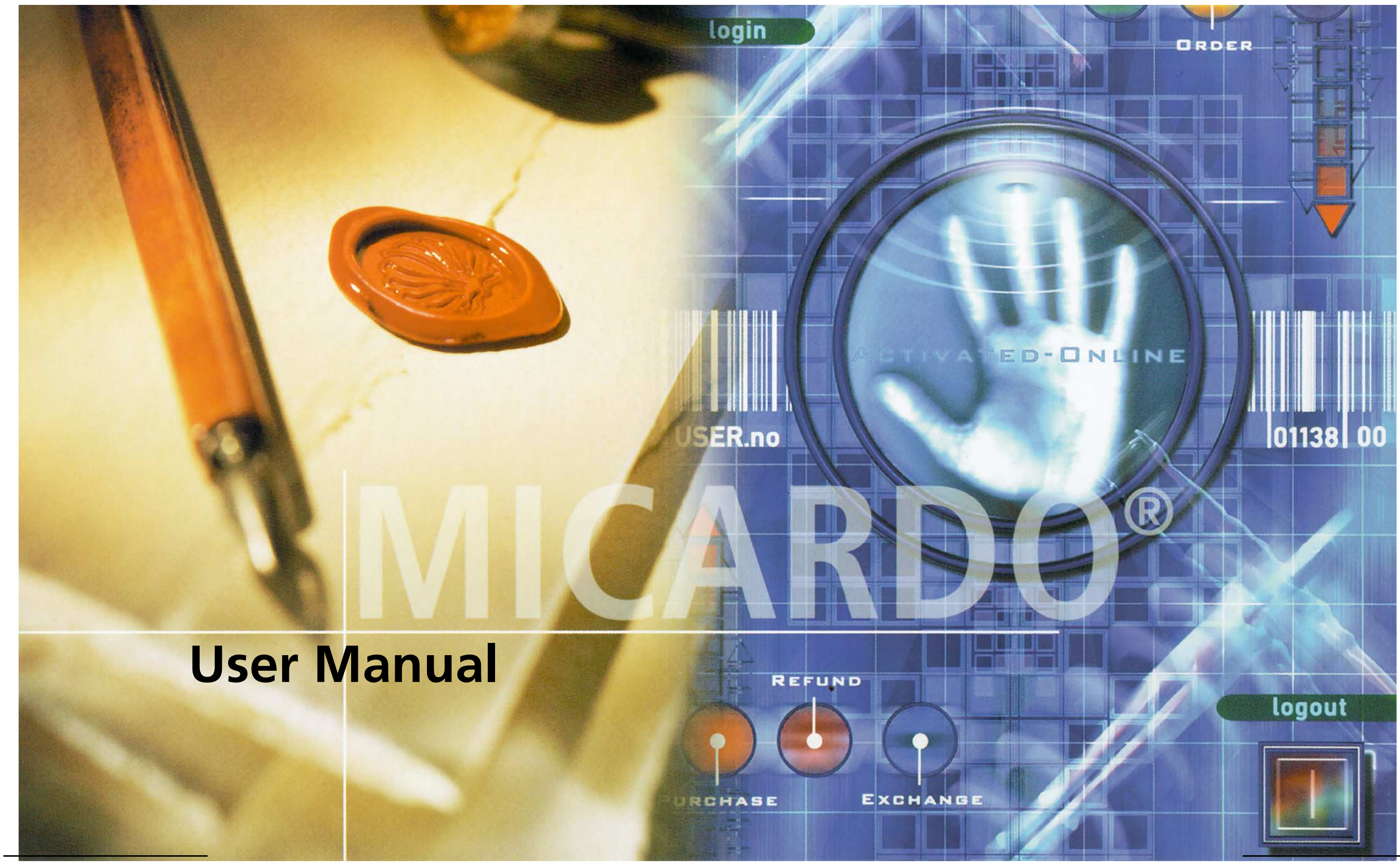
ORGA
The Smart Card Integrator

MICARDO® Public

Chip Card Operating System

Version 2.1 64/32 R1.0



User Manual

ORGA

MICARDO Public

MICARDO Public Chip Card Operating System - User Manual Version 1.00

MICARDO Public Chip Card Operating System - Version 2.1 64/32 R1.0

Contents

C

1 Introduction

1.1 What is MICARDO?

MICARDO (MICARDO PUBLIC Version 2.1 64/32 R1.0) can be described in a few words as a **multifunctional chip card operating system** providing a digital signature function.

In principle, a chip card operating system performs the same tasks as a PC's operating system. In a PC, the operating system manages the files and directories, controls access to storage media and is used to configure the hardware.

The tasks of a chip card operating system can be described as follows:

– it controls data transfer to the chip card
– it controls the execution of commands
– it manages the files
– it executes cryptographic algorithms

Like a PC operating system MICARDO is completely independent of all applications. It provides a large number of functions which can be used by a wide variety of different applications.

For example, MICARDO offers cryptographic services, loading of non-resident applications and extensions to existing applications.

1.2 Target Group

This manual has been written for MICARDO users who are already familiar with chip card technology and with the terminology used in this field. This means, for example, that they know what a PIN is and what DES encryption involves. In addition, they know the different standards which have been defined by national and international standards committees.

You should consult the relevant ISO standards if you need more detailed information on the technical terms and definitions used in this manual.
A list of standards at the end of the manual shows which of them are relevant to MICARDO.

The MICARDO manual contains all the information you need to design your own applications and to implement them in a chip card.

1.3 Notes for the Reader

In this manual the terms MICARDO and operating system are used to mean one and the same thing. When mention is made of a chip card, this normally refers to a chip card that MICARDO has implemented.

You do not need to read the manual through completely from start to finish. Each chapter deals with a separate subject and can be selected for reading as required.
However, the chapter ”Structure“ should be one of the first chapters that you read. It not only gives an excellent overview of the MICARDO structure but also explains in detail all the structures used in the manual, such as the structure of the command descriptions.

An additional point worth mentioning here is that the individual chapters of the manual are related to each other or overlap. Please use the index if you cannot find the explanation for a term or procedure in a particular chapter and then consult the pages specified there.

1.4 Overview of Chapters

The manual is divided into the following chapters:

Chapter 1	This chapter gives a brief introduction and explains how to use the manual.
Chapter 2	This chapter describes the characteristic features and security functions. Here you will find the norms and standards with which MICARDO complies.
Chapter 3	This chapter describes in detail the structure of MICARDO and the layout of all other relevant structures.
Chapter 4	This chapter shows which data fields must be available in MICARDO and how they are created. It also contains information on passwords and keys.
Chapter 5	This chapter explains which security features are used by MICARDO.
Chapter 6	This chapter explains the MICARDO operating system commands.
Chapter 7	This chapter describes which protocols MICARDO uses and how they have been adapted.
Chapter 8	This chapter describes the basic initialisation and personalisation procedures.
Appendix	The appendix contains for example a list of abbreviations, a list of relevant standards and a glossary explaining the terms used in this manual.
Index	The index will facilitate your search for individual terms.

1.5 Notational Conventions

The following notational conventions are used in this manual:

"..."	Text between inverted commas indicates references to other chapters of the manual

/.../ Text between slashes indicates references to other documentation

Special attention must be paid to text following this symbol.

Text following this symbol contains information of a general nature.

This mode	indicates the following...
bold	Important text passages
italics	MICARDO components
´FF´	Hexadecimal figures
<...>	Placeholder
B<n>	Byte with the number <n>
b<n>	Bit with the number <n>
||	Concatenation
COMMAND	Command
`File`	Directory structure, file or example
Mode	Status descriptions

Characteristic Features

2.1 Security Functions

MICARDO is a chip card operating system for high security applications. Therefore MICARDO offers different kinds of security functions.

2.1.1 Authentication procedures

MICARDO allows the authentication of users in order to authorize access to certain objects in the intended way. After successful authentication the user may be authorized to access certain objects stored in the chip card.

2.1.1.1 Password Based User Verification

MICARDO offers a password (password or PIN) based authentication procedure. The card user proves his identity to the MICARDO chip card by entering the password. This process is referred to as cardholder authentication.

Users can be verified by means of this cardholder authentication process. For this purpose, MICARDO compares the password provided by a card user with a password reference data stored in the chip card.

MICARDO detects when a defined number of unsuccessful authentication attempts occur related to any cardholder authentication with the help of retry counters. These retry counters are mandatory.

When the defined number of unsuccessful attempts has been met or surpassed, MICARDO blocks the corresponding password. Only a resetting function may reset the retry counter to unblock the corresponding password which is protected by a security attribute.

MICARDO limits the password usage, when a usage counter is present. When the defined number of possible usages has been surpassed, MICARDO invalidates the corresponding password. Only a change-reference-data function may reset the usage counter to unblock the corresponding password which is protected by a security attribute. A new password has to be set.

When the password will be transfered to MICARDO, secure messaging may be used.

The section "Basics about Password and PIN" on page 42 gives among other things basic information about the storage and transmission format and about the values of retry and usage counters.

Additionally there are sections, which describe the structure and the contents of those data fields where passwords and PINs are stored (section "EF_Pwd" on page 51, section "EF_PwdD" on page 52 and section "EF_PwdC" on page 56).

2.1.1.2 Key Based User Authentication

Another way of authentication is the key based user authentication where the external world performs a component authentication.

Users can be authenticated by means of a challenge/response procedure using random numbers and cryptographic keys. MICARDO allows the authentication of the user as well as of the chip card or both.

MICARDO detects when a defined number of unsuccessful authentication attempts occur related to any key-based authentication process with the help of retry counters. These retry counters are optional. When the defined number of unsuccessful attempts has been surpassed, MICARDO invalidates the corresponding key. Another key has to be used.

MICARDO limits the key usage of an authentication key, when a usage counter is present. When the defined number of possible key usages has been surpassed, MICARDO invalidates the corresponding key. Another key has to be used.

MICARDO offers symmetrical authentication procedures as well as asymmetrical procedures. The section ”Authentication Procedures” on page 102 shows the difference between these procedures and explains which operating system commands MICARDO uses.

2.1.2 Access Control

MICARDO controls the access to all its objects like directories, data fields, passwords and keys by checking the corresponding security attributes.

2.1.2.1 Management of Access Control

MICARDO enforces the access control to restrict the ability to create, modify and delete the security attributes to the administrator. MICARDO maintains the following list of security attributes:

- Access rules consisting of
  - Access modes
  - Access conditions for objects (files, keys and passwords)
- Usage Qualifiers
- Life cycle status integer of files

An object can only be accessed if an access rule (AR) has been attached to the object during its creation. After deletion of an object’s access rules it can no longer be accessed.

Access conditions (AC) define the conditions under which a command executed by a user is allowed to access objects. MICARDO maintains the following list of primitive access conditions:

ALW	Always, no condition
NEV	Never
PWD	Password based user authentication
AUT	Key based user authentication
PRO SM SC	Secure messaging providing data authentication and integrity

ENC SM SC	Secure messaging providing data confidentiality
SPEC	Command specific access condition

It is allowed to combine these primitive access conditions. The object’s access condition assigned to an access mode comprises a logical expression of primitive access conditions using logical AND and logical OR.

A detailed description of access rules as security attributes can be found in section ”Access Modes” on page 134, section ”Access Conditions” on page 137 and in section ”Combinations and Evaluations” on page 141.

Furthermore, MICARDO maintains conditions that restrict the usage of keys, so called usage qualifiers (UQ). MICARDO maintains the following list of control reference templates (CRT) for usage qualifiers:

DST	Key usage is restricted to digital signatures
CCT	Key usage is restricted to cryptographic checksums
CT	Key usage is restricted to confidentiality
AT	Key usage is restricted to authentication

The CRT is followed by one usage qualifier. The section ”Usage Qualifier” on page 16 explains the coding of the usage qualifier in detail.

Additionally, MICARDO maintains the following list of life cycle status integer (LCSI):

Activated	File is in operational status
Deactivated	File is in operational status but its data cannot be accessed

The LCSI cannot be deleted. The administrator can switch between the defined status using an activate or a deactivate command.

The security attributes are stored at different places.

The data field EF_Rule contains access rules. The section ”Data Field EF_Rule“ on page 132 describes the structure of this data field.

The data field EF_PwdD contains the access rule references for the passwords. A detailed description of the data field and its special data objects can be found in section "EF_PwdD" on page 52 and section "SE Data Object for Passwords" on page 47.

The data field EF_KeyD contains the access rule references and the usage qualifiers for the keys. The structure of the data field is explained in section "EF_KeyD" on page 70. The section "SE Data Object for Keys" on page 72 contains information about the special data objects.

2.1.2.2 Security Attribute Based Access Control

MICARDO enforces the access control to objects based on security attributes, e.g. a set of access rules (AR). Additionally, the access control uses the actual security status (ASS) as the reference value for the rule decisions.

MICARDO explicitly authorizes access of users to objects based on the following rules:

- MICARDO allows access to an object for a defined access mode, if the object's access condition is valid for this access mode.
- MICARDO evaluates the logical expression of primitive access conditions (boolean expression) according to the following rules:
  - Primitive access condition ALW is set to **true**.
  - Primitive access condition NEV is set to **false**.
  - Primitive access condition PWD is set to **true**, if PWD complies with the actual security status.
  - Primitive access condition AUT is set to **true**, if AUT complies with the actual security status.
  - Primitive access condition PRO SM SC is set to **true**, if PRO SM SC complies with the user indication for PRO SM SC.
  - Primitive access condition ENC SM SC is set to **true**, if ENC SM SC complies with the user indication for ENC SM SC.
- MICARDO allows access to a key for a defined access mode, if the access condition of this key is valid and the key is used in compliance with the key's usage qualifier.

MICARDO explicitly denies access of users to objects based on the following rules:

- MICARDO denies access to a referenced key if it is to be used in a context which does not match the key's usage qualifier even if the access condition is valid for this key.
- MICARDO denies access to an object for a defined access mode, if PRO SM SC is indicated by the user and there does not exist a primitive access condition PRO SM SC in the object's access condition.
- MICARDO denies access to an object for a defined access mode, if ENC SM SC is indicated by the user and there does not exist a primitive access condition ENC SM SC in the object's access condition.
- MICARDO denies access to a file's user data, if it is deactivated, regardless of the file's access condition.

The actual security status (ASS) for PWD and AUT is valid for a single password, a single key or a key group.

2.1.3 Integrity of Data

MICARDO ensures the integrity of stored data by using checksums.

MICARDO monitores user data stored within the chip card for integrity errors on all data fields (EF) and directories (DF), based on the following attributes:

- A checksum (CRC) is attached to each header of a file.
- A checksum (CRC) is attached to the user data contained in file.

Upon detection of a data integrity error, MICARDO informs the user about this integrity error. If the file header's checksum has been corrupted, the user data containded in the file is no longer accessible.

The section "Security Features for the EEPROM" on page 145 describes which mechanisms are used.

2.1.4 Authenticity of Data

MICARDO provides a capability to verify the authenticity of images.

Images can only be used if their authenticity has been verified by an authentication code.

2.1.5 Data Exchange

2.1.5.1 Data Exchange Confidentiality

MICARDO provides a capability to ensure that secret data which is exchanged between the chip card and the user remains confidential during transmission. For this purpose, encryption based on cryptographic keys is applied to the secret data.

MICARDO ensures that the user and the user's data access condition have indicated data exchange confidentiality.

Keys used are either loaded by a user, or they are generated during user authentication (so called session keys).

The confidentiality of data during exchange is ensured by using secure messaging. The section "Secure Messaging" on page 113 contains all information about secure data transmission.

2.1.5.2 Data Exchange Authenticity and Integrity

MICARDO provides a capability to ensure that the data which is exchanged between the chip card and the user remains authentic during transmission. Cryptographic checksums, which are based on cryptographic keys, are used for this purpose.

MICARDO ensures that the user and the user's data access condition have indicated data exchange authenticity and integrity.

Keys used are either loaded by a user, or generated during user authentication (so called session keys).

The authenticity of data during exchange is ensured by using secure messaging. The section "Secure Messaging" on page 113 contains all information about secure data transmission.

2.1.6 Object Reuse

MICARDO ensures that any previous information content of a resource is explicitly erased upon the deallocation of the resource form the following objects:

- all data fields (EF) and directories (DF)
- volatile memories used for computations in which secret keys and other secrets (e.g. passwords) are involved

2.1.7 Physical Protection

2.1.7.1 Hardware Failure Protection

MICARDO preserves a secure state when the following types of failures occur:

- Induced hardware failures, transient or permanent, during the execution of a command.
- Tampering

Upon detection MICARDO reacts in a way that the security policy is not violated, i.e. that the chip card cannot be used any longer during the session. It has to be resetted.

MICARDO uses hardware based security functions and corresponding mechanisms to enforce and monitor hardware failures.

2.1.7.2 Covert Channel Analysis Control

MICARDO manages all hardware based mechanisms which can be applied to prevent a covert channel analysis like simple power analysis (SPA) and differential power analysis (DPA).

MICARDO acts in such a way that all crytographic operations are supported by these hardware mechanisms. MICARDO ensures that all software and hardware countermeasures available are used such that they support each other.

MICARDO enforces that a secure session is installed before any cryptographic key is loaded into the volatile memory and processed in a cryptographic operation.

2.1.8 Cryptographic Operation

2.1.8.1 Cryptographic Key Generation

MICARDO generates asymmetric cryptographic keys (512 - 1024 bit RSA keys). It enforces that the key material meets the following requirements:

- Random numbers used in the key generation have a high quality to ensure that the key is unique with a high probability.
- Prime numbers used in the key generation are unique with a high probability.
- Any private key cannot be derived from the corresponding public key.

Especially, for generating keys used for digital signatures compliant to the German digital signature act the RSA keys at present must have a key length of 1024 bits. MICARDO works in a manner that only cryptographically strong keys (with the given key length) are generated.

2.1.8.2 Computation of Digital Signatures

MICARDO provides a digital signature functionality. The digital signature function works in a manner that no information about the cardholder's secret key stored in the chip card and used for the digital signature may be disclosed during generating the digital signature. Furthermore, the cardholder's private key cannot be derived from the signature, and the signature cannot be generated by other individuals not possessing that secret.

The keys used for digital signatures are either generated inside MICARDO or they are generated by the external world and loaded into the chip card. Optionally, the usage of a signature key for digital signatures can be restricted in such a way that every computation of a digital signature with this key requires a successful verification of a corresponding password.

Under assumption that a signature key suffices the requirements of the German digital signature act MICARDO issues act compliant digital signatures by using this key.

A detailed description can be found in section "Digital Signatures" on page 107.

2.2 ITSEC E4 High Evaluation

MICARDO is available in an ITSEC E4 high evaluated version (see /ITSEC/).

This version is present if during the initialisation the evaluated standard image has been loaded (see section "Initialisation and Personalisation" on page 213).

If the standard image has been changed due to customer request, the following two cases have to be regarded:

- New directories and data fields are created additionally.
  The E4 evaluation of the standard image is also valid for this image.
- New operating system commands are inserted.
  An evaluated E4 card is only present if the new operating system is evaluated.

2.3 Standards

2.3.1 Security Features Standards

The MICARDO security features are based on a number of international standards, enabling the operating system to provide interoperable cryptographic services to a variety of different applications.

