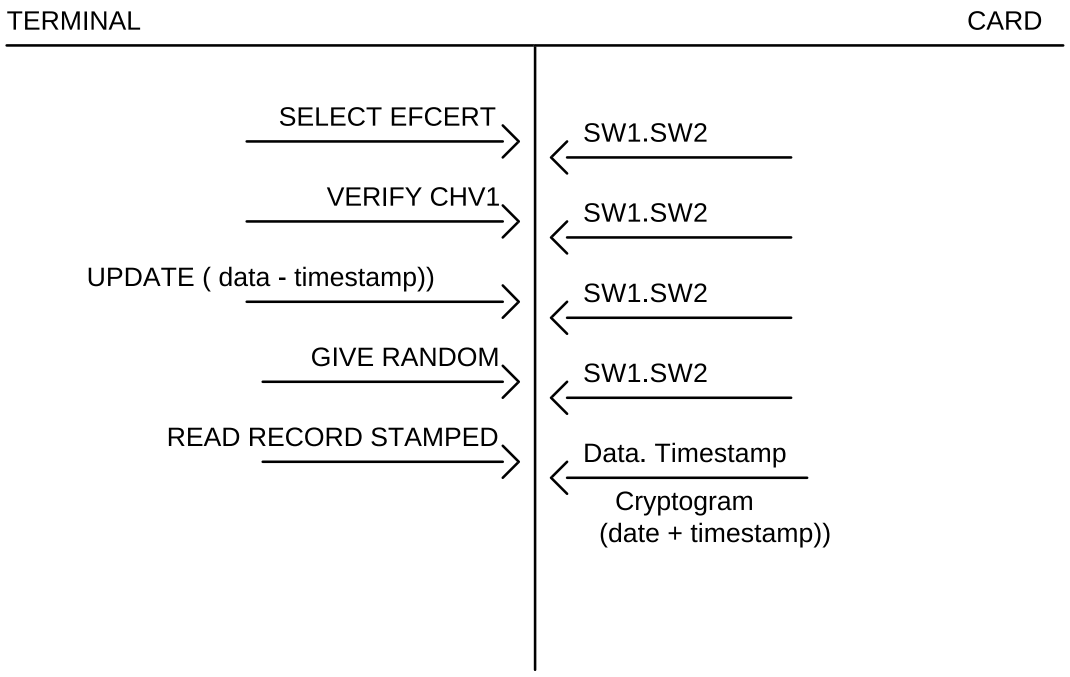


Figure B.1: Certification of external data

Access conditions for EF_{CERT}:

- WRITE CHV1;
- UPDATE CHV1.

B.2 Certification of data written in the card (in EF1)

Figure B.2 shows how to provide this certification.

After an UPDATE function on a file EF1, the terminal has to send to the card the time stamp by using the GIVE RANDOM function (here, the timestamp is considered as the random number). Then, using a READ RECORD STAMPED with AC=CHV1 on the same record, the card shall send a cryptogram comprizing the data of the previous UPDATE and the timestamp.

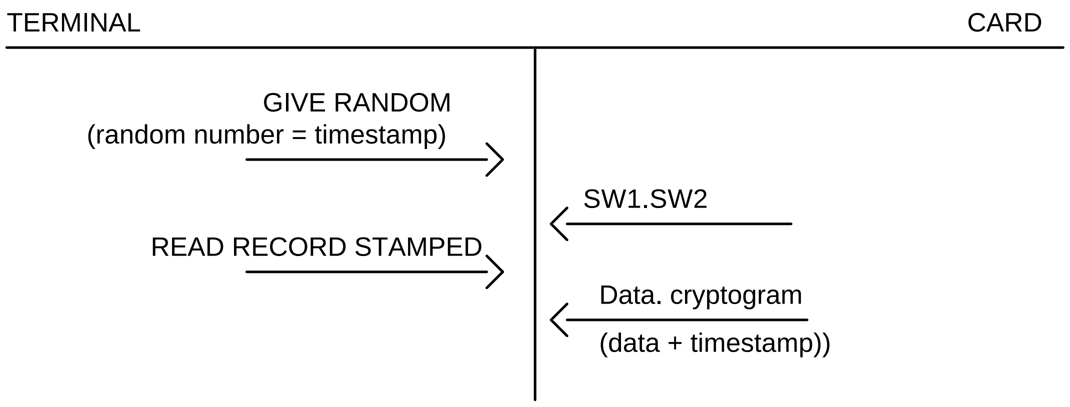


Figure B.2: Certification of data written in the card (in EF1)

Annex C (informative): Administrative actions

C.1 Card preparation

The card shall be prepared for personalization: the MF shall be created with access conditions required by the issuer. In the MF, 2 EFs shall be defined and filled by the card manufacturer: EF_{ICC} containing information related to the card manufacturer, and EF_{KEY_MAN} containing temporary keys (e.g. a production key). These temporary keys shall be diversified using the IC serial number. The AC for EF_{KEY_MAN} and EF_{ICC} shall be defined in such a way that EF_{KEY_MAN} can be updated (AC = PRO) in order to replace the temporary keys by Issuer keys. EF_{ICC} shall not be able to be updated nor deleted thanks to some specific features of the operating system or of the IC itself. Nevertheless, the AC for the MF shall not forbid deletion (e.g. for deleting a DF).