Additionally MICARDO takes the following standards into account:

- DES algorithm compliant with /ANSIX392/
- RSA signature generation and decryption compliant with /PKCS1/
- Hash algorithm SHA-1 compliant with /FIPS180-1/

2.3.2 ISO Standards

MICARDO complies with the following ISO standards:

- The contact-based interface and the transmission protocols that are used comply with the standard /ISO 7816-3/.
- The file system, secure messaging and the standard operating system commands comply with the standard /ISO 7816-4/ and its ammendment /ISO 7816-4A1/.

– The PSO commands (perform security operation), which provide the external world with security services, comply with /ISO 7816-8/.
– The access rules and additional standard operating system commands comply with /ISO 7816-9/.

2.3.3 Banking Sector Standards

In addition to these, MICARDO meets the requirements of the following national and international standards from the banking sector:

– The current specification for the data structures and commands of the GeldKarte (Germany's electronic purse) operating system /ZKA41/.
– The international standard /EMV2000/ for EMV-compatible card products.

2.3.4 Additionally Supported Applications

MICARDO also supports other standard applications for a variety of sectors and markets. These applications are:

– Signature application compliant with /DINV66291-1/
– Health professional card compliant with /HPC1.0/

2.4 Miscellaneous

2.4.1 CRC Generation

The CRC (cyclic redundancy check) is generated to comply with CCITT X.25, ISO 2111, ISO 3309, whereby the check polynome is defined as follows:

$x^{16} + x^{12} + x^{5} + 1$

2.4.2 Data Objects

The data objects used in MICARDO are coded to comply with /ISO 7816-4/. They thus comply with the distinguished encoding rules (DER) in accordance with ASN.1 and have a TLV structure coded (tag, length, value) complying with /ISO 8825-1/.

The following rules must be noted when specifying length:

Tag	Length	Value
T	L	V
1..2 byte	1..3 byte	<n> byte

1 byte	2 byte	3 byte
‘00‘.. ‘7F’	...	...
decimal 0 .. 127	...	...
‘81‘	‘80´.. ‘FF’	
‘81‘	decimal 128 .. 255	...
‘82‘	> ‘FF’	
‘82‘	decimal 256 ...65535	

MICARDO uses both simple data objects and compound data objects.

2.4.3 Usage Qualifier

Coding of the usage qualifier complies with /ISO7816-9/ and defines for which service in which it is set the CRT is to be used. It is coded in a simple data object with the tag ´95´.

b8	b7	b6	b5	b4	b3	b2	b1	Description
1	...	...	...	...	...	...	...	External and mutual authentication (AT)
...	1	...	...	...	...	...	...	Data authentication (DST), data confidentiality (CT), internal and mutual authentication (AT)
...	...	1	...	...	...	...	...	Secure messaging for response to message (CCT, CT)
...	...	...	1	...	...	...	...	Secure messaging for command message (CCT, CT)

b8	b7	b6	b5	b4	b3	b2	b1	Description
...	...	...	...	1	...	...	...	User authentication (AT)
...	...	...	...	...	x	x	x	RFU

A set bit permits the CRT to be used in the relevant service, a cancelled bit prohibits usage. Theoretically the usage qualifier can enable all services, but in practice a more detailed description of the CRTs only permits a subset of the possible usage qualifiers to be used.

2.4.4 Access Types

The AM byte, which defines the access types for the standard operating system commands, is coded to comply with /ISO7816-8/.

2.4.5 DSI Format

The DSI format (digital signature input) defines which format may be used when decrypting data or generating a digital signature.

The DSI format is coded in a simple data object with the tag ´80´ and the length ´01´. At present MICARDO does not consider the value and uses the relevant DSI format automatically.

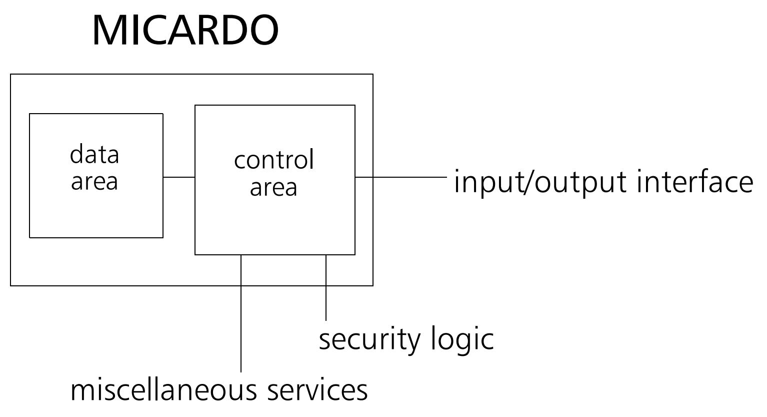
ORGA

Structure

3

3.1 Interfaces and Components

The operating system has the following structure:



The following external interfaces are available:

– Input/output interface
  MICARDO users send data to the operating system and fetch response data from it via the input/output interface.

– Security logic
  The security logic includes all hardware mechanisms which discover and repel any hardware intrusion.
– Miscellaneous services
  These services comprise all other hardware mechanisms that are used by the operating system, e.g. random number generators, CRC modules or ROM firmware.

Internally the operating system can be divided into two general areas: data and control.

The control area contains the following components:

– *Input/output handler*
– *Command set*
– *Access control*
– *Secure messaging*
– *Cryptographic services*
– *Security status*
– *Security environment*
– *File management*

The data area comprises the:

– *File system*

All components of the data and control areas are explained in the sections that follow. An initial illustration gives an overview of the memory areas which accommodate these components.

input/output handler	ROM
command set (standard commands)	
access control	
secure messaging	
cryptographic services	
file management	
command set (additional commands)	EEPROM
file system	
security environment	
security environment	RAM
security status	

3.1.1 Input/Output Handler

The *input/output handler* is located in ROM and is the component of the operating system which makes the chip card protocols available.

The following chip card protocols are supported:

– International T=0 protocol
– International T=1 protocol

Together with the chip card the *input/output handler* uses special commands to set the transmission speed of the chip card protocol.

3.1.2 Command Set

The MICARDO *command set* consists of two parts: the standard operating system commands are stored in ROM and the additional commands, which are stored in the EEPROM during initialisation, when they are required.

The *command set* includes:

- Commands which permit access to data in the file system
- Commands to authenticate the cardholder
- Commands to manage the access conditions

These operating system commands and therefore also the *command set* contain command-specific error handling.

3.1.3 Access Control

Access control is stored in ROM and monitors access to all data which has been stored in the *file system*.

This component checks whether the access conditions stored in the data fields or directories actually comply with the relevant *security status*.
Besides this, it activates *secure messaging* if the access conditions require this.

3.1.4 Secure Messaging

Secure messaging is the component in MICARDO which maintains the security of confidential data by using MAC security and encryption.

3.1.5 Cryptographic Services

MICARDO offers the following cryptographic services:

- Data encryption standard (DES or Triple DES)
- Random number generator
- RSA decryption and signature algorithm (RSA)
- Secure hash algorithm (SHA-1)

All cryptographic services are located in ROM.

3.1.6 Security Status

The *security status* component, which is stored in RAM, stores the access rights that are reached through verification and external authentication.

Storage in volatile memory results in loss of the *security status* as the immediate result of a power interrupt.

3.1.7 Security Environment

The non volatile part of the *security environment* is stored in the EEPROM. This part consists of key and algorithm references which are stored in a separate file in the *file system*. This file belongs to the particular directory in which it is stored. Several different *security environments* can be stored in one file.

The volatile part of the *security environment* is stored in RAM. This part consists of key and algorithm references which can be input via a special command in order to manage the *security environment*.

3.1.8 File Management

This operating system component manages the data of the *file system*. It is located in ROM.

File management also controls the selection of applications and their data fields. It guarantees that all commands access the selected data correctly.

This component also makes search routines available for directories, data fields, keys and passwords.

3.1.9 File System

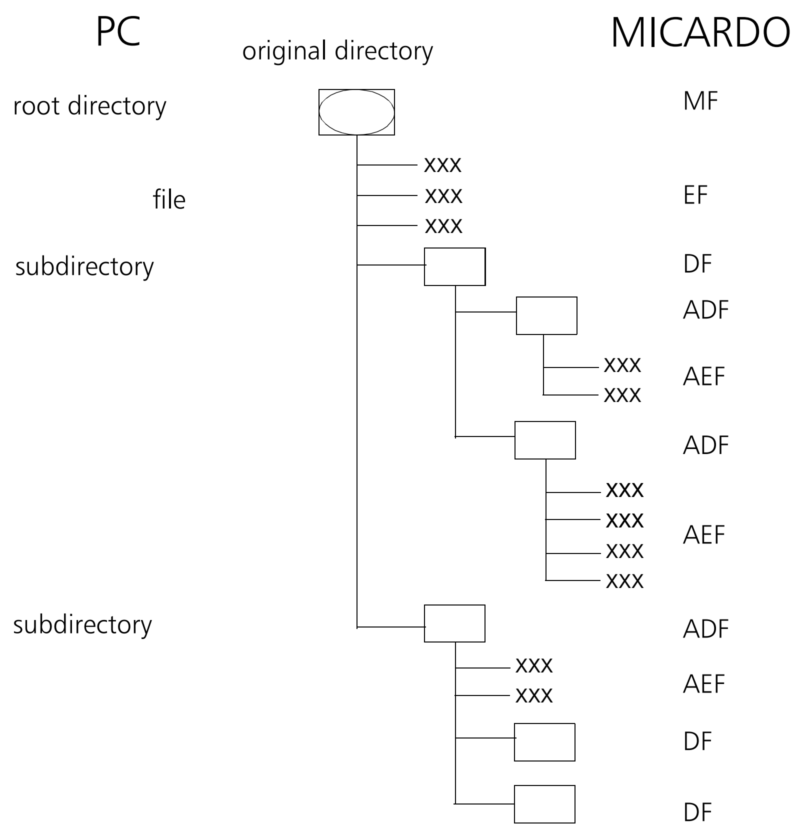
The MICARDO data is stored in the EEPROM as files in a *file system*.

The structure of this component is described in detail in the next section ”Data Structures in the File System“.

3.2 Data Structures in the File System

The MICARDO file system contains similar data structures to the directory structure (tree structure) of a PC's operating system.

The following figure compares these structures and explains the special terms used in the MICARDO data structure.



Legend:

MF	Master file
EF	Elementary file (data field)
DF	Dedicated file (directory)
ADF	Application dedicated file
AEF	Application elementary file

3.2.1 Master File

The master file (MF) is the original file in the MICARDO file system. It is automatically selected following each chip card reset.

It contains data fields, also known as elementary files (EF), with global data which any application can access. This is, for example, manufacturer information, card number, global PINs or global keys for authentication and cryptographic functions.
Directories, also referred to as dedicated files (DF), and application dedicated files (ADF) can be configured under the original file.

3.2.2 Directories

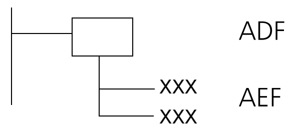
In principle, any number of directories (DF) can be configured. Memory space alone may restrict the number that is created.

3.2.3 Application

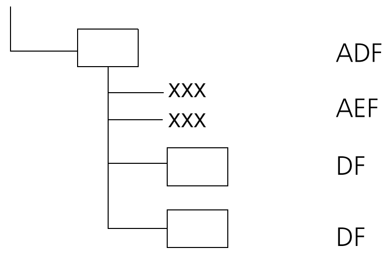
An application is a logical group of data within the MICARDO file system. Each application has its own application dedicated file (ADF). The following rules apply to this directory:

– Each DF, also the MF, can be the ADF of an application.
– A DF may only be ADF for a maximum of one application.

Application-specific data is stored in elementary files (AEF) in the application dedicated file.



However, additional dedicated files (DF) can be created under an application dedicated file to obtain a clearer view of an application and, for example, in order to store similar information of different users of an application.



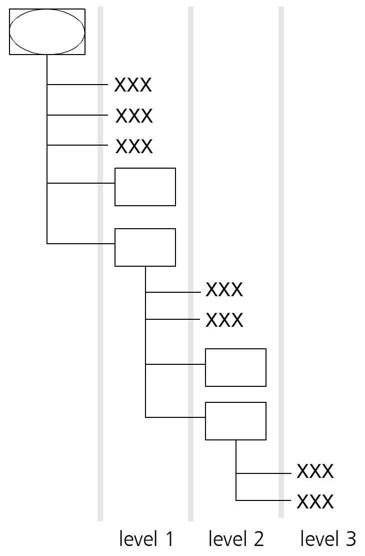
3.2.4 Data Fields

Each directory (MF, ADF or DF) can have data fields (EF or AEF). The information necessary for operation (data, keys, passwords, etc.) is stored here.

EFs and AEFs can be read and also written by the external world. However, they can be protected or disabled (e.g. with key and PIN) for access (read or write). It is therefore possible to make certain information available to authorised persons only.

3.3 Directory Levels

MICARDO's directory structure also divides its file system into various directory levels.



If subdirectories are created, these - together with the higher-level directory - are referred to as nested directories. A nested directory thus consists of at least one higher-level directory and one to <n> subdirectories.

A subdirectory is always at a lower level than the higher-level directory but at the same level as the <n> subdirectories.

3.4 Identifiers

3.4.1 Designations

Directories and data fields must be selected before MICARDO can perform activities with or in them. These activities include deleting directories and reading or writing data in data fields.

It is thus crucial that MICARDO can identify the precise directory or data field which is selected for an activity.
This unambiguity is ensured through identifiers.

The following table describes these identifiers:

File	Length	Value	Designation/Comments
MF	2 bytes	´3F 00´	File identifier (FID) (reserved FID)
DF	2 bytes 1- 16 bytes	freely definable freely definable	File identifier (FID) Name of directory
ADF	2 bytes 5-16 bytes	freely definable freely definable	File identifier (FID) Application identifier (AID) - Name of application
EF/ AEF	2 bytes 5 bits	freely definable freely definable	File identifier (FID) Short file identifier (SFI) - Name of data field

All identifiers, with the exception of the short file identifier (SFI), are mandatory and must be assigned during creation of directories and data fields.

3.4.2 Rules for Assignment

In order to guarantee unambiguity, it is **mandatory** that the following rules are complied with when assigning identifiers:

– FIDs reserved for directories and data fields must not be used
– DFs and EFs within a directory must have different FIDs
– Nested directories may have the same FID
– An AID may only occur once in the entire file system
– An FID for a data field may only occur once in a directory but the FID may be identical to that of the directory

3.4.3 Addressing

Internally MICARDO manages a directory pointer, which points to the current directory, and a field pointer, which points to the current data field.

Directories can be selected in two different ways:

Implicit	This type of selection includes the name of the directory. As the names in the file system are unique, selection is possible from any position within the file system.
Direct	Direct selection uses the FID. Here, a directory can only be reached from the higher-level directory, as an FID is only unique within a directory.

There are two ways of selecting data fields, similar to the options for directories:

Implicit	This type of selection includes the name of the data field (SFI). This means that a data field can be selected from the directory in which it has been created.
Direct	Explicit selection uses the FID. An EF can only be selected directly from the directory in which it has been created.

3.5 Data Field Types

The following table summarises the data field types available in MICARDO:

Data field type	Description
Transparent data field	
Transparent	Non formatted data field; sequence of bytes which can be accessed individually or in variable length
Formatted data field	
Linear, constant length	Formatted data field with records which all have a constant length
Linear, variable length	Formatted data field with records which have a variable length
Cyclic	Formatted data field derived from a linear data field with constant length; the records are arranged cyclically

3.6 Structure of Commands/Responses

Communications with a chip card always consist of a command from the external world and a response from the chip card. We distinguish here between secured (with secure messaging, see page 22) and non secured transmission.

Detailed descriptions of the connection setup in the case of secured transmission can be found in the chapter chapter "Structure of Messages under Secure Messaging" on page 114. The following section deals with the structure of non secured transmission.

A command or a response can contain information (data). We thus distinguish between four different cases:

Case	Command	Response	Sample command
Case 1	No Data	No Data	**ACTIVATE FILE**
Case 2	No Data	Data	**READ BINARY**
Case 3	Data	No Data	**UPDATE BINARY**
Case 4	Data	Data	**SEARCH BINARY**

The command and response data is embedded in APDUs (application protocol data unit).

3.6.1 Structure of a Command APDU

MICARDO accepts command APDUs which have the following structure:

Command header				Command data		
CLA	INS	P1	P2	Lc	Data	Le

CLA:
This byte (class byte) identifies the command class of a command and is used for control purposes, e.g. to indicate secure messaging.

INS:
This byte (instruction byte) references the required command.

P1, P2:
P1 and P2 are the control parameters for the particular command.

Lc:
Lc can assume values from 1…255 inclusive and indicates the number of bytes in the data area (Data).

Data:
This area, also referred to as the data area, contains the actual data.

Le:
Le can assume values from 1…256 inclusive, where the value 256 is represented by ´00´ and indicates the expected length of the response data.

Evaluations

MICARDO only accepts the short version of the command data and subsequently analyses it.

Let's assume that the command data contains <n> bytes:
B1, B2, …, B<n>, whereby n=0 is permitted.

MICARDO evaluates this byte sequence in the following way:

Conditions		Case
n=0		Case 1
n=1		Case 2
n>1	B1 = ´00´	Error: ´67 00´
n=1+B1	B1 ≠ ´00´	Case 3
n=2+B1	B1 ≠ ´00´	Case 4
n=other values	B1 ≠ ´00´	Error: ´67 00´

Based on the definitions shown in this table Lc and Le are always one byte in length if they are present in the command data.

3.6.2 Structure of a Response APDU

MICARDO outputs response APDUs which have the following structure:

Response data	Trailer	
Data	SW1	SW2

The length of the response data is indicated by Lr. Lr is restricted to a maximum of 256 bytes. If Lr is larger than this, the command aborts with an error message.
Lr can be zero. In this case there is no response data.

3.7 Descriptions

3.7.1 Description of Data Objects

MICARDO uses simple and compound data objects.

Compound data objects are described in the following way:

– Identifier of the higher-level data object (tag)
– TLV table with a list of corresponding data objects (also known as data elements) and a description of the values

Structure of a TLV table:

Tag	Length	Value
		Description of the value of the data element

If it contains optional data elements, these are indicated as shown below with a lower case (o) in the last column.

Tag	Length	Value	
			o

3.7.2 APDU Descriptions

The chapters "Operating System Commands" and "Initialisation and Personalisation" contain descriptions of the commands and responses. Their structures are described below.

A command APDU is represented in the following way:

	Length	Contents	Description
CLA	Header data		
INS			
P1			
P2			

	Length	Contents	Description
Lc	Command data		
Data			
Le			

The **Contents** column of the APDUs either have concrete values entered or variables for hexadecimal values are specified.
The entries have the following meanings:

´0A´	Fixed value
´X1´	X stands as a placeholder for 4 bits (MSB)
´FX´	X stands as a placeholder for 4 bits (LSB)
´XX´	XX stands as a placeholder for one byte
´00´...´FF´	Possible values covering one byte from ´00´ to ´FF´
´XX...YY´	X and Y stand as placeholders for a variable number of bytes with any values
´XX YY´	XX and YY stand as placeholders for one byte each; altogether two bytes are coded
´XX YY´ || ´XX YY´	XX and YY stand as placeholders for one byte each; the two bytes are concatenated; altogether four bytes are coded

Sometimes the values which the placeholders can assume are of no significance for the explanation of the command, e.g. data for an update command. In this case, the placeholders are not described in further detail.