Card preparation consists of the following two steps:

a) MF personalization (see subclause C.1.1);

b) MF activation (see subclause C.1.2).

C.1.1 MF personalization

The card issuer is responsible for the card personalization process. The card issuer shall receive from the manufacturer, in an appropriate and secured way, the cards which have to be personalized, and the corresponding temporary keys (e.g. production keys). Therefore, the card issuer:

a) shall replace the content of the EF_{KEY_MAN} under the MF with his own keys. This replacement is done under the control of the temporary keys;

b) can create and fill new EFs;

c) can create and fill EF_{KEY_OP} at the MF-level.

C.1.2 MF activation

The MF activation prepares the IC for use by the user, and the application provider.

C.2 Application preparation

DF preparation consists of the following three steps:

a) DF allocation (see subclause C.2.1);

b) DF personalization (see subclause C.2.2);

c) DF activation (see subclause C.2.3).

C.2.1 DF allocation

DF allocation shall be conducted under the security policy of the card issuer for DF creation. This operation can be done directly after personalization of the MF or during the life of the card, under control of the issuer.

It is possible to modify or extend the general command set by creating an ASC (Application Specific Command set), which shall be valid for this DF and its subtree.

C.2.2 DF personalization

The application provider shall be responsible for the DF personalization process. The general process which is used for loading the files of a new application in the card, shall be consistent with the process described in annex A.

If protection against unauthorized use is required, then the filling of the file can be (depending on the AC for writing) protected by a cryptographical process (see clause 7: AC = PRO for writing/updating).

C.2.3 DF activation

Annex A gives an example of the creation of an application in the card fulfilling these general requirements.

C.3 Usage

The security mechanism is defined in clause 7.

C.4 Termination of use

Two main cases of termination of use exist:

a) MF termination;

b) DF termination.

The use of MF or DF is terminated by means of invalidating (under the AC defined for this action). The respective termination can be cancelled by the rehabilitate function.

Use of MF or DF protected by a CHV can be prevented by N consecutive wrong CHV entries and can be allowed again by using the UNBLOCK CHV function.

The prevention of use of MF or DF shall only be irreversible when no more possibilities for unblocking attempts exist (Number of remaining Unblock mechanism use = 0).

Once an application has been deleted, it is irreversibly lost.

Bibliography

The following material, though not specifically referenced in the body of the present document (or not publicly available), gives supporting information.

- EN 27810 (1989): "Identification cards - Physical characteristics".
- ISO/IEC 7812 (1993): "Identification cards - Identification of issuers".
- ISO 9564-1 (1991): "Banking - Personal Identification Number management and security - Part 1: PIN protection principles and techniques".
- ISO 9564-2 (1991): "Banking - Personal Identification Number management and security - Part 2: Approved algorithm(s) for PIN encipherment".
- ISO 9807 (1991): "Banking and related financial services - Requirements for message authentication (retail)".
- ISO 9992-1 (1991): "Financial transaction cards - Messages between the integrated circuit card and the card accepting device - Part 1: Concepts and structures".
- ISO 9992-2: "Financial transaction cards - Messages between the integrated circuit card and the card accepting device - Part 2: Functions, messages (commands and responses), data elements and structures".
- ISO 10202-0: "Financial transaction cards - Security architecture of financial transaction systems using integrated circuit cards - Part 0: System overview".
- ISO 10202-1 (1991): "Financial transaction cards - Security architecture of financial transaction systems using integrated circuit cards - Part 1: Card life cycle".
- ISO 10202-2: "Financial transaction cards - Security architecture of financial transaction systems using integrated circuit cards - Part 2: Transaction process".
- ISO 10202-3: "Financial transaction cards - Security architecture of financial transaction systems using integrated circuit cards - Part 3: Cryptographic key relationships".
- ISO 10202-4: "Financial transaction cards - Security architecture of financial transaction systems using integrated circuit cards - Part 4: Secure application modules".
- ISO 10202-5: "Financial transaction cards - Security architecture of financial transaction systems using integrated circuit cards - Part 5: Use of algorithms".
- ISO 10202-6: "Financial transaction cards - Security architecture of financial transaction systems using integrated circuit cards - Part 6: Cardholder verification".

History

Document history		
V1.1.1	August 1997	Not published (Reason: CEN copyright)
V1.2.1	January 1998	Publication
V1.3.2	December 1998	Publication

ISBN 2-7437-2741-1
Dépôt légal : Décembre 1998