However, if the values which can be inserted are of importance for the command they are explained in detail. A (+) in the description column indicates that an additional description can be found at the end of the table.

A response APDU has the following appearance:

	Length	Contents	Description
Data			Response data
Trailer			SW1 and SW2

3.7.3 Byte Table

A byte table describes the contents of a byte and has the following structure:

b8	b7	b6	b5	b4	b3	b2	b1	Description
x	x	x	x	x	x	x	x	

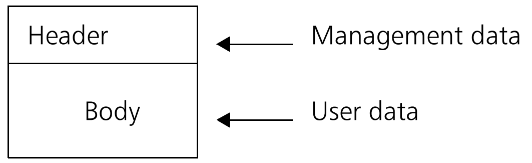
Possible inputs are:

0	The bit is set to 0
1	The bit is set to 1
x	The bit is used; the **Description** column describes in detail the purpose for which it is used
...	The bit is of no importance for the task described

3.8 File Structure

All MICARDO data is stored in files. These are divided into dedicated files (DF), also known as directories, and elementary files (EF), also known as data fields.

Files in MICARDO always have the same structure. All information on an individual file is stored in the file itself. A file can therefore be described as follows:



The header of the file contains management data. The actual user data is stored in the body.

3.9 File Header

All files in MICARDO, directories (DF) and data fields (EF), have a header. This header is also referred to as file control information (FCI) and comprises the file control parameter (FCP) and the file management data (FMD).
The FCP contains the information that is important for the file to function correctly. FMD contains optional additional information.

3.9.1 File Control Parameter

The file control parameter is a compound data object with the tag ´62´.

The following table lists the possible data elements. The sequence is always that shown. A data element may occur only once and optional data elements (o) may be omitted.

Tag	Length	Value	
´82´	´01´ or ´05´	File type (file descriptor)	
´83´	´02´	File identifier (FID)	
´84´	´01´…´10´	Name of a DF or an ADF	
´85´	´02´	Free memory space; number of bytes for user data	
´88´	´01´	Name of an EF - short file identifier (SFI)	o
´8A´	´01´	Life cycle status integer (LCSI)	
´A1´	´XX...YY´	ARR per transmission type (access rule reference)	

The next table contains an overview showing when the inputs for the data elements must be specified or when they are output by MICARDO.

Data element	At create	At select
File descriptor	must exist	is output
File identifier	must exist	is output

Data element	At create	At select
Name of a DF or ADF	must exist	is output
Free memory space		
- directories (DF)	must not exist	is output
- transparent EF	must exist	is output
- linear EF, constant	must not exist	is not output
- linear EF, variable	must exist	is output
- cyclic EF	must not exist	is not output
Name of an EF (SFI)	can exist	is output if it exists
Life cycle status integer (LCSI)	must exist	is output
Access rule reference per transmission type	must exist	is output

3.9.1.1 File Descriptor (Tag ´82´)

The file type is displayed in the data object with the tag ´82´. The data object is defined as follows:

	T	L	V
Directory (DF)	´82´	´01´	´38´
Transparent data field	´82´	´01´	´01´
Formatted data field	´82´	´05´	each byte is coded as follows:

Byte 1:

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	0	...	...	...	...	...	...	Other values RFU
		x						Access:
...	...	0	...	...	...	...	...	Standard
...	...	1	...	...	...	...	...	Time-optimised

b8	b7	b6	b5	b4	b3	b2	b1	Description
			x					Usability:
...	...	...	0	...	...	...	...	Jointly
...	...	...	1	...	...	...	...	Locally
...	...	...	...	0	...	...	...	Mandatory entry
					x	x	x	File type:
...	...	...	...	...	0	1	0	Linear EF, records constant length
...	...	...	...	...	1	0	0	Linear EF, records variable length
...	...	...	...	...	1	1	0	Cyclic EF

Byte 2:

This byte is always coded with the value ´41´. This value represents the setting of a logical OR function for write operations and the setting of one byte per data content.

Bytes 3 and 4:

These two bytes - MSB (byte 3) and LSB (byte 4) - specify the maximum length of the records. Values between ´00 01´ and ´00 FF´ may be specified since a record can be a maximum of 255 bytes long.

In the case of formatted data fields with records with a constant length this specification corresponds exactly to the record length.

Byte 5:

The fifth byte specifies the maximum possible number of records in the data field. It can assume values between ´01´ and ´FE´.

Formatted data fields with records with a constant length can accommodate precisely the specified number of records.

3.9.1.2 File Identifier (Tag ´83´)

The file identifier (FID) must be present for all directories (DF) and data fields (EF).

In this chapter, page 29 explains in detail the procedure for assigning long identifiers to these two bytes.

3.9.1.3 Name of a DF (Tag ´84´)

Each directory (DF) in the chip card must be given a name. A name is binary coded and is between 1 and 16 bytes long.

If the name is the name of an application (AID), it must be at least 5 bytes long.

3.9.1.4 Memory Space (Tag ´85´)

The entries for the memory space differ for directories (DF) and data fields (EF).

In the case of directories the two bytes contain a specification of how much memory space is still available in the entire file system for creating files.

⚠ Users must note that, when creating a file, not only the user data but also the administration data must be saved.
It is thus possible that a file cannot be created despite the fact that there is sufficient memory space for user data available.

In the case of data fields the two bytes contain the number of bytes which can be occupied by the user data. Values between ´00 00´...´FF FF´ are permitted. In the case of linear data fields with records of a constant length and of cyclic data fields, the memory space results from the record length and the number of records as specified in the file descriptor.

3.9.1.5 Short File Identifier (Tag ´88´)

The name of a data field, the short file identifier (SFI), is coded in the upper five bits of this byte.

b8	b7	b6	b5	b4	b3	b2	b1	Description
x	x	x	x	x	0	0	0	Short file identifier

The SFI can assume values between 1 and 30.

The value ´00000´ in commands references the data field that is currently selected. The value 31 is not permitted.

3.9.1.6 Life Cycle Status Integer (Tag ´8A´)

The **operational state** of the file is displayed in this byte. The following applies:

b8	b7	b6	b5	b4	b3	b2	b1	Description
								Operational state:
0	0	0	0	0	1	0	1	Activated
0	0	0	0	0	1	0	0	Deactivated

The data of a deactivated file cannot be accessed.

3.9.1.7 ARR per Transmission Type (Tag ´A1´)

The ARR per transmission type is a compound data object with the tag ´A1´. The access conditions are defined in this data object.

Exact details of coding can be found in the chapter "Security Features" on page 132.

3.9.2 File Management Data

The file management data (FMD) is a compound data object with the tag ´64´.

MICARDO does **not** evaluate the information specified in this data object.

If it contains no information, this data object is coded as follows:

T	L	V
´64´	´00´	(omitted)

Data Fields, Passwords and Keys

4

4.1 Overview of Data Fields

Each data field (EF) has a file identifier (FID). The chapter "Structure", section "Identifiers" contains a detailed description of how these identifiers are structured and how they are assigned.

The following table lists the identifiers that are already reserved. These may not be used in your own applications as the contents of these data fields are evaluated by MICARDO.

FID	Data field	Create type	Create location
´00 10´	EF_Key	Optional	MF and each DF
´00 12´	EF_Pwd	Optional	MF and each DF
´00 13´	EF_KeyD	Optional	MF and each DF
´00 15´	EF_PwdD	Optional	MF and each DF
´00 16´	EF_PwdC	Optional	MF and each DF
´00 30´	EF_Rule	Mandatory	MF and each DF
´00 33´	EF_SE	Optional	MF and each DF
´2F E4´	EF_TIN	Optional	MF
´2F 01´	EF_ATR	Mandatory	MF
´3F FF´	Relative path	-	Reserved (ISO)
´FF FF´	RFU	-	Reserved (ISO)

The two data fields EF_Rule and EF_SE are described in detail in the chapter "Security Features".

4.2 Basics about Password and PIN

As MICARDO does not perform a quality check, **only** the external world is responsible for the choice of suitable passwords and PINs. To achieve sufficient security, the PIN must have **six** digits at least and the retry counter should be set to **three**.
Furthermore, the transmission of passwords to the chip card should always be performed with secure messaging.

4.2.1 General

The PIN (personal identification number) and the password are two ways of identifying users. By entering this secret code or word the card user proves his identity to the chip card, i.e. that he is an authorised user. This process is referred to in the following as cardholder authentication.

The following section uses both PIN and password together under the term password. Although MICARDO never stores the password itself but only a reference value, we will also refer to the term password here.

Where it is necessary to **distinguish** between PIN and password or reference value, this will be referred to explicitly and described in detail.

The effectiveness of the cardholder authentication is mainly based on the secrecy of the password.
In the external world, alone the cardholders have to take the responsibility that the password is not known to anybody else.

The following measures are recommended:
- passwords should be stored safely in the chip card (protection by access rules)
- passwords should be kept safely by the cardholder
- data input in terminals should be performed in an unobserved way

- passwords should regularly be changed by the cardholder (old passwords should not be used repeatedly)
- in case someone was able to spy out the password, the cardholder should change the password immediately

Each password is assigned unambiguously to a data field by storing it in the data field **EF_Pwd**, which is contained in the relevant directory.

The passwords contained in the MF are referred to as **global passwords**. If a directory which is not the MF contains passwords, these are referred to as **DF-specific passwords**.

Additional information must be assigned to the passwords, with the following defined for each password,

- how the stored reference value is computed from the password,
- what the minimum length of the password must be,
- what access rules must be applied when executing the commands for cardholder authentication and for resetting the retry counter,
- in which transmission format the password must be presented at the interface to the chip card in order to perform cardholder authentication.

The definition of the procedure for computing the reference value and the minimum length of the password apply irrespective of the active *security environment* (SE). ARR per transmission type and transmission format are defined specifically for the SE.

The additional information for the passwords of a directory is stored in the data field **EF_PwdD**.

The data field EF_PwdD can be declared a **jointly** usable or only **locally** usable data field by defining the file type in the file header (data object: file descriptor). If the data field EF_PwdD is set as only locally usable, all the passwords of the data field are local passwords.

In addition each password of a directory is assigned

- a retry counter and its initial value
- an optional usage counter with its initial value
- an optional reference to an additional password for resetting the retry counter.

This data is stored in the data field **EF_PwdC**.

The following table again gives an overview of the different data fields and their contents.

EF_Pwd ---------	Password - Password reference value - PIN reference value
EF_PwdD -------	Additional information - Computation method - Minimum length - ARR per transmission type per *security environment* - Transmission format per *security environment*
EF_PwdC -------	Counter information - Retry counter - Usage counter (optional) - Password reference for resetting the retry counter(optional)

4.2.2 Password Reference

Passwords are referenced within a data field with a password number (PwdID). This password number is stored in the data field EF_PwdD and can currently assume values between 0 and 3. A PwdID may only occur once, the data field EF_PwdD can therefore contain a maximum of four records.

A record number from data field EF_Pwd under which the required password is stored is assigned to a password number. One password can be referenced by different password numbers.

Example:	Password number	Record number
	00	02
	01	02

A password reference is always two bytes long and has the following structure in the different data fields:

EF	Meaning	Representation
EF_PwdD	The first byte specifies the password number, the second byte the record number	PwdID || Rec.No.
Access rule in EF_Rule	The first byte specifies the search type: ´00´ for global search ´80´ for EF-specific search the second byte specifies the password number	´X0´|| PwdID

4.2.3 Storage Format of Password and PIN

The storage format is defined by the data object with the tag ´89´ in the data field EF_PwdD. This data object defines:

– the format of PIN or password
– how the reference value stored in the record of the EF_Pwd is generated
– the minimum length of the PIN or the password

4.2.3.1 Generating Reference Values for PINs

A PIN may consist of 4 to 12 decimal digits. The minimum length is the minimum number of digits that must be used.

To generate the reference value, the 8-byte format-2 PIN block is generated from the PIN as follows:

C	L	P	P	P	P	P/F	P/F	P/F	P/F	P/F	P/F	P/F	P/F	F	F

Each field represents a half-byte:

C	Control field, binary coded	Has the value ´2´
L	PIN length, binary coded	Possible values from ´4´ to ´C´
P	PIN number, BCD coded	Permitted values from 0 - 9
F	Filler, binary coded	Has the value ´F´
P/F	PIN number/filler	Assignment depends on PIN length

The reference value that is to be saved is generated from this PIN block (PB) through DES encryption with itself.

```
PIN reference value = eKpb(PB), with Kpb = DES key = ParityAdjustment(PB)
```

4.2.3.2 Generating Reference Values for Passwords

A password can consist of 6 to 8 ASCII characters. The minimum length represents the minimum number of characters that must be used.

If the password consists of fewer than 8 ASCII characters, it is padded to 8 bytes with ´00´ bytes. From the block thus created, the reference value is generated again with itself using DES encryption, possibly after a parity adjustment for usage as a key.

4.2.3.3 Coding the Storage Format

The storage formats that are available are represented as a sequence of natural digits with the exception of the value 0.

1: Format for PINs	1: DES encryption of the format-2 PIN block with itself	4: Minimum length 4 digits
		... ...
		12: Minimum length 12 digits
2: Format for pass-words	1: DES encryption with itself of the password padded with ´00´ to 8 bytes	6: Min. length 6 characters
		7: Min. length 7 characters
		8: Min. length 8 characters

The permitted maximum length of the PIN or of the password results from the definition of the storage format, currently 12 PIN digits or 8 password characters.

The storage formats for PINs are therefore coded as follows: ´11 40´ to ´11 70´ for PIN lengths between 4 and 7 digits and ´11 90´ to ´11 94´ for PIN lengths between 8 and 12 digits.

The coding of storage formats for passwords is equivalent: ´21 60´ and ´21 70´ for password lengths of 6 and 7 characters and ´21 90´ for a password length of 8 characters.

4.2.4 SE Data Object for Passwords

The additional information for a password, ARR per transmission type and transmission format, is defined by the compound data object with the tag ´7B´ in the data field EF_PwdD.
This data object is also referred to as SE data object. It defines the following for the password numbers per *security environment*:

– which access rules the commands **VERIFY, CHANGE REFERENCE DATA** and **RESET RETRY COUNTER** must comply with and
– in which transmission format the password must be passed to the commands **VERIFY**, **CHANGE REFERENCE DATA** and **RESET RETRY COUNTER**.

If different *security environments* are to be defined for one password number, the record of the EF_PwdD contains several different SE data objects.

4.2.5 Transmission Format

The transmission format of the password is defined with tag ´89´ in the SE data object for passwords in the data field EF_PwdD.

4.2.5.1 Transmission Format for PINs

A PIN can be transferred in the following ways:

– in a format-1 PIN block
– in a format-2 PIN block
– as a string of BCD digits, possibly padded with ´F´ to the full byte length
– as a string of ASCII digits

Format-1 PIN block:
The 8-byte format-1 PIN block is generated from a PIN with 4 to 12 decimal digits as follows:

C	L	P	P	P	P	P/Z	P/Z	P/Z	P/Z	P/Z	P/Z	P/Z	P/Z	Z	Z

Each field represents a half-byte:

C	Control field, binary coded	Has the value ´1´
L	PIN length, binary coded	Possible values from ´4´ to ´C´
P	PIN digits, BCD coded	Permitted values from 0 - 9
Z	Random value, binary coded	Has a random value between ´0´ and ´F´
P/Z	PIN digits/random value	Assignment depends on the PIN length

Format-2 PIN block:
The format-2 PIN block has already been described in the section chapter "Generating Reference Values for PINs" on page 45.

BCD coded transmission:
If the PIN consists of an even number 2*<n> of digits, the PIN is BCD coded in <n> bytes.
If the PIN consists of an odd number 2*<n>-1 of digits, the PIN is coded in <n> bytes, by setting the PIN BCD coded left-adjusted in <n> bytes and the last half-byte is padded with ´F´.

ASCII coded transmission:
In this case <n> PIN digits are set in <n> bytes ASCII coded (values ´30´ to ´39´).

4.2.5.2 Transmission Format for Passwords

A password can currently only be transmitted as ASCII code. Here <k> password characters are set in <k> bytes of ASCII code.

4.2.5.3 Coding the Transmission Format

The available transmission formats are represented as a sequence of natural numbers with the exception of the value 0.

1: Format for PINs	1: Format-1 PIN block
	2: Format-2 PIN block
	3: BCD coded
	4: ASCII coded
2: Format for passwords	1: ASCII coded

4.2.6 Storing Counters

The following values are stored in a record of the data field EF_PwdC for a PIN or a password:

– the current value of the retry counter and its initial value
– optionally the current value of the usage counter and its initial value
– optionally the reference to a password for resetting the retry counter

The record numbers from the data field EF_PwdC must match those in the data field EF_Pwd.

4.2.6.1 Retry Counter

Following successful cardholder authentication, the retry counter of the corresponding password is always reset to its initial value.

The retry counter is decremented by one every time a cardholder authentication process with the relevant password contains an error or is aborted. When this counter reaches the value ´00´, the password is disabled for any further cardholder authentication activities.

The retry counter can be reset to its initial value using the **RESET RETRY COUNTER** command. To prevent unauthorised retry counter resets, the usage of this command is controlled by access rules.

4.2.6.2 Usage Counter

Optionally a usage counter and its initial value can be assigned to a password.

A password should be assigned a usage counter to controll the frequency of use to prevent it from being attacked.

MICARDO itself provides additional objects so that the following values are relevant for full functionality:

1. Initial value of the usage counter
   This start value is entered when the record is created.
2. Minimum value of the usage counter
   The default minimum value is ´00´.
3. Maximum value of the usage counter
   The default maximum value is ´FF´.
4. Current value of the usage counter
   The current value is read from the data field EF_PwdC.
5. Start value of the usage counter
   The start value is entered as current value when the record is created.

If the current value is the same as the minimum value, cardholder authentication will not be possible, i.e. the user will not be able to authenticate himself with the corresponding password.

MICARDO changes the usage counter according to the following rules:

1. If the current value is smaller than the maximum value and larger than the minimum value, it is decremented by one each time a cardholder authentication is successful.
2. If the current value is equal to the maximum value, it is not changed.
3. If the relevant password is changed during the **CHANGE REFERENCE DATA** or **RESET RETRY COUNTER** command, the current value is set to the initial value.

The following list shows which effect the different settings of the usage counter of a password have on the cardholder. Possible settings are:

1. The password has no usage counter or its start value is equal to the maximum value.
   In this case MICARDO never forces the cardholder to change his password.
2. The password has a usage counter whose start value is equal to the minimum value.
   In this case MICARDO forces a password change before successful cardholder authentication can take place.
3. The password has a usage counter whose start value lies between the minimum and maximum values.
   In this case, after a certain period of use, the cardholder is forced to change the password.
4. The initial value of the usage counter is equal to the maximum value.
   In this case the cardholder may be forced to change his password only once.

4.3 EF_Pwd

The data field EF_Pwd contains the reference values for the PINs and the passwords. They are referenced in the data field EF_PwdD by the record number.

Typical security conditions for EF_Pwd:

FID: ´00 12´	**Data field type:**	**linear, constant length**
Security conditions	UPDATE	: Personalisation office
	READ / SEARCH	: NEV
	DELETE	: NEV
	APPEND	: Set by user
	ACTIVATE	: Set by user
	DEACTIVATE	: Set by user

Data content: records - 9 bytes in length		
Byte	**Designation**	**No. of bytes**
1	Binary coded length of PIN Values: ´04´...´0C´ or Binary coded length of the password Values: ´06´...´08´	1
2...9	Binary coded reference value	8

Explanations:

The length entered in the first byte is only relevant in connection with the commands **ChangeReferenceData** or **ResetRetryCounter** if transmission is to take place in BCD or ASCII format. The length must assume a value which enables the two passwords to be separated correctly within the command data.
In all other cases the length specification is irrelevant.

4.4 EF_PwdD

The data field EF_PwdD contains the additional information on the password.

Typical security conditions for EF_PwdD:

FID: **´00 15´**	**Data field type: linear, variable length**	
Security conditions	UPDATE	: Personalisation office
	READ / SEARCH	: ALW
	DELETE	: NEV
	APPEND	: Set by user
	ACTIVATE	: Set by user
	DEACTIVATE	: Set by user

Data content: 2 simple, 1 compound data object		
Tag	**Length**	**Value**
 ´84´ ´94´ ´83´ ´93´	´01´ ´02´	Password reference: Password number as password reference to the EF_PwdD of the MF (not permitted in the EF_PwdD of the MF) For jointly usable password reference or only for locally usable password reference or Password number (1st byte) and record number (2nd byte) from the corresponding EF_Pwd For jointly usable additional information or only for locally usable additional information
´89´	´02´	Storage format of the password (for explanations see page 45)
´7B´	´XX...YY´	Defines the ARR per transmission type and the transmission format, both per *security environment* for the password - SE data object for passwords

Explanations:

The following data objects may be present:

– Password reference (must be present)
  defines jointly usable password or locally usable password
– Storage format
– Additional information (exists at least once if required)
  defines jointly usable additional information or locally usable additional information

The following table gives an overview of the relationships between these data objects:

	Data object Password reference	Data object Storage format	Data object Additional information
Tags	´84´ joint	-	-
Tags	´94´ local	-	-
Tags	´83´	´89´	´7B´ joint
Tags	´93´	´89´	´7B´ local

The following applies:

– If the password reference points to the MF (tag ´84´ or ´94´) the record contains no further data objects.
– A data object with tag ´89´ and at least one data object with tag ´7B´ are present if the password reference does not point to the MF.
  If different ARRs per transmission type or different transmission formats were defined for the password in different *security environments*, several data objects with tag ´7B´ are created.

The different records of an EF_PwdD can refer to the same password in EF_Pwd with different PwdID. To do this, the first byte of the relevant password references contains a different PwdID and the second byte the same record number.

Example:	T	L	V
	´83´	´02´	´00 01´
	´83´	´02´	´03 01´

4.4.1 SE Data Object for Passwords

The SE data object for passwords is a compound data object with the tag ´7B´.

Tag	Length	Value
´80´	´01´	Binary coded SE# with a value between ´00´ and ´FE´ except ´EF´
´A1´	´XX...YY´	ARR per transmission type
´89´	´01´	Transmission format

If the first data object (also referred to as SE reference) has the value ´00´, further definitions apply to all SEs except those SEs explicitly listed in any other data objects present.

4.4.1.1 ARR per Transmission Type (Tag ´A1‘)

The ARR per transmission type is a compound data object with the tag ´A1´.

The ARR per transmission type is evaluated by the commands **VERIFY**, **CHANGE REFERENCE DATA** and **RESET RETRY COUNTER** when the password is accessed in order to find the access rules which must be complied with for the active *security environment*.

The access rules are applied by the above commands when the SE which is referenced in the first data object is active. If a *security environment* whose SE# is not referenced is active, the commands must not be executed.

The coding of this data object can be found in the chapter "Security Features" on page 132.

4.4.1.2 Transmission Format (Tag ´89´)

This simple data object with the tag ´89´ defines the transmission format for the password or for the PIN. A detailed description can be found on page 47.

4.5 EF_PwdC

The data field EF_PwdC contains the counter information on the password or the PIN.

Typical security conditions for EF_PwdC:

FID: ´00 16´	Data field type: linear, fixed length	
Security conditions	UPDATE	: Personalisation office
	READ / SEARCH	: ALW
	DELETE	: NEV
	APPEND	: – / –
	ACTIVATE	: NEV
	DEACTIVATE	: NEV

Data content: 2 (- 5) simple data objects		
Tag	**Length**	**Value**
´80´	´01´	Binary coded initial value for the retry counter
´90´	´01´	Binary coded retry counter
´81´	´01´	Initial value for the usage counter, default value = ´FF´ (optional)
´91´	´01´	Usage counter (current value), default value = ´FF´ (optional)
´83´	´02´	Password reference to reset the retry counter (optional) (see section “Password Reference” on page 44)

Explanations:

The data objects must be specified in the sequence shown, whereby optional data objects may be missing.

The password reference is used when the command **RESET RETRY COUNTER** is called with resetting mode.

4.6 Basics about Keys

⚠ The security of the procedures where keys for authenticaton procedures or cryptographic algorithms are used is based on the secrecy of keys.
In the external world alone the users of the keys are responsible for the secrecy of keys.

The following measures are recommended:

- data fields for keys have to be protected against reading and modification (protection by access rules)
- in case someone was able to spy out the key, it should be changed or blocked immediately
- the transmission of keys to the chip card should always be performed with secure messaging.

4.6.1 Security Features

MICARDO uses the following security features, which are based on DES or RSA:

– Authentication procedures
– Cryptographic algorithms
– Key management
  - Key derivation
  - Key negotiation

Detailed information on the individual procedures can be found in the chapter "Security Features".

The different security features operate with keys. In order to prevent key abuse, a definition must be created for each key, stating the security features for which it may be used. The security features can be assigned to a key as follows:

– Authentication procedures only,
– Cryptographic algorithm only,
– First key derivation, then cryptographic algorithm,
– First key derivation, then authentication procedures,

– First key negotiation, then cryptographic algorithm,
– First key derivation, then key negotiation, then cryptographic algorithm.

⚠ To achieve sufficient security and strength of the used security features, DES3 respectively RSA with a key length of 1024 bit at least should be used.

4.6.2 Key Characteristics

Keys can be distinguished by certain characteristics.

Usability:

Directly usable key
A directly usable key is a key which can be used directly for cryptographic algorithms or authentication procedures.
If it is non-volatile stored, this key is filed in the relevant data field EF_Key.
A volatile stored key, which is the result of a key management procedure, is stored in RAM.
Master key
A master key is a key to which a key derivation procedure is assigned.
A master key is always a non-volatile stored key.

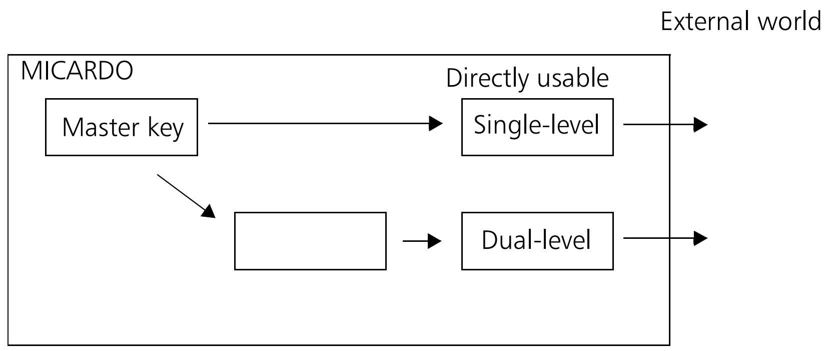
Negotiation key
A negotiation key is a key which is used to negotiate another key. A key negotiation procedure is therefore assigned to this key.
A negotiation key is either a non-volatile stored key or a volatile stored key, which is the result of a key derivation procedure.

Key management levels:

Single-level key management key
A key is referred to as a single-level key management key if a key that is derived from it or negotiated with it is directly usable.

Dual-level key management key
A key is referred to as a dual-level key management key if the keys derived from it are negotiation keys.

The following illustration shows the key management levels using the example of a derivation from the master key.



Cryptographic algorithm membership:

DES key
A DES key is a secret, symmetrical key used to execute the DES cryptographic algorithm.

RSA key
A RSA key is either the public or private key part of a key pair used to execute the RSA cryptographic algorithm.

4.6.3 Data Fields for Keys

Each non-volatile stored key is uniquely assigned to a directory by storage in the data field **EF_Key**, which is contained in the relevant directory.

A directory can contain several different EF_Key data fields.

The keys which the MF contains are referred to as **global keys**. If a directory which is not the MF contains keys, these are referred to as **DF-specific key**s.

Additional information must be assigned to the keys, defining the following for each key,

– what type of key it is,
– how it may be used,

– whether an optional retry counter or usage counter is used.

Apart from the optional counters, all the additional information is defined for individual SEs. It is stored in the data field **EF_KeyD**.

The data field EF_KeyD can be declared a **jointly** usable or only **locally** usable data field through the file type definition in its file header (data object: file descriptor). If it is only set for local use, all the keys of the data field are local keys.

4.6.4 Storing Keys

As a rule, symmetrical keys are stored with the FID ´00 10´in the data field EF_Key. However, it is also possible to distribute these keys over several EF_Key data fields with other FIDs.

Asymmetrical keys are stored in special EF_Key data fields. There are two different types:

– Data fields for information on the public part of a key pair
– Data fields for information on the private part of a key pair
.

Symmetrical	Asymmetrical key	
Data field type: linear, variable	Data field type: transparent	Data field type: transparent
Content: DES key	Content: public part (RSA)	Content: private part (RSA)
FID usually: ´00 10´	FID: freely selectable	FID: freely selectable

The specifications in the section chapter “Identifiers” on page 28 apply to freely selectable FIDs.

4.6.5 Key Numbers and Key Versions

A key always has a key number (KID) and a key version (KV). The following applies:

Key	Key number	Key version
Symmetrical, secret (g)	KIDs ´01´...´FE´	KVs ´00´...´FE´
Asymmetrical, private part (pr)	KIDpr ´01´...´FE´	KVpr ´00´...´FE´
Asymmetrical, public part (pu)	KIDpu ´01´...´FE´	KVpu ´00´...´FE´

All keys in MICARDO can thus be referenced via two bytes. In the following we will therefore refer in general to key numbers (KID) and key versions (KV) when statements refer to all three key types.

The keys of a directory with the same KID form a **key group**. If a key number occurs only once in a directory, it is referred to as a **single key**.

Key groups are used to restrict the spread of master keys in the external world. The external world only needs one key of the group in order to communicate with various chip cards.

Example:	External world	Chip card 1	Chip card 2	Chip card 3
	KID KV	KID KV	KID KV	KID KV
	09 02		09 02	
	09 03	09 03		
	09 04	09 04		09 04
	09 05			09 05

4.6.6 Key Reference

The key reference gives a key a unique reference.

Within the data field EF_KeyD the key reference can assume the following lengths and values:

Length	Meaning	Representation
2	The first byte contains the key number The second byte indicates the key version	KID || KV

4	First and second bytes as for length 2 The third and fourth byte contain the file identifier of the file in which the key is to be found	KID || KV || FID

Key references are also used outside the data field EF_KeyD. A key reference can then be coded as follows:

Length	Meaning	Representation
1	A key version is specified	KV
2	The first byte indicates the search type: ´00´ for global search ´80´ for DF-specific search The second byte indicates the key number	´X0´|| KID
3	First and second bytes as for length 2 The third byte indicates the key version; the value ´FF´ for KV is a wildcard representing any key version	´X0´|| KID || KV

4.6.7 Retry Counter

A key can have a retry counter assigned to it.

To prevent the key from being attacked, it should be assigned a retry counter, which controls misuse.
This retry counter cannnot be reset by a command. Unauthorized resetting in the data field EF_KeyD should be excluded by using access rules (update of the data field not allowed).

The value of the retry counter is decremented by one when a function which operates with this key executes **incorrectly** and MICARDO detects a security violation. It is also decremented when a command which uses the above function is aborted.

The key is disabled for all further use when the retry counter reaches the value zero.

The length and initial value of the retry counter are used to define how many retries MICARDO permits.

The maximum length of the retry counter is 3 bytes = 24 bits. This corresponds to a maximum counter status of approximately 16 million. This value exceeds the guaranteed number of write cycles of today's non-volatile memory by more than one order of magnitude and should therefore be considered sufficient.

4.6.8 Usage Counter

A usage counter can be assigned to a key.

The value of the usage counter is decremented by one when a function operates **correctly** with this key. It is also decremented if a command that accesses this key is aborted.

The key is disabled for further use when the usage counter reaches the value zero.

The length and initial value of the usage counter are used to define how many operations are permitted. The maximum length of the usage counter is 3 bytes = 24 bits.

4.7 EF_Key

A directory can contain several EF_Key data fields. The following applies:

Data field	FID	Data field type
EF_Key for DES keys (EF_Key_DES)	´00 10´ (default)	Linear
EF_Key for the private parts of a RSA key pair (EF_Key_RSA_Private)	Freely selectable	Transparent
EF_Key for the public parts of a RSA key pair (EF_Key_RSA_Public)	Freely selectable	Transparent

The FID is freely selectable according to the specifications in the section chapter "Identifiers" on page 28.

Typical security conditions for EF_Key_DES and EF_Key_RSA_Private:

FID: see above	**Data field type: see above**	
Security conditions	UPDATE	: Personalisation office
	READ / SEARCH	: NEV
	DELETE	: NEV
	APPEND	: Set by user
	ACTIVATE	: Set by user
	DEACTIVATE	: Set by user

Typical security conditions for EF_Key_RSA_Public:

FID: see above	**Data field type: see above**	
Security conditions	UPDATE	: Personalisation office
	READ / SEARCH	: ALWAYS
	DELETE	: Set by user
	APPEND	: – / –
	ACTIVATE	: Set by user
	DEACTIVATE	: Set by user

4.7.1 EF_Key_DES

Data content: records - 10 or 18 bytes long		
Byte	**Designation**	**No. of bytes**
1	Binary coded key number KID Values: ´01´…´FE´	1
2	Binary coded key version KV Values: ´00´…´FE´	1
3...10 3...18	Binary coded DES key, or Binary coded DES3 key Values: ´XX...YY´	8 or 16

The key reference, KID || KV, must be unique within a directory. The requirement for it to be unique is directed at the instance which defines this data. MICARDO does not detect the existence of double entries.

The length of the keys must be consistent with the definition contained in the EF_KeyD of the directory.

4.7.2 EF_Key_RSA_Private

The data field is transparent and contains data objects with a fixed length which result from key generation with the Chinese remainder theorem.

The following tables list the data objects stored in the data field in the sequence stated:

Data content: 5 simple data objects		
Tag	**Length**	**Value**
´81´	´44´	Bit length p (2 byte) ‖ prime factor p ‖...
´82´	´42´	Bit length q (2 byte) ‖ prime factor q ‖...
´83´	´44´	Bit length dp (2 byte) ‖ reduced exponent dp = d mod (p-1) ‖...
´84´	´42´	Bit length dq (2 byte) ‖ reduced exponent dq = d mod (q-1) ‖...
´85´	´44´	Bit length c (2 byte) ‖ coefficient c = q^{-1} mod p ‖...

The two bytes for the length represent the high (first byte) and the low byte (second byte) of a number, which contains the length of the following value in bits, e.g. a length of 33 bytes are 264 bits and 108 hexadecimal.

If the generated result is shorter than the expected length, it is set left justified and padded with random bytes.

Example of an EF_Key_RSA_Private corresponding to the EF_Key_RSA_Public example a:

```
81 44   0108 03CBCE749FDAD21031FFA51B82D58B38
             3B7226E4CECBA6348573597D1B6083CE
             15000000000000000000000000000000
             000000000000000000000000000000000000
82 42   0100 244BEF0D5FE5A8781D3E835530202586
             D804361E86EC534E521E11321CA60F5D
             00000000000000000000000000000000
             00000000000000000000000000000000
83 44   0108 030C94E36162CE9F54A604E97F47B85F
```

```
                C5421F8415BADBD0967B1C1B0463B577
                E3000000000000000000000000000000
                000000000000000000000000000000000000
84 42    0100 1CF941BB78B61EC63D71207B0FD3E433
                B35D913FA4E11A024979BE9FAE4BCAEB
                00000000000000000000000000000000
                00000000000000000000000000000000
85 44    0108 02E72822BBF735260240076215F3AB01
                A6ECCDFC85378041EFEC7EAA949497B6
                0C000000000000000000000000000000
                000000000000000000000000000000000000
```

Example of a EF_Key_RSA_Private corresponding to the EF_Key_RSA_Public example b:

```
81 44    0208 059C8677EBA8D7FFF79E1D7FB04715CF
                BDE612F4861D054F9A713757EA6AA709
                C8EE956D596FEC8B9734DFF62A70D8F6
                2197A679F9A0A0068682AC5CD7E7018549B00
82 42    0200 1E84DC934B89E9A1DE809FF28E5E988D
                1AC139FBF71390E802B6F146A25107AA
                0195B5660B93A082744CE8A6AB65F77A
                38BC35C26290D8A7EB9D9E7E384A4D5B
83 44    0208 02B4445ECE9790F3F1571791A5D2F4DC
                F0AB2605AA131755764AA43BFF7F2F0D
                470FDC9E03AAC839060B7489C501DFDF
                9F4C789EFD1A450D5FD7724830EC268FE900
84 42    0200 15952FB42640019102E506B91E0D9355
                9BED54E5B6AF3943561998924B6FB757
                D64F1CCCFF137025155B2CB846F3C3B4
                1F32ECDF96294FEBF04E72FB797F94FF
85 44    0208 0269E7360627924484368D40A6AA6A5C
                C717C1E6C70F37A09D8164E1854A2A4F
                CB948DB593D598912FBEBCB430BB2F03
                F01364C2709636DF5155FB9B8C7170BD3900
```

4.7.3 EF_Key_RSA_Public

The data field is transparent and contains the digital signature template (DST).

This template is a compound data object with the tag ‘B6’ and contains the public part of the RSA key pair and the information needed for key generation.

The following tables list the data objects stored in the data field in the sequence stated:

Data content: 3 simple and 1 compound data object		
Tag	**Length**	**Value**
´83´	´XX...YY´	Key name, e.g. register number
’7F 49’	’XX...YY’	Data object with key parameters (see below)
´80´	´XX...YY´	Algorithm ID of the following signature (SHA-1 with DSI according to PKCS#1)
‘9E’	´XX...YY´	Digital signature of the public part

The data object with the tag ’7F 49’ contains the following key parameters:

Tag	Length	Value
´80´	´XX...YY´	Algorithm ID
´81´	´XX...YY´	Modulus n
´82´	´XX...YY´	Public exponent e

Example a of an EF_Key_RSA_Public with a modulus length of 512 bit:

```
B6 8197
  83 04 01234567
  7F49 49
    80 01 02
    81 40 89C9495B2A013BB719E893AA6F8A18B3
          01D35C3854DF0E5A16B8C0FDF5A8FE81
          F1D132772C8A62E10E3AA08C8D5F3849
          5245505563D9DD8FBF09B2B5B69318A1
    82 02 C413
  80 01 11
  9E 40 337281FC35EA48FBF9E0A1672C5B72CD
        C19D0B2A30D9B7D1993F9C9EA2C5833F
        91DDF6C5F40146D3B48C499F82371B27
        7E1F3405A9F941DF2770994CA37D2A6C
```

Example b of an EF_Key_RSA_Public with a modulus length of 1024 bit:

```
B6 82011A
  83 04 01234567
  7F49 818A
    80 01 02
    81 8180 AB414D218B7308E5800FAF77DF497F48
            F0D26A3D6CF3E146E2A32C9A28B3453C
            356DD682D3CFBD15FF0D7492B5BD036B
            4CC44130156400DD41B4A35F3E7EB4AC
            E006735E33FF6E7D7C9B9B358361C56C
            108C3F2E2ECBC5A847FEB7DBB5E8C987
            CC8D06FE1B0A22B65217C3AC944AEA0F
            2693D6EE73164F231E711B7D86E6B219
    82 02 5AC5
  80 01 11
  9E 8180 0862BC8DEA405A6BCB319B5662D6A4B6
          0D0ED8AB00278E8DFBB7F746E1768111
          8508184FDAE800E33D175685A96B8974
          2D4D4EB532016210627128F4156AACCB
          6BF00B0EF879E437357DC66495C9C613
          47B273A5C10F33219D65BB872B99F5BF
          E3689421E081312891C2D4517AEFDBFC
          96303EA24885A0CBBAAFDF49787A0C58
```

4.8 EF_KeyD

The data field EF_KeyD contains the additional information for the key.

Typical security conditions for EF_KeyD:

File ID: ´00 13´	Data field type: linear, variable length	
Security conditions	UPDATE	: Personalisation office
	READ / SEARCH	: ALW
	DELETE	: NEV
	APPEND	: Set by user
	ACTIVATE	: Set by user
	DEACTIVATE	: Set by user

Data content: 2 (- 4) simple, 1 compound data object		
Tag	**Length**	**Value**
´83´ ´93´	´02´ ´04´	Key reference: KID || KV of EF_Key (FID ´00 10´) or KID || KV || FID of the corresponding EF_Key For additional information for joint use or for additional information that can only be used locally
´CX´	´02´	Key structure
´90´	´01´...´03´	Binary coded retry counter (optional)
´91´	´01´...´03´	Binary coded usage counter (optional)

Data content: 2 (- 4) simple, 1 compound data object		
Tag	**Length**	**Value**
´7B´	´XX...YY´	Definition of key usability - SE data object for key

Explanations:

The data field EF_KeyD may contain a maximum of 254 records, which means that a maximum of 254 keys can be stored.

The sequence of data objects must be complied with, although optional data objects may be omitted.
If different security features are to be defined for different *security environments*, the SE data object contains several data objects with the tag ´7B´.

The following key structures are possible:

Tag	Value
´C0´	Directly usable key
´C1´	Key for single-level key management
´C2´	Key for dual-level key management

The value field is always 2 bytes long. The first byte always contains ´81´. The second byte specifies

– The length of the key for DES keys. This is either eight for simple DES or sixteen for DES3.
  Examples: ´CX 02 81 08´ or ´CX 02 81 10´
– The length of the modulus n in bytes for RSA keys.
  Examples: ´CX 02 81 60´ for 768 bit or ´CX 02 81 80´ for 1024 bit

If the data objects with tags ´90´and ´91´are omitted, the key has no retry counter and no usage counter.

4.8.1 SE Data Object for Keys

This data object defines the security features which the keys support for the key itself and for keys which are derived from the key or negotiated with the key.

The SE data object is a compound data object with the tag´7B´.

The SE reference data object (tag ´80´) must always be in first position. If the data object ARR per transmission type (tag ´A1´) is present, it must be in second position. The sequence of the other data objects is variable.

Tag	Length	Value
´80´	´01´	Binary coded SE# with a value between ´00´ and ´FE´ except ´EF´ (SE reference)
´A1´	´XX...YY´	ARR per transmission type
´A4´	´XX...YY´	CRT for authentication (AT)
´B4´	´XX...YY´	CRT for data authentication (CCT)
´B6´	´XX...YY´	CRT for digital signature (DST)
´B8´	´XX...YY´	CRT for data confidentiality (CT)
´BA´	´XX...YY´	Key management template (KMT)

If the first data object (also known as SE reference) has the value ´00´, additional definitions apply to all SEs with the exception of any SEs that are explicitly listed in other data objects present.

Dependencies exist between the data objects; these are described below:

1. A key which is used in an SE for a key management procedure may not be used in the same SE or in another SE directly for a cryptographic algorithm or for an authentication procedure.
   This means that an SE data object with a key management template may not contain CRT data objects.
2. A key may only be used for one key management procedure.
   This means that an SE data object may only contain a maximum of one key management template.
3. A key that is used in an SE to generate digital signatures may not be used in the same SE or in another SE for other purposes.

This means that an SE data object with a CRT for digital signature must not contain any other templates.

4. If the value field of an SE data object contains a key management template in a record of the data field EF_KeyD, no other SE data object in this record may contain a CRT data object or a key management template with a different algorithm ID.
5. If the value field of an SE data object contains a CRT for digital signature in a record of the data field EF_KeyD, no other SE data object in this record may contain a CRT data object or a key management template.

MICARDO only checks item 1 and 2.

The SE data object can contain different tags to define that the key has to be generated before using it. In this case, bit 7 of the CRT has to be set to 1.

The following table shows the tag entries and commands, which uses the generated key.

CRT	Tag	Tag (generation necessary)	Command
AT	'A4'	'E4'	**INTERNAL AUTHENTICATE**
DST	'B6'	'F6'	PSO command: **COMPUTE DIGITAL SIGNATURE**
CT	'B8'	'F8'	PSO command: **DECIPHER**

4.8.1.1 ARR per Transmission Type (Tag ´A1´)

The ARR per transmission type is a compound data object with the tag ´A1´. This data object can occur a maximum of once.

This data object is evaluated by the commands **INTERNAL AUTHENTICATE, EXTERNAL AUTHENTICATE** (also in the version **MUTUAL AUTHENTICATE**) and by the

PSO commands when accessing the key in order to find the access rules which must be complied with for the active *security environment*.

Although the internal *secure messaging* routines also use keys, their access rules are not evaluated within *secure messaging*. From the aspect of access conditions any key can thus be used for *secure messaging*. In case of *secure messaging* keys, the ARR data object with tag 'A1' is unnecessary. In order to enable a key to be used within *secure messaging* it requires suitable CRT data objects.

Coding of this data object is described in the chapter "Security Features" on page 132.

4.8.1.2 CRT for Authentication (Tag ´A4´)

CRT for authentication (AT) is a compound data object with the tag ´A4´. This data object can occur a maximum of twice.

The sequence specified in the table must be complied with, optional data objects (o) may be omitted.

Tag	Length	Value	
´95´	´01´	Usage qualifier, default value = ´C0´	o
´89´	´XX...YY´	Algorithm ID or object ID	
'80'	'01'	DSI format (RFU)	o

If the SE data object contains two CRTs for authentication, various bits must be set in pairs in the usage qualifiers.

4.8.1.3 CRT for Data Authentication (Tag ´B4´)

The CRT for data authentication (CCT) is a compound data object with the tag ´B4´. This data object can occur six times.

The specified sequence of data objects must be complied with, optional data objects (o) can be omitted

Tag	Length	Value	
´95´	´01´	Usage qualifier, default value = ´30´	o

Tag	Length	Value	
´89´	´XX...YY´	Algorithm ID	
´87´	´01´	ICV indicator, default value = ´01´	o
´8A´	´01´	Padding indicator, default value = ´01´	o

If it contains several CRTs for data authentication, various bits must be set in pairs in the usage qualifiers.

If the algorithm ID prescribes the use of an ICV that is not zero, it must be defined which of the available procedures is to be used for ICV handling. The meaning of the ICV indicator value depends on the usage qualifier.

Value	UQ	Possible procedures
´01´	´01´, ´10´	Previously outputs the ICV with the command **GET CHALLENGE**
´01´	´02´, ´20´	Transfers the ICV using the command **MANAGE SECURITY ENVIRONMENT** - SET version or in the *secure messaging* command message
´01´	´03´, ´30´	Previously outputs the ICV using the command **GET CHALLENGE** for the *secure messaging* command message and transfers it with the command **MANAGE SECURITY ENVIRONMENT** - SET version or in the *secure messaging* command message
´00´…´FF´	´40´, ´80´, ´C0´	Transfers the ICV using the command **MANAGE SECURITY ENVIRONMENT** - SET version or in the *secure messaging* command message

If ICV handling is prescribed by the algorithm ID and ICV indicator, the prescription of the algorithm ID has higher priority.

The value field of the data object with tag ´8A´ (padding indicator) defines which padding procedure is to be used for MAC computation.

Value	Meaning
´01´	ISO padding, default value for padding within *secure messaging*

Value	Meaning
´80´	Zero padding, default value within command-specific data authentication

4.8.1.4 CRT for Digital Signature (Tag ´B6´)

The CRT for digital signature (DST) is a compound data object with the tag ´B8´. This data object can occur a maximum of twice.

The specified sequence must be complied with, optional data objects (o) can be omitted

Tag	Length	Value	
´95´	´01´	Usage qualifier, default value = ´80´	o
´89´	´XX...YY´	Algorithm ID or object ID	
‘80’	‘01’	DSI format (RFU)	o

If the SE data object contains two DSTs, various bits must be set in pairs in the usage qualifiers.

4.8.1.5 CRT for Data Confidentiality (Tag ´B8´)

The CRT for data confidentiality (CT) is a compound data object with the tag ´B8´. This data object can occur a maximum of six times.

The specified sequence of data objects must be complied with, optional data objects (o) may be omitted.

Tag	Length	Value	
´95´	´01´	Usage qualifier, default value = ´10´	o
´89´	´XX...YY´	Algorithm ID	
´87´	´01´	ICV indicator, default value = ´01´	o

If the SE data object contains two CTs, various bits must be set in pairs in the usage qualifier.

If the usage qualifier defines that command-specific encryption is involved, additional data objects whose tags and values are defined and evaluated for specific commands may occur.

If the algorithm ID prescribes the use of an ICV which is not zero, it must be defined which of the available procedures is to be used for ICV handling. The meaning of the ICV indicator value depends on the usage qualifier.

Value	UQ	Meaning
´01´	´01´ ´10´	ICV is sent in the command
´01´	´02´ ´20´	ICV is sent in the response
´01´	´03´ ´30´	ICV is sent in the command or in the particular response
´02´	´10´ ´20´ ´30´	ICV identical to that of MAC security within *secure messaging*
´00´…´FF´	´40´ ´80´ ´C0´	Transfers the ICV using the command **MANAGE SECURITY ENVIRONMENT** - SET version or in the *secure messaging* command message

If the algorithm ID and the ICV indicator prescribe ICV handling, the prescription of the algorithm ID has higher priority.

MICARDO does **not** use the key for encryption if data is inconsistent. This occurs when an ICV is to be used that should be identical to that of MAC security (value ´02´) but there is no ICV available for MAC security.

4.8.1.6 Key Management Template (Tag ´BA´)

The key management template (KMT) is a compound data object with the tag ´BA´. This data object can occur a maximum of once.

The specified sequence must be complied with, optional data objects (o) may be omitted.

Tag	Length	Value	
´89´	´XX...YY´	Algorithm ID	
´9F 21´	´01´	Initial value for usage counter, binary	o
´A4´	´XX...YY´	CRT for authentication (AT)	o
´B4´	´XX...YY´	CRT for data authentication (CCT)	o
´B8´	´XX...YY´	CRT for data confidentiality (CT)	o
´BA´	´XX...YY´	Key management template (KMT)	o

The TLV structures of the different CRTs are the same as described in the SE data object for keys. The key management template (KMT) defines the CRTs for derived key.

Tag ´89´, algorithm ID:
Within a KMT one of the algorithm IDs for key management can be assigned to a key. The algorithm ID must be consistent with the type and structure of the key.
When a single-level key management key is involved, all defined algorithm IDs for the key management may occur.
When a dual-level key management key is involved, only algorithm IDs for a key derivation may occur.

Tag ´9F 21´, initial value for usage counter:
When the algorithm ID defines that the key is a master key, an optional initial value can be defined for a usage counter for a key derived from this master key.
This initial value is binary coded in one byte and can assume the value ´FF´ as maximum.

Tag ´A4´, CRT for authentication (AT):
A KMT may only contain a CRT for authentication if the algorithm ID defines that the key which is stored non-volatile in the chip card is a single-level key management key.

Tag ´B4´, CRT for data authentication (CCT):
Only if the algorithm ID defines that the key is a single-level key management key may this data object be present.

If the algorithm ID shows that this is a negotiation key, the algorithm ID defines implicitly the way in which the negotiated session keys SK1 or SK2 for MAC security may be used in *secure messaging*.

1. If only one session key SK1 is negotiated without SSC, this key must be used for command and response MAC security in *secure messaging* for the following commands.
   This SK1 therefore always has the following CCT assigned:
   CCT = ´B4 07 (95 01 30 || 89 02 (12 21))´.
2. If the session keys SK1 and SK2 are negotiated without SSC, SK2 must be used for command and response MAC security in *secure messaging* for the following commands.
   SK2 is therefore assigned the same CCT as SK1 under 1.
3. If only one session key SK1 is negotiated with SSC, this key must be used for command and response MAC security in *secure messaging* for the following commands, whereby the incremented SSC must be used as ICV.
   This SK1 is therefore always assigned the following CCT:
   CCT = ´B4 07 (95 01 30 || 89 02 (12 22))´.
4. If session keys SK1 and SK2 are negotiated with SSC, SK2 must be used for command and response MAC security in *secure messaging* for the following commands, whereby the incremented SSC must be used as ICV.
   SK2 is therefore assigned the same CCT as SK1 under 3.

Tag ´B8´, CRT for data confidentiality (CT):
This data object may only occur if the algorithm ID defines that the key is a single-level key management key.

If the algorithm ID indicates a negotiation key, a definition for the negotiated session key SK1 must state whether or how it may be used for encryption. Here a CT is used in the following way:

1. If an SK1 was negotiated and this SK1 is not to be used for encryption, the KMT does not contain a CT.

2. If an SK1 was negotiated without SSC and this SK1 is to be used for encryption, the KMT contains a CT, which defines how the key may be used for encryption of the command or response message. In this case only DES3 CBC with ICV = 0 may be used as algorithm ID.
3. If an SK1 with SSC was negotiated and this SK1 is to be used for encryption, the KMT contains a CT, which defines how the key may be used for encryption of the command or response message and whether the SSC increment or the value 0 is to be used as ICV.

Tag ´BA´, key management template (KMT):
If the algorithm ID in the value field of the KMT defines that the key stored as a non volatile entity in the chip card is a dual-level key management key, the definitions and specifications defined in this chapter so far apply to a KMT it contains in restricted form, as these refer to a derived key.

The value field of a KMT within a KMT contains data objects from the following list in the sequence specified, whereby optional data objects (o) may be omitted.

Tag	Length	Value	
´89´	´XX...YY´	Algorithm ID	
´B8´	´XX...YY´	CRT for data confidentiality (CT), default value = ´B8 00´	o

Within a KMT which is contained in a KMT a key can only be assigned an algorithm ID for key negotiation. The algorithm ID must be consistent with the type and structure of the key previously derived. It must be a 16-byte key. If the type and structure of the key and the algorithm ID are not consistent, the key cannot be used for key negotiation.

The key for key negotiation must have a definition of how the negotiated session key or the negotiated session keys may be used. This takes place, as already described above, largely implicitly.

A CT within a KMT contained in a KMT may be omitted if the corresponding algorithm ID defines that only one session key is negotiated. Otherwise, the CT defines how encryption is to take place with session key SK1. Here, a CT is used as already described above.

4.9 EF_TIN

The EF_TIN (technical information) is a transparent data field with information about the customer and customer data.

In the file management data (FMD) of the file header, an ID embedding the customer and the customer specific part of the image is stored.

The contents of the record themselves can be freely selected by the customer.

FID: ´2F E4´	**Data field type: transparent**	
Security conditions	UPDATE	: Personalisation office
	READ / SEARCH	: ALW
	DELETE	: NEV
	APPEND	: – / –
	ACTIVATE	: NEV
	DEACTIVATE	: NEV

Data content: record - <n> bytes in length	
Byte	**Designation**
1...<n>	Assigned by customer

4.10 EF_ATR

The EF_ATR is stored in a specific EEPROM area and contains the cold and the warm ATR.

i The following table contains the ATR values which are entered when the minimal file system is created by the initialisation routine.

FID: ´2F 01´	Data field type: transparent	
Security conditions	UPDATE	: ALW
	READ / SEARCH	: ALW
	DELETE	: NEV
	APPEND	: – / –
	ACTIVATE	: ALW
	DEACTIVATE	: ALW

The following ATR specific abbreviations are used:

TS	Initial character
T0	Format character
TA1, TB1, TC1...	Interface character
HB01, HB02...	Historical character
TCK	Check character

Data content: record - <n> bytes in length

Byte	Value	Character	Designation	No. of bytes
1	‘1B’		Length cold ATR	1
2	‘3B’	TS	Initial character, specifies direct convention	1
3	´FF´	T0	Interface characters TA1, TB1, TC1 and TD1 follow, 15 histori-cal byte	1
4	´94´	TA1	Frequency conversion factor FI = 1001 corresponds to Fi = 512; Baud rate adaptation factor DI = 0100 corresponds to Di = 8 at maximally 5 MHz	1

5	’00’	TB1	No external Vpp	1
6	’FF’	TC1	Extra guardtime from 12 etu for T=0 and 11 etu for T=1	1
7	´80´	TD1	TD2 follows, first protocol offered is T=0	1
8	´B1´	TD2	TA3, TB3, TD3 follow with specific parameters for T=1	1
9	´FE´	TA3	Maximum length of the information field (information field size card) IFSC = 254	1
10	´45´	TB3	Character waiting time (CWT) = 5 Block waiting time (BWT) = 4 (default)	1
11	’1F’	TD3	TA4 follows with global interface bytes	1
12	’03’	TA4	No clock stop, Vcc = 3 V or Vcc = 5 V	1
13	’00’	HB01	ISO ATR	1
14	’68’	HB02	Preissuing data object of length 8	1
15...19	’D2 76 00 00 28’	HB03...HB07	International RID (Registered Identifier) of ORGA	5
20	´FF´	HB08	Mask ID = SY-SCA	1
21	´05´	HB09	Chip manufacturer = Infineon	1
22	´1E´	HB10	Chip number, 0x1E = 30 decimal	1
23...24	’31 80’	HB11...HB12	Card profile data object	2
25	’00’	HB13	Card life status	1
26...27	´90 00´	HB14...HB15	Status bytes	2

28	´23´	TCK	Checksum according to the formula: $T0 \oplus TA1 \oplus \ldots \oplus TCK \equiv$ ´00´	1
29...34	’00..00’		Padding	6
35	’13’		Length of the warm ATR	1
36	’3B’	TS	Initial character, specifies direct convention	1
37	´6F´	T0	Interface characters TB1 and TC1 follow, 15 historical bytes	1
38	’00’	TB1	No external Vpp	1
39	´FF´	TC1	Extra guardtime from 12 etu for T=0	1
40...54		HB01... HB15	see above	15
55...68	’00..00’		Padding	14
69...71	’00 00 00’		RFU	3

The contents of the data field can be changed with the command **UPDATE BINARY.** Changes should be an exception, but it could be necessary to reduce the length of the ATR.

If the ATR is changed, the following rules have to be considered:

– the historical bytes could be changed, but as they contain descriptional information they should never be changed
– the cold ATR can be reduced, but as a minimum it has to contain the entries from the warm ATR (except the historical bytes)
– the first protocol offered has to be the protocol T=0
– the fast I/O (set with the frequency conversion factor) is not mandatory
– the maximum length of the information field can be reduced

5 Security Features

5.1 Cryptographic Algorithms

The basic algorithm that is integrated into MICARDO is DES (data encryption standard), which is used for standard security features such as *secure messaging* and the encryption of passwords.
The cryptographic algorithm specifies not only the procedures for encryption and decryption, but also the length and structure of random numbers and cryptographic checksums.

MICARDO also provides asymmetrical cryptographic algorithms (RSA) for PSO commands and authentication based on keys.

5.1.1 Basics

5.1.1.1 Padding

MICARDO supports two types of padding:

– ISO padding	(Data block || ´80 00 … 00´)
– Null padding	(Data block || ´00 00 … 00´)

The following rules apply:

– Data is always padded to make the length of the padded data block a multiple of 8 bytes.

– Encryption and MAC computation under *secure messaging* always uses ISO padding, and padding is always applied even if the length of the data block is already a multiple of 8 bytes.
– ISO padding is always used for the PSO command **DECIPHER**.

5.1.1.2 Initial Chaining Value

Block-oriented cryptographic algorithms operating in CBC mode (cipher block chaining) require a starting value, the ICV (initial chaining value), in order to process the first block.

The ICV is coded in a simple data object with tag ´87´. This data object is known as an ICV data object and its length may be one of the following:

Length	Significance
´00´	Not an ICV
´01´	Specifies which procedure the cryptographic algorithm should use for ICV handling.
´08´	Contains an ICV.

Here is a description of how and where the individual commands which need an ICV can find it.

PSO Commands

When needed, all PSO commands use the volatile IcvPso stored in the security environment. This is passed to MICARDO using the command **MANAGE SECURITY ENVIRONMENT**.

Secure Messaging

Generally, the party receiving the message determines the ICV.

The following five tables show where the ICV is located for *secure messaging* purposes.

The significance of the abbreviations is as follows:

CCT	CCT for ICV - CRT for data authentication in the command message or response message

CT	CT for ICV - CRT for data confidentiality of the command message or response message
Algo ID	The algorithm ID
SSC	Send sequence counter
GC	The command **GET CHALLENGE**
Ind	The ICV indicator (in SE data object for keys)

Missing combinations in the tables are not of importance. An ´x´ in a table cell indicates that the content of that cell is not relevant .

You should note that with both chaining mode and MAC command security you cannot use the command variant with the command **GET CHALLENGE** because this interrupts the chain.

ICV for MAC command security:

CCT	Algo ID	SSC	GC	Operation
no	ICV = 0	no	x	MAC is computed with ICV = 0
	ICV ≠ 0	no	yes	ICV is the random number from the previous **GET CHALLENGE** command
		yes	x	SSC := SSC + 1; ICV = SSC

ICV for MAC response security:

CCT	Algo ID	SSC	GC	Operation
no	ICV = 0	no	x	MAC is computed with ICV = 0
	ICV ≠ 0	no	yes	ICV is IcvSecureMessaging from the volatile security environment
		yes	x	SSC := SSC + 1; ICV = SSC
yes	ICV ≠ 0	no	x	Uses ICV from the data object of type response descriptor

ICV for command encryption:

CCT	Algo ID	Ind	GC	Operation
no	ICV = 0	x	x	Data is decrypted with ICV = 0
	ICV ≠ 0	´02´	yes	ICV encryption is ICV MAC security
yes	ICV ≠ 0	´01´	x	Uses ICV from command message

ICV for response encryption:

CCT	Algo ID	Ind	GC	Operation
no	ICV = 0	x	x	Data is decrypted with ICV = 0
	ICV ≠ 0	´01´	x	ICV is chosen arbitrarily by MICARDO and included in the response message
		´02´	yes	ICV encryption is ICV MAC security

The following applies for secure messaging with negotiated session keys:

ICV for command or response encryption:

CT	Algo ID	SSC	Operation
no	ICV = 0	no	Data is decrypted with ICV = 0
	ICV ≠ 0	yes	Use SSC as ICV and decrypt

5.1.2 DES

5.1.2.1 DES Variants

MICARDO knows two different ways of performing the DES calculation, standard DES and triple DES. The same basic encryption and decryption routines are used in both forms, but differences arise in the way input data is treated. Which variant is used depends on the key length of the DES keys in the data field EF_Key and is indicated by a different algorithm ID. MICARDO supports 8-byte keys for standard DES and 16-byte keys for triple DES. A 16-byte key consists of two 8-byte keys.

Standard DES always uses 64-bit keys (56 bits + 8-bit parity). MICARDO always tests the keys' parity bits before executing a DES encryption or decryption; the DES keys in MICARDO must always have odd parity. Incorrect parity provokes an error message and the associated command is then not executed.

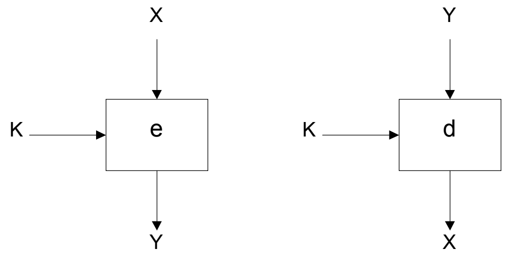
5.1.2.2 DES

DES is the fundamental algorithm used in MICARDO. Both encrypt mode and decrypt mode are supported.

The DES encryption of an 8-byte block of plaintext X to an 8-byte ciphertext Y, and the DES decryption of an 8-byte ciphertext Y to an 8-byte block of plaintext using an 8-byte key K (56-bit entropy) are illustrated as follows:

Encryption: Y = eK(X)

Decryption: X = dK(Y)



MICARDO uses DES for simple encryption and triple encryption (triple DES).

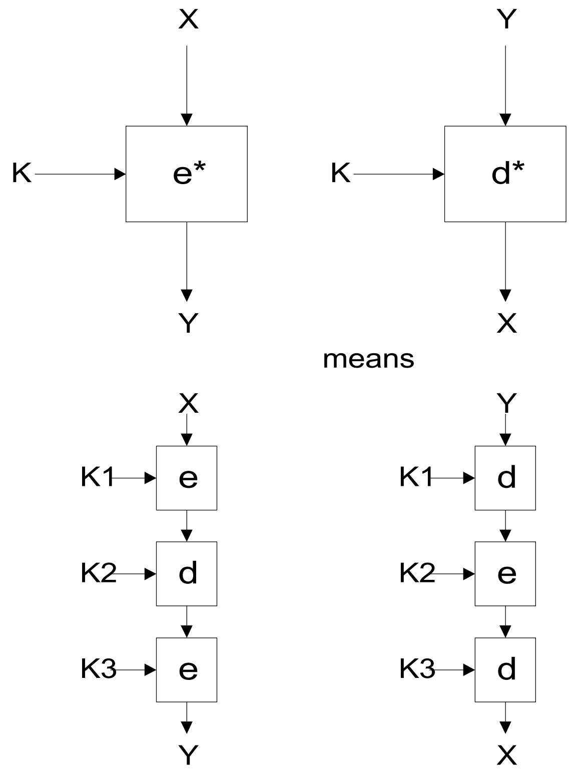
5.1.2.3 Triple DES

Triple DES is a triple application of DES in the manner described below.

The triple DES encryption of an 8-byte block of plaintext X to an 8-byte ciphertext Y and the triple DES decryption of an 8-byte ciphertext Y to an 8-byte block of plaintext using a 24-byte key K=K1 || K2 || K3, where K1, K2 and K3 are three 8-byte DES keys, are illustrated as follows:

Encryption: $Y = e^{*}K(X) := eK1(dK2(eK3(X)))$

Decryption: $X = d^{*}K(Y) := dK1(eK2(dK3(X)))$



Remarks:

– If K1 = K2 = K3, the result of a triple encryption is the same as that of a simple encryption.

– If K1 = K3 we refer to a 2-key triple DES. Triple DES is generally operated in this mode.
MICARDO supports only this variant of triple DES. This document will also refer to this 2-key triple DES as DES3.

5.1.2.4 Encryption and Decryption

DES CBC Mode

For simple encryption, DES is used in CBC mode both for encryption and decryption and for calculating cryptographic checksums. CBC mode requires an 8-byte ICV to be defined.

The following notation is used to depict the CBC mode encryption of the data field X whose length is a multiple of 8 bytes (if necessary the input string must be brought up to such a length by appending a padding string), using the 8-byte key K and an ICV to produce the output string Y:

Y = eK(ICV, X)

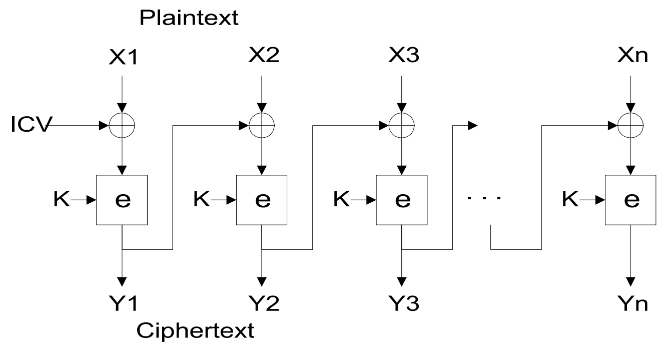
where Y has the same length as X (a multiple of 8 bytes)

The CBC mode decryption of the data field Y, whose length is a multiple of 8 bytes, using the 8-byte key K and the ICV is then depicted as follows:

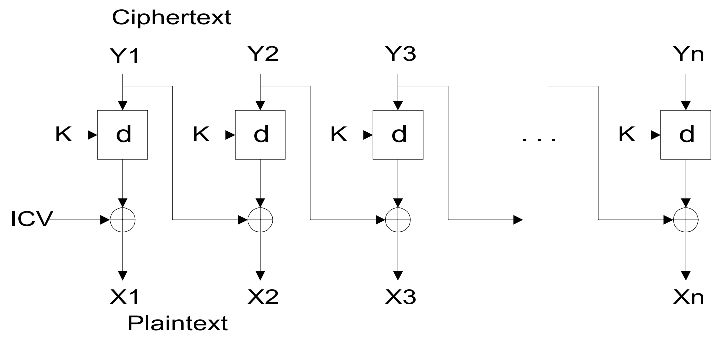
X = dK(ICV, Y)

The following diagrams illustrate encryption (first diagram) and decryption (second diagram) in CBC mode.
X represents the concatenation of n 8-byte blocks X1,..., Xn, and similarly Y represents the concatenation of n 8-byte blocks Y1,..., Yn; K represents the DES key.
´XOR´ indicates a bit-wise exclusive OR operation, ´e´ is the DES encryption and ´d´ the decryption.

Encryption



Decryption



Triple DES CBC Mode

MICARDO also uses triple DES in CBC mode. The triple DES encryption in CBC mode of a data field X whose length is a multiple of 8 bytes (if necessary the input string must be brought up to such a length by appending a padding string), using the 16-byte key K = K1 || K2 (DES3 key) and an 8-byte ICV is then depicted as follows:

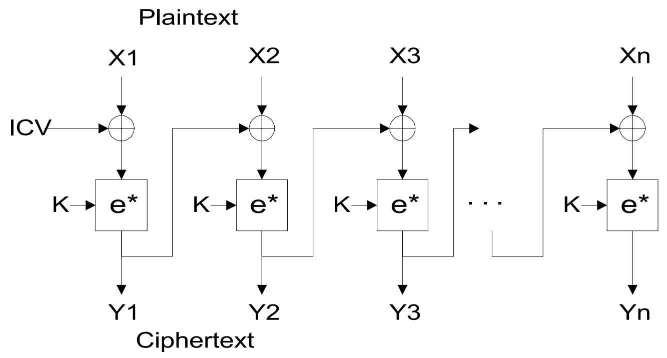
Y = e*K(ICV,X)

and the corresponding decryption as:

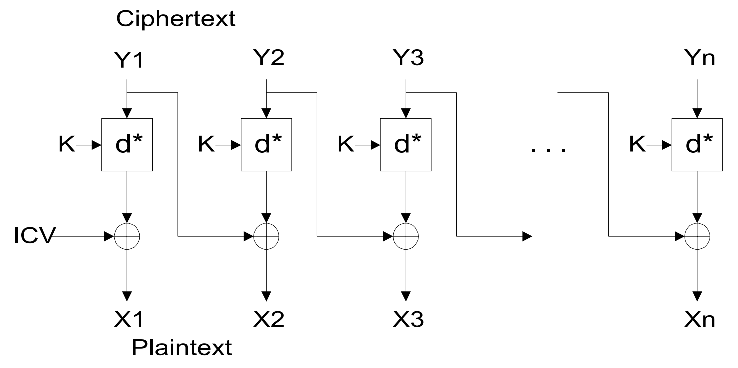
X = d*K(ICV,Y)

The next two diagrams illustrate encryption (first diagram) and decryption (second diagram) in DES3, where X represents the concatenation of n 8-byte blocks X1,..., Xn, and similarly Y represents the concatenation of n 8-byte blocks Y1,..., Yn; K = K1 || K2 represents the concatenation of two DES keys to form a DES3 key, e* signifies DES3 encryption and d* DES3 decryption

Encryption



Decryption



5.1.2.5 Cryptographic Checksums

DES is used in CBC or CFB mode for the compilation of cryptographic check-sums.

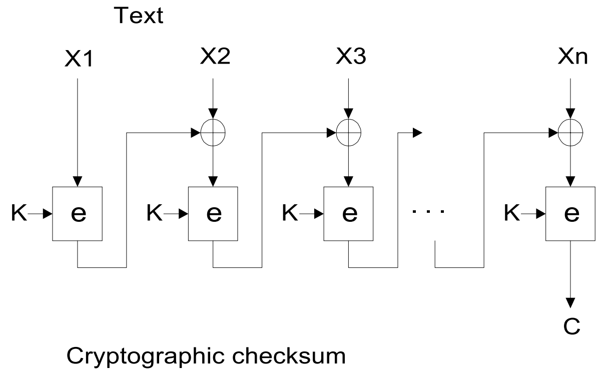
A cryptographic checksum is a MAC (message authentication code). A MAC is the final block of a DES encryption in CBC mode or CFB mode. This means that only the last 8-byte output block represents the result of the function, so the output is no longer reversible (one-way function).

DES CBC MAC

You can compute a MAC using DES in CBC mode on a data field X whose length is a multiple of 8 bytes. If necessary the input string must be brought up to such a length by appending a padding string. The CBC MAC uses an 8-byte key K and an ICV of zero, and computes an output C with a length of 8 bytes. The CBC MAC for such a case is depicted as follows:

C = mK(X)

The following diagram illustrates the use of DES in CBC mode, where X is the concatenation of n 8-byte blocks X1,...,Xn, K is a DES key and ´e´ signifies DES encryption.

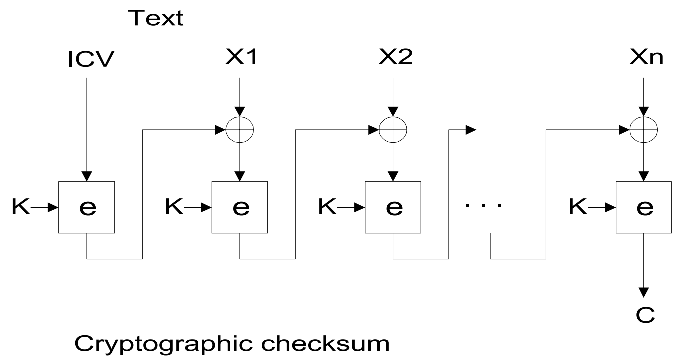


DES CFB MAC

You can compute a MAC by using DES in CFB mode on a data field X whose length is a multiple of 8 bytes. If necessary the input string must be brought up to such a length by appending a padding string. The CFB MAC uses an 8-byte key K and an ICV and computes an output C with a length of 8 bytes. The CFB MAC for such a case is depicted as follows:

C = mK(ICV, X)

The following diagram illustrates the use of DES in CFB mode, where X is the concatenation of the n 8-byte blocks X1,...,Xn, K is a DES key and ´e´ signifies DES encryption.



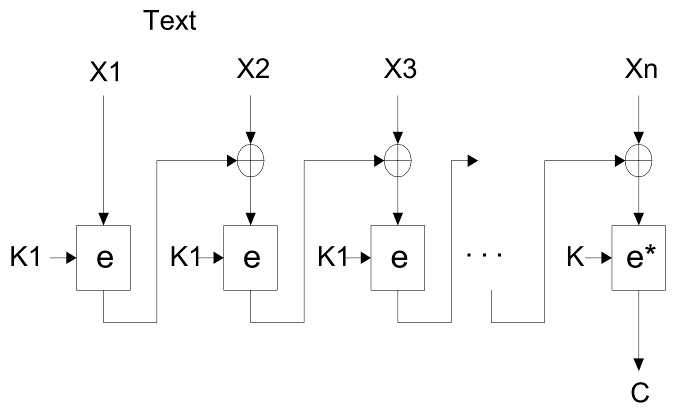
Retail CBC MAC

To compute a cryptographic checksum using triple DES in CBC mode you apply DES in a similar way as for the simple application. The word "triple" here refers only to the final block. This gives you the retail CBC MAC.

The computation of a cryptographic checksum using DES3 in CBC mode is depicted as follows, where K = K1 || K2:

C = m*K(X),

The following diagram illustrates the use of DES3 in CBC mode, where X is the concatenation of the n 8-byte blocks X1,...,Xn, K is a DES3 key and ´e*´ signifies DES3 encryption.



Cryptographic checksum

Retail CFB MAC

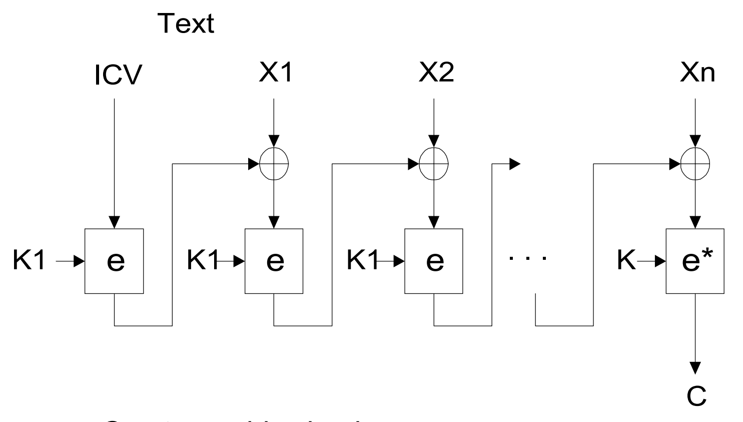
To compute a cryptographic checksum using triple DES in CFB mode you apply DES in a similar way as for the simple application. The word "triple" here refers only to the final block. This gives you the retail CFB MAC.

The computation of a cryptographic checksum using DES3 in CFB mode is depicted as follows:

C = m*K(ICV,X),

where K = K1 || K2.

The following diagram illustrates the use of DES3 in CFB mode, where X is the concatenation of the n 8-byte blocks X1,...,Xn, K is a DES3 key and ´e*´ signifies DES3 encryption:



Cryptographic checksum

5.1.3 RSA

Since MICARDO Public possesses a cryptographic coprocessor for RSA calculations, its cryptographic algorithm is used for a variety of commands (e.g. PSO commands).

5.1.3.1 Parameters for RSA

MICARDO uses the RSA algorithm for decryption as well as signatures. The maximum length of the modulus is 1024 bits.

The RSA algorithm uses keys that are computed as follows:

1. Take two different prime numbers p and q, the product of which is the modulus n, i.e.:
   $n = p \bullet q$
2. Then choose an arbitrary number e and determine the number d which is the inverse of e in the remainder class ring Z(j(n)). The function j is Euler's

phi function. We have:
$d \bullet e \equiv 1 \bmod (p-1)(q-1)$

3. The number e together with the modulus constitutes the public key, while d together with n is the private (secret) key. The two together constitute what is called a key pair.

5.1.3.2 Parameters for RSA - Chinese Remainder Theorem

You can optimise the elaborate process of modulo exponentiation in the RSA algorithm by applying the Chinese remainder theorem. An optimized variant of this theorem for calculation is a follows:

$y = (((E_p - E_q) \bullet c) \bmod p) \bullet q + E_q$

where

$E_p = x^{d \bmod p-1} \bmod p,$
$E_q = x^{d \bmod q-1} \bmod q$

and

$c = q^{-1} \bmod p$ (modulo inverse of q)

In this case the private key is made up of the following values:

p (prime)
q (prime)
d mod p-1 (reduced secret exponent modulo p-1)
d mod q-1 (reduced secret exponent modulo q-1)
c (constant)

5.1.3.3 RSA Encryption in General

The encryption of a message m is effected by exponentiating it with a public key e.

The ciphertext c is calculated from the plaintext that is to be encrypted, the message m, using the following formula:

$c = m^e \bmod n$

The hardware provides efficient cryptographic algorithms for calculating the modulo exponentiation.

The message m itself may previously have been formatted in any way, because the message that is to be encrypted is supplied by the user. To ensure that the decrypted message is unique, the modulus needs to be larger than the message, where "larger" is used here in the sense of a comparison between two integer values.

5.1.3.4 RSA Decryption in General

The decryption of a message m is effected by exponentiating it with a private key d. This key is one of a key pair of which the private key is always stored on the chip card itself.

The plaintext contained in the message m is calculated from a ciphertext that is to be decrypted using the following formula:

$m = c^d \bmod n$

The complete message m is output to the user. The decryption can also be carried out using the Chinese remainder theorem.

5.1.3.5 RSA Decryption According to PKCS#1

MICARDO expects an encrpyted text. The plaintext included in the encrypted text must have the following format (block type 02):

Header	Block type	Padding string	Separator	Plaintext
´00´	´02´	´XX´ || ´XX´ || ...	´00´	m

The length of the padding string is lg(n) - lg(m) - 24 bits. The padding string is made up of random non-zero bytes, so that the padding can be distinguished from the separator.

The length of the plaintext m may be at most the modulus length minus 11 bytes. The formatted message itself is the same length as the modulus.

5.1.3.6 RSA Signatures in General

The signature of a message m is created by exponentiation with a secret key d. This key is one of a key pair of which the private key is always stored on the chip card itself.

The signature of a message m is created in three steps, which are carried out as follows:

Step 1: Calculate the hash value
The message m passes through a hash algorithm which produces a 160-bit hash value h = f(m). The calculation of the hash value is performed outside the chip card, or can optionally be carried out in the chip card using an independent command (the PSO command **HASH**).

Step 2: Format the DSI
The hash value h is extended as directed by a prescribed formatting and brought up to the length of the modulus that is to be used, n. The result of this is the DSI (digital signature input).

Step 3: Calculate the signature
The signature is calculated according to the following formula:

$s = DSI^d \bmod n$

The chip card returns the computed signature s as its result.

5.1.3.7 RSA Signature Using PKCS#1

MICARDO enables you to use PKCS#1 format for creating signatures. In this case a message m hashed in the chip card using SHA-1, is formatted as follows before being signed (block type 01):

Header	Block type	Padding string	Sepa-rator	Object ID	Hash value
´00´	´01´	´FF´||´FF´|| ...	´00´	3021 3009 06052B0E03021A 0500 0414	h

The length of the padding string is lg(n)-24-280 bits. The formatted message is thus of the same length as the modulus. The object ID contains an ASN.1 code sequence to flag the use of SHA-1.

You can alternatively send the entire message digest md (ASN.1 coding plus hash value) to the chip card, so that the DSI is formatted as follows:

Header	Block type	Padding string	Sepa-rator	Message digest
´00´	´01´	´FF´ || ´FF´ || ...	´00´	md

The length of the padding string is lg(n)-24-lg(md). The maximum length of the message digest is 384 bits.

5.1.4 Hash Algorithms

The SHA-1 hash algorithm has been implemented for calculating hash values within the chip card.
From a message m of any desired length the algorithm calculates a specific 160-bit value h. Such a value is called a hash value.

5.2 Authentication Procedures

5.2.1 Symmetrical Procedures

5.2.1.1 Internal Authentication

An internal authentication establishes, for the external world, that the chip card or a particular application is genuine.

The chip card achieves this by encrypting a random number that it is given, RND1 (8 bytes), using a specified key, K.ICC, and passing the result RSP1 to the external world for verification.

No padding is inserted and the value ´00 00 00 00 00 00 00 00´ is always used as ICV for the DES CBC mode procedure.

External World		MICARDO
Random number RND1	**INTERNAL AUTHENTI- CATE** --------------------------> RND1	Encrypt RND1 with K.ICC
		| V
Verify RSP1 with K.IFD and RND1	<-------------------------- RSP1	Cryptogram RSP1

The authenticity is considered to have been confirmed if the cryptogram from the chip card was generated with the same key as that possessed by the external world (K.IFD = K.ICC).

5.2.1.2 External Authentication

An external authentication is performed when the external world has to prove its authenticity to the chip card or a particular application.
For this purpose the chip card outputs an 8-byte random number RND0, which is fetched using the command **GET CHALLENGE**. The external world encrypts this random number using a key K which is known both to itself and to the chip card. No padding is inserted.
The value ´00 00 00 00 00 00 00 00´ is always used as ICV for the DES CBC mode procedure. The cryptogram RSP0 is passed to the chip card for verification.

External World		MICARDO
Encrypt RND0 with K.IFD	**GET CHALLENGE** <-------------------------- RND0	Random number RND0

```
External World                                              MICARDO

      |
      V                EXTERNAL AUTHENTICATE
Cryptogram RSP0       -------------------------->         Verify RSP0
                               RSP0                   with K.ICC and RND0
```

The authenticity is considered to have been confirmed if the cryptogram RSP0 from the external world was generated with the same key as that possessed by the chip card (K.ICC = K.IFD).

5.2.1.3 Mutual Authentication

In the case of a mutual authentication, performed using the command **MUTUAL AUTHENTICATE**, the external world first has to prove its authenticity to the card or application. When it has done so the chip card then proves its authenticity to the external world.

For this purpose the chip card outputs an 8-byte random number RND0, which is fetched using the command **GET CHALLENGE**. The external world encrypts this random number using its key K.IFD. In this case the value ´00 00 00 00 00 00 00 00´ is always used as ICV for the DES CBC mode procedure. The cryptogram RSP0 is passed to the chip card together with another random number RND1 generated by the external world.

MICARDO first checks the authenticity of the external world. The authenticity is considered to have been confirmed if the cryptogram RSP0 from the external world was generated with the same key as that possessed by the chip card (K.ICC = K.IFD).

MICARDO then encrypts the random number RND1 using its key K.ICC. In this case the value ´00 00 00 00 00 00 00 00´ is always used as ICV for the DES CBC mode procedure. The cryptogram RSP1 is then passed to the external world for verification.

No padding is inserted in either case.

```
   External World                                        MICARDO

                           GET CHALLENGE              Random number
      Encrypt         <--------------------------         RND0
  RND0 with K.IFD                RND0
         |
         V
  Cryptogram RSP0      MUTUAL AUTHENTICATE
 and random number    -------------------------->      Verify RSP0
       RND1                   RSP0, RND1          with K.ICC and RND0
                                                           |
                                                           V
                                                        Encrypt
                                                    RND1 with K.ICC
                                                           |
                                                           V
    Verify RSP1       <--------------------------    Cryptogram RSP1
with K.IFD and RND1              RSP1
```

5.2.2 Asymmetrical Procedure

5.2.2.1 Internal Authentication

An internal authentication using public key procedures establishes, for the external world, that the chip card or a particular application is genuine.

For this purpose, the chip card receives an n-byte random number, RND1, signs it with a private key and returns the result, RSP1, to the external world for verification.

External World		**MICARDO**
Random number RND1	**INTERNAL AUTHENTI- CATE** --------------------------> RND1	Format RND1
		| V
Verify SIGN1 with PK.ICC and RND1	<-------------------------- SIGN1	DSI and SK.ICC SIGN1

It is also possible to work with a hash value instead of RND1 (using the PSO command **HASH**). In this case the command **INTERNAL AUTHENTICATE** is called with an empty command data field.

The signature is computed by using one of the signature procedures associated with the private key SK.ICC.

External World		**MICARDO**
Random number RND1	**HASH** --------------------------> RND1	Hash RND1 HASH1
	INTERNAL AUTHENTI- CATE -------------------------->	HASH1 and SK.ICC SIGN1

External World		MICARDO
		|
		V
Verify SIGN1	<--------------------------	SIGN1
with PK.ICC and RND1	SIGN1	

5.3 Digital Signatures

MICARDO supports the computation of digital signatures using a variety of different methods. The following sections describe the command procedures for these variants.

To achieve sufficient security of digital signatures, cryptographically strong keys have to be used.
These standards are laid down in the German digital signature act, at the moment a minimum key length of 1024 bits is demanded.

5.3.1 Digital Signatures with Hashing in the Chip Card

When large volumes of data have to be hashed and signed in the chip card the system first uses the PSO command **HASH** in chaining mode to hash the message that is to be signed, TOKEN1 (or part of it, such as the final block). This hash value, HASH0, is then temporarily stored.

If the PSO command is immediately followed by the command **COMPUTE DIGITAL SIGNATURE** with an empty data field, then this **COMPUTE DIGITAL SIGNATURE** will encrypt the temporary hash value using the referenced private signature key SK.ICC.

The computed signature SIGN0 is returned in the response data.

External World		MICARDO
	HASH	
TOKEN1	-------------------------->	Hash TOKEN1
	TOKEN1	HASH0

External World		MICARDO
	COMPUTE DIGITAL SIGNATURE	Sign
	-------------------------->	HASH0 with SK.ICC
		|
		V
	<--------------------------	SIGN0
	SIGN0	

5.3.2 Digital Signatures without Hashing

If the external world itself hashes the message that is to be signed the chip card will interpret this hash value as the token.

TOKEN1 is then used to format the DSI, which is subsequently signed using the private signature key SK.ICC.

External World		MICARDO
	COMPUTE DIGITAL SIGNATURE	Sign
TOKEN1	-------------------------->	TOKEN1 with SK.ICC
	TOKEN1	
		|
		V
	<--------------------------	SIGN0
	SIGN0	

5.4 RSA Key Generation

MICARDO generates asymmetric cryptographic keys (512 - 1024 bit RSA keys). It enforces that the key material meets the following requirements:

- Random numbers used in the key generation have a high quality to ensure that the key is unique with a high probability.
- Prime numbers used in the key generation are unique with a high probability.

- Any private key cannot be derived from the corresponding public key.

MICARDO uses the true random number generator of the semiconductor to get real random numbers. These true random numbers are then used to generate the prime numbers for the keys.

If the keys are generated for computation of digital signatures according to the German digital signature act, a key length of 1024 bit has to be chosen.

The appendix contains a detailed description of an RSA key generation. The section "Example of Generation of a Key Pair" on page 245 lists all necessary steps and commands.

5.5 Key Management

MICARDO supports two types of internal key management:

- The derivation of a card-specific key from a master key.
- The negotiation of session keys following mutual authentication.

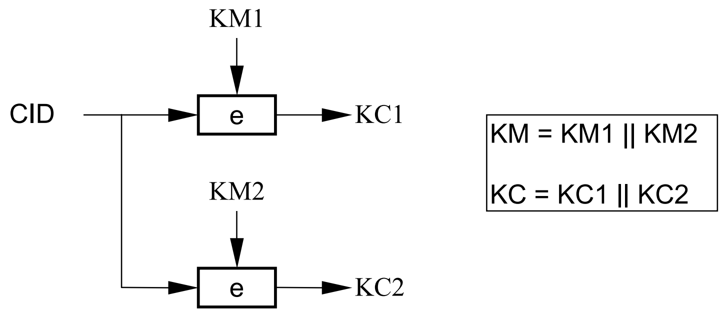
The procedure for determining common ciphers K0 and K1 and the derivation of session keys SK0 and SK1 is described in the following sections.

MICARDO does not check the cryptographic strength of derived or negotiated keys; such a check is to be performed by the external world.

5.5.1 Master Key Method for the Health Professional Card

The master key method for health professional cards as laid down by HPC makes it possible to implement authentication protocols based on symmetrical cryptographic algorithms. In this case the chip card, which is able to perform the master key procedure, functions as master, able to verify the authenticity of a number of chip cards (clients). The <n> chip cards differ in their serial number ICCSN (the 8 lower bytes of this number). The serial number is passed to MICARDO as the CID using the command **MANAGE SECURITY ENVIRONMENT**.

The authentication of the chip cards to the master references a master key in the key management template (KMT) within the authentication command. The chip card then derives the card-specific key KC (DES3) as shown in the following diagram.



The master key procedure according to HPC applies DES to the CID twice, each time using one half of the master keys, KM= KM1 || KM2.

The derived key KC is given odd parity and it can then be used for the subsequent authentication.

5.5.2 Symmetrical Negotiation of Ciphers

The procedure for negotiating common ciphers K by using symmetrical procedures uses the **MUTUAL AUTHENTICATE** variant of the command **EXTERNAL AUTHENTICATE**.
The negotiation of session keys involves mutual authentication of the chip card and the external world. At the same time you can optionally initialise an 8-byte SSC (send sequence counter).

RND.ICC and RND.IFD are each 8 bytes long. K.ICC and K.IFD can each be 16 or 32 bytes long.

When session keys are negotiated, the command **GET CHALLENGE** has to be used to obtain a random number RND.ICC from the chip card, and this must be done immediately prior to calling the command **MUTUAL AUTHENTICATE**.

When the chip card receives the command **Mutual Authenticate** it must check that the random number RND.ICC it previously generated and supplied is correctly present in bytes 9 to 16 of the decrypted command data.

No padding is used for this encryption. The encryption and decryption is performed using DES CBC mode and ICV=´00 00 00 00 00 00 00 00´.

External World		**MICARDO**
	Get Challenge	Random number
Encrypt	<--------------------------	RND.ICC
RSP0	RND.ICC	
|		
V	**Mutual Authenticate**	
RSP0	-------------------------->	Verify RSP0 with
	RSP0	RND.ICC and K.ICC
		|
		V
Verify RSP1 with	<--------------------------	Encrypt
RND.IFD and K.IFD	RSP1	RSP1
|		|
V		V
Session key from		Session key from
K.ICC and K.IFD		K.ICC and K.IFD

When each of the parties has authenticated its communications partner, session keys are then derived from the two keys K.ICC and K.IFD as described in the chapter chapter "Negotiating the Session Keys" on page 111.

5.5.3 Negotiating the Session Keys

K.IFD and K.ICC must have the same length. The length of K.IFD ⊕ K.ICC is then 16 or 32 bytes. If K.IFD ⊕ K.ICC is of length

– 16 bytes, we will get one session key of length 16 bytes:
  SK1 = K.IFD ⊕ K.ICC.

– 32 bytes, we will get two session keys, each of length 16 bytes:
SK1 || SK2 = K.IFD ⊕ K.ICC.

The least significant bit of each byte in the session keys is changed, if necessary, to give the byte odd parity.

If you are simultaneously initialising an SSC (send sequence counter), then the 4 least significant bytes of each of RND.ICC and RND.IFD are concatenated:

SSC = 4 least significant bytes of RND.IFD || 4 least significant bytes of RND.ICC.

The negotiated session keys are used in the following way:

One session key SK1 and no SSC:
For *secure messaging* the command message has to have been given a correct SK1 retail CBC MAC. An SK1 retail CBC MAC is also calculated for the response message. The command and/or response data can also be encrypted using the SK1 from triple DES CBC mode, using an ICV of zero.

One session key SK1 and SSC:
For *secure messaging* the command message has to have been given a correct SK1 retail CFB MAC. An SK1 retail CFB MAC is also calculated for the response message. Before forming each MAC you must increase the SSC by 1 and then use this value as the ICV. Thus the ICV for calculating the MAC of the first command message is SSC+1 and the ICV for calculating the MAC of the first response message is SSC+2. The command and/or response data can also be encrypted using the SK1 from triple DES CBC mode, either with ICV=0 or with the same value as was used for calculating the MAC.

Two session keys SK1 and SK2 and no SSC:
For *secure messaging* the command message has to have been given a correct SK2 retail CBC MAC. An SK2 retail CBC MAC is also calculated for the response message. The command and/or response data can also be encrypted using the SK1 from triple DES CBC mode, using an ICV of zero.

Two session keys SK1 and SK2 and SSC:
For *secure messaging* the command message has to have been given a correct SK2 retail CFB MAC. An SK2 retail CFB MAC is also calculated for the response message. Before forming each MAC you must increase the SSC by 1 and then use this value as the ICV. Thus the ICV for calculating the MAC of

the first command message is SSC+1 and the ICV for calculating the MAC of the first response message is SSC+2. The command and/or response data can also be encrypted using the SK1 from triple DES CBC mode, either with ICV=0 or with the same value as was used for calculating the MAC.

MICARDO does not check the cryptographic strength of the negotiated key; such a check is to be performed by the external world.

5.6 Secure Messaging

Secure messaging is used to secure data transmissions. MICARDO includes a component which carries this name and deals with this task. It employs MAC security for the purposes of data authentication and encryption to maintain the security of confidential data.

When confidential data are exchanged between external world and MICARDO, secure messaging should be performed with coding and MAC security of command and response.
To protect the data transmission best, session keys und send sequence counters (SSC) should be used additionally (see section "Symmetrical Negotiation of Ciphers" on page 110 and section "Negotiating the Session Keys" on page 111).

The use of *secure messaging* implies that the command and/or the response are carried out in *secure messaging* format. This formatting effects a secured data transfer. The transferred messages are always TLV-encoded and are made up of SM data objects.

The following are used:

– Data objects of type data field
– Data objects of type Le
– Data objects of type status information
– Data objects of type MAC
– Data objects of type response descriptor
– Data objects of type CRT

You should be aware that secure messaging as defined by ISO does specify the TLV-coding of the messages but not the values of actual data objects.

The class byte indicates that the command is being executed under *secure messaging*. This applies even when only the command or only the response is MAC-secured or encrypted under *secure messaging*.

A command may be executed without the TLV code for *secure messaging* only if neither a command message nor a response message is to be created under *secure messaging*. The class byte ´X0´ indicates that no *secure messaging* is being carried out.

With an unsecured command, user data of a length up to 255 bytes can be sent to and user data of a length up to 256 bytes can be received from the card.

When using *secure messaging*, the length of the secured message must not exceed these limits. Thus the length of the user data has to be reduced by the space needed for tag and length fields and optional secure messaging information.

5.6.1 Structure of Messages under Secure Messaging

A command without *secure messaging* is described as **unsecured**, and a command with *secure messaging* is described as **secured**.

The layout of unsecured command APDUs and response APDUs is explained in detail in the section chapter “Structure of Commands/Responses” on page 30.

Here we include only the schematic representation for comparison purposes, with no further explanation.

Command APDU of an unsecured command

CLA = 'X0'	INS	P1	P2	Lc	Data field	Le

Response APDU of an unsecured command

Data field	SW1	SW2

Command APDU of a secured command

CLA	INS	P1	P2	Lc´	Data field´	Le´ = ´00´

The class byte for a secured command has one of the values ´XC´ and ´X8´.
Lc´ and the entries Data field´ and/or Le´ = ´00´ may be omitted.
If the command message contains a Data field´ this must be TLV-encoded.

Response APDU of a secured command

Data field´	SW1	SW2

Here the entry Data field´ is always present and always TLV-encoded.

5.6.2 Class Bytes

For commands that are to be executed under *secure messaging* you must use a class byte with one of the following values:

CLA	Explanation
´XC´	The command header is included in the MAC security of the command message.
´X8´	The command header is not included in the MAC security of the command message.

For commands that are executed without *secure messaging* you must use a class byte with the value ´X0´.

CLA = ´X8´ must specifically be used if the command message is not MAC-secured although encryption is used for the command message and/or MAC security and/or encryption will be used for the response message.

5.6.2.1 Data Objects of Type Data Field

When a command is to be executed under *secure messaging* and the command or the response contains a data field there are two methods of converting it to a data object of type data field.

1. **As plaintext:**
   The value field is placed unchanged in a simple data object with tag ´80´ or ´81´.

   ´80´ || L || data field   or   ´81´ || L || data field

   Such data objects are referred to as plaintext data objects. They can be placed in the data field´ of a command and/or response message.
   – Data objects with tag ´80´ are used to hold the data field of an unsecured command when the command is to receive neither MAC security nor encryption, but one or other security measure is to be used for the response.
   – Data objects with tag ´80´ are used to hold the data field of an unsecured response when the response is receiving neither MAC security nor encryption, but one or other security measure was used for the command.
   – Data objects with tag ´81´ are used to hold the data field of a command or response message if that message has been or will be given MAC security.
   – When the plaintext data object is extracted from the data field of a command the length L of the plaintext data object has the value Lc.
   – When the plaintext data object is extracted from the data field of a response the length L of the plaintext data object has the value Lr.

2. **As a cryptogram:**
   The command or response is encrypted and the cryptogram, preceded by a byte of value ´01´, is inserted as a value field in a simple data object with tag ´86´ or ´87´.

   ´86´ || L || ´01´ || enc(data field || ´80´ [|| ´00´..´00´])
   or
   ´87´ || L || ´01´ || enc(data field || ´80´ [|| ´00´..´00´]).

   Such data objects are referred to as cryptogram data objects. They contain the encrypted data fields, padded as necessary, and are placed in the data field´ of a command and/or response message.

– The length L of a cryptogram data object is always one greater than a multiple of the block length of the encryption algorithm.
  When DES-based cryptographic algorithms are used for the encryption, L will therefore have one of the values 9, 17, 25, …
– Data objects with tag ´86´ are used to hold the encrypted data field of the command or response when security is provided by encryption.
– Data objects with tag ´87´ are used to hold the encrypted data field of the command or response when security is provided by both encryption and MAC security.

If the command or response contains no data field then the corresponding secured message contains no data object with tag ´80´, ´81´, ´86´ or ´87´.

The following table summarises which data object with which tag may be used to hold the (encrypted) data field of a command or response message. Other combinations of tags cannot occur.

Each entry in the table first gives you the tag for the command message and then the tag for the response message.

Data objects for data fields under secure messaging:

Command / Response	**No security**	**MAC security**	**Encryption**	**MAC security and encryption**
No security	- -	´81´ ´80´	´86´ ´80´	´87´ ´80´
MAC security	´80´ ´81´	´81´ ´81´	´86´ ´81´	´87´ ´81´
Encryption	´80´ ´86´	´81´ ´86´	´86´ ´86´	´87´ ´86´
MAC security and encryption	´80´ ´87´	´81´ ´87´	´86´ ´87´	´87´ ´87´

The value field of a data object with tag ´80´, ´81´, ´86´ or ´87´ is the complete (encrypted) data field of the command or response. This means that the command or response messages of secured commands will always contain at most one plaintext or cryptogram data object.

5.6.2.2 Data Objects of Type Le

If a command which contains an Le byte is to be executed under *secure messaging* the Le byte is inserted as a value field in a simple data object with tag ´96´ or tag ´97´ and length ´01´.

´96´ || ´01´ || Le   or   ´97´ || ´01´ || Le

A data object with tag ´96´ or ´97´ is referred to as an Le data object. Such data objects are placed in the data field´ of a command message.

An Le data object with tag ´96´ is not included in the MAC security of the command message, whereas a data object with tag ´97´ is included.

– If the command is not MAC-secured but an encrypted command message has been received, or MAC security or encryption is to be used for the response message, then you must use a data object with tag ´96´. In this situation MICARDO will reject tag ´97´ as an error.
– Tag ´96´ is generally also used if the command message is MAC-secured. It indicates that the Le data object is not included in the MAC security. However, MICARDO will not reject a tag ´97´ that is used in such a case when the Le data object is included in the MAC security.

5.6.2.3 Data Objects of Type Status Information

If a command whose response does not contain a data field is executed under *secure messaging* and the response message is to be MAC-secured, then the return code (SW1 and SW2) is inserted as a value field in a simple data object with tag ´99´ and length ´02´.

´99´ || ´02´ || SW1 SW2

A data object with tag ´99´ is referred to as a status information data object. Such a data object is placed in the data field´ of a response message and included in the MAC security of the response message.

The return code is inserted in the data only when

– The response does not contain a data field and
– The code is either a positive return code or a warning.

The positive return codes are return code ´90 00´ and – for commands relating to cardholder authentication and component authentication – also return code ´63 CX´.
Warnings that are not output together with a data field are return code ´62 82´ for the command **SEARCH RECORD** and return code ´63 CX´ – except when it is output by a command relating to cardholder authentication and component authentication.

For commands whose response does not contain any response data the unsecured command message is laid out as follows:

'X0'	INS	P1	P2	Lc	Data field

Le is not present, because no response data is expected.

If the response to such a command is to be MAC-secured the command message must contain an Le´ byte with the value ´00´, which requests a data field´ in the secured response message containing a data object of type status information:

CLA	INS	P1	P2	Lc´	Data field´	'00'

In this case the class byte has one of the values ´XC´ and ´X8´.

The response message is then laid out like this

Data field´	SW1	SW2

where the data field´ contains at least the data object of type status information.

5.6.2.4 Data Objects of Type MAC

If a command is to be executed under *secure messaging* and the command and/or its response are to be MAC-secured, then the message is first composed with a TLV coded data field´ for the secured command without the MAC.
The MAC is then computed over the message.

The MAC, or from 4 to 7 of its most significant bytes, is inserted as a value field in a simple data object with tag ´8E´ and length ´08´, or length ´04´ to ´07´ as appropriate, for example:

´8E´ || ´08´ || MAC

A data object with tag ´8E´ is referred to as a MAC data object. The MAC data object is placed in the data field´ of the command or response message.

The MAC data object must not be used if it is not possible to compute a 'sensible' MAC. This will be the case when:

– There is no class byte with the value ´XC´ ,
– There is no SM data object with an odd tag in the command,
– There is no SM data object with an odd tag in the response.

5.6.2.5 Data Objects of Type Response Descriptor

If a command is to be executed under *secure messaging* you can place a compound data object with tag ´BA´ in the data field´ of the command message, in order to specify the requirements for the security of the response message.

´BA´ || L || CRT

Such a data object is referred to as a data object of type response descriptor.

This data object can contain a CCT and/or a CT. At least one of the two CRT data objects must be present:

Tag	Length	Value
´B4´	´XX...YY´	CRT for data authentication (CCT)
´B8´	´XX...YY´	CRT for data confidentiality (CT)

If a command message contains a data object of type response descriptor then the response message will be MAC-secured (for CCT) and/or encrypted (for CT).

If a command contains a data object of type response descriptor and the command is to be MAC-secured, then this data object will not be included in the MAC security of the command message.

5.6.2.6 Data Objects of Type CRT

When a command is to be executed under *secure messaging* you can place the CRT data objects CCT and CT in the data field´ of the command and/or the response message. These components are used to specify the security requirements for the command and/or response message.

The CRT data objects are handled as compound data objects with tags ´B4´ and ´B8´ respectively. The following data objects can be included:

Tag	Length	Value
´83´	´01´ or ´03´	Key reference: KV or ´X0´ || KID || KV
´87´	´08´	ICV

Within a command message the CRT data objects may occur in the actual data field´ itself or within the data object of type response descriptor.

If the command message contains a CRT data object then the command message must be MAC-secured and/or encrypted.

The data field´ of a response message can meaningfully contain only one CT. If a response message does contain a CT then the message must be encrypted.

5.6.3 Calculating the MAC

The code of the class byte determines whether the command header is included in the calculation of the MAC (CLA = ´XC´) or not (CLA = ´X8´).

If the command and response messages contain a data field´ then it is TLV coded. All data objects with an **odd** tag in the data field´ are included in the corresponding MAC calculation.

Command:
The full set of data that has to be included in the MAC calculation for a command message (including padding) is laid out as follows.

CLA = ´XC´ - command header is included:

CLA || INS || P1 || P2 || ´80 00 00 00´ || ...
...Data block$_1$|| ´80´ || ´00…00´ ||...

...||
...Data block$_n$ || ´80´ || ´00…00´

CLA = ´X8´ - command header not included:

Data block$_1$ || ´80´ || ´00…00´ ||…|| Data block$_n$ || ´80´ || ´00…00´

Response:
The full set of data that has to be included in the MAC calculation for a response message (including padding) is laid out as follows:

Data block$_1$ || ´80´ || ´00…00´ ||…|| Data block$_n$ || ´80´ || ´00…00´

Data objects containing an odd tag which are adjacent in the command or response data fields are here referred to as blocks.

Command data or response data can only contain more than one data block if it contains data objects with an odd tag separated by at least one data object with even tag.

Each data block is supplemented by ISO padding to give it a length that is a multiple of 8 bytes.
The bytes of ISO padding are significant only for the MAC calculation and are not transferred together with the command or response message.

5.6.4 Encryption

ISO padding is also used for encryption under *secure messaging*. Encrypted padding bytes are a part of the cryptogram that has to be transferred.

5.7 Security Environments

5.7.1 Basics

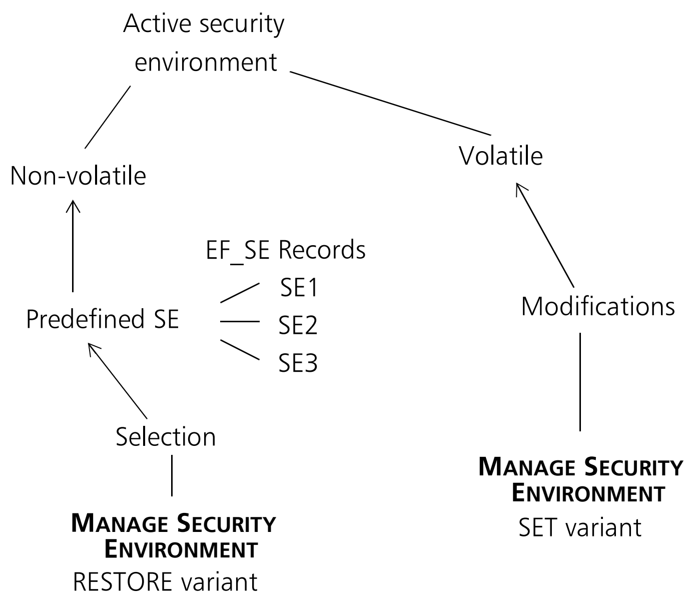
A security environment (SE) describes the preset security features and key references that are referenced in security-related commands and *secure messaging*.

A security environment governs the following points:

- For each directory there is a non-volatile record of which security features are supported by the cryptographic keys belonging to an SE.
  In the additional information for each key in the data field EF_KeyD you can record for each SE# whether that key is able to support the relevant procedure and if so, which cryptographic algorithm is to be used for this purpose.
  If a key is to be used for one and the same procedure but with different cryptographic algorithms you can associate it with the different cryptographic algorithms within different SEs.
- For each SE# there is a non-volatile record in the additional information for passwords showing whether the password may be used in that environment, and if so, subject to which access rule(s) and in which transfer format. This information is stored in the data field EF_PwdD.
- For each SE# there is a non-volatile record in the additional information for keys showing whether the key may be used in that environment, and if so, subject to which access rule(s). This information is stored in the data field EF_KeyD.
- For each directory there is a non-volatile record of which access rules apply within an SE. The data field EF_Rule correlates access rules with passwords, keys and data fields for each SE#.
- You can optionally maintain a non-volatile record in the data field EF_SE in each directory showing which keys are to be used for certain security features for the various SE#.

5.7.2 Components

An SE consists of non-volatile and volatile stored components. A number of SEs can be predefined for a given directory. They are identified by a serial number (SE#) between ´01´ and ´FE´ (´EF´ being omitted).



At each reset MICARDO activates a global SE by means of its SE#. This associates an active SE with each directory.

These assignments can be modified using the RESTORE variant of the command **MANAGE SECURITY ENVIRONMENT**. Such a change activates the specified SE only for the current directory, the global SE# remains active for all other directories.
The SET variant of the command **MANAGE SECURITY ENVIRONMENT** can be used to temporarily change the SE of a current directory.

5.7.3 Predefined Elements in the Data Field EF_SE

Each directory can optionally contain a data field EF_SE, in which elements can be predefined for each SE# ready for when it is activated.

The number in the first byte will generally not correspond to the record number; the numbers do not need to be sorted in ascending order. The following rules apply:

- If the first byte of the record is non-zero it is an SE#. In this case, the data objects contained in this record apply to the specified SE and the directory in which this EF_SE data field is located.
- If the first byte of the record is zero, the data objects contained in this record apply to all those SEs, whose numbers are not present as the first byte of any of the other records.
- If the first byte has the value ´EF´ or ´FF´ the record is not relevant to any SE and is therefore superfluous; MICARDO does, however, tolerate such records.

The data objects may be specified in any desired order. If any optional data objects are omitted, the specified default values will be used.

Data contents: 1st byte and 5 compound data objects		
Tag	**Length**	**Value**
-	´01´	Binary coded SE#
´A4´	´XX...YY´	CRT for authentication (AT)
´B4´	´XX...YY´	CRT for data authentication (CCT)
´B6´	´XX...YY´	CRT for digital signatures (DST)
´B8´	´XX...YY´	CRT for data confidentiality (CT)

If you do not adhere to the order and specified lengths of the CRT data objects described below, MICARDO will consider the data to be inconsistent.

If a record does not contain a CRT data object, then no corresponding information is defined for the SE specified by its first byte.

5.7.3.1 CRT for Authentication (Tag ´A4´)

A CRT for authentication (AT) is a compound data object with tag ´A4´. The items should be present in the order shown; optional data objects (o) may be omitted.

Tag	Length	Value	
´95´	´01´	Usage qualifier, default = ´80´	o
´83´	´03´	Key reference: ´X0´ || KID || KV	
‘80’	‘01’	DSI format (RFU)	o

5.7.3.2 CRT for Data Authentication (Tag ´B4´)

The CRT for data authentication (CCT) is a compound data object with tag ´B4´. The items should be present in the order shown; optional data objects (o) may be omitted.

Tag	Length	Value	
´95´	´01´	Usage qualifier, default = ´20´	o
´83´	´03´	Key reference	

5.7.3.3 CRT for Digital Signatures (Tag ´B6´)

The CRT for digital signatures (DST) is a compound data object with tag ´B6´. The items should be present in the order shown; optional data objects (o) may be omitted

Tag	Length	Value	
´95´	´01´	Usage qualifier, default = ´80´	o
´83´	´03´	Key reference: ‘X0´ || KID || KV	
‘80’	‘01’	DSI format (RFU)	o

5.7.3.4 CRT for Data Confidentiality (Tag ´B8´)

The CRT for data confidentiality (CT) is a compound data object with tag ´B8´. The items should be present in the order shown; optional data objects (o) may be omitted.

Tag	Length	Value	
´95´	´01´	Usage qualifier, default=´20´	o
´83´	´03´	Key reference	

A record may contain any number of CTs. MICARDO takes at most four CRTs into account for encryption and decryption.

5.7.3.5 Sample CRT

The contents of a record in the data field EF_SE are as follows:

```
SE# CCT                               CCT
02||B4 08                           ||B4 05
          UQ        Key ref.          Key ref.
         (95 01 30||83 03 (80 12 13))     (83 03 (80 12 14))
```

This record contains specifications for SE# = ´02´.
The first CCT will be ignored because the usage qualifier does not agree.
The second CCT will be recognised and a key reference will be extracted from it and will be used for *secure messaging* of response messages, as indicated by its usage qualifier = ´20´ (default).

5.7.4 Volatile Elements

The SET variant of the command **MANAGE SECURITY ENVIRONMENT** can be used to temporarily modify the active SE in the current directory.

The parameters P1 and P2 for this command specify which CRT data objects are to be modified. The command data then contains the new values for these CRT data objects, which replace the previous values.
Such CRT data objects may also be empty. This can be used, for example, to delete key references.

The active SE contains other elements in addition to the predefined CRT data objects. These are the volatile elements that are dealt with here.

1. Session key SK1, session key SK2, send sequence counter (SSC):
   These elements are agreed upon in the course of a mutual authentication and available only to *secure messaging*.
2. IcvSecureMessaging:
   This ICV is passed using the command **MANAGE SECURITY ENVIRONMENT**.
   It is used in *secure messaging* for the security or encryption of response messages.
3. IcvPSO:
   This ICV is passed using the command **MANAGE SECURITY ENVIRONMENT**.
   It is used by the PSO commands of DES-based cryptographic algorithms.

Here is how MICARDO deals with elements of the security environment that are modified using the command **MANAGE SECURITY ENVIRONMENT**:

– All modifications performed using the SET variant lose their validity when the chip card is reset.
– CRT data objects modified using the SET variant apply only to the active security environment and current directory at the time the command is executed.
– CRT data objects stored in the data field EF_SE are not permanently overwritten or deleted by the SET variant.
– For as long as they are valid, CRT data objects modified using the SET variant will override any CRT data objects previously set or cleared using the SET variant and those stored permanently in the data field EF_SE.
– When the RESTORE variant is used to activate a security environment, all changes previously made using the SET variant are revoked.
  This also applies if the previously active security environment is activated afresh using the RESTORE variant.
– If you leave the current directory, all changes previously made using the SET variant are revoked.

5.7.5 Cross-SE Key Derivation Data

You can use the SET variant of the command **MANAGE SECURITY ENVIRONMENT** to pass MICARDO additional or alternative key derivation data (CID) for deriving a card-specific key from a master key.

You do this by placing the key derivation data in the value field of a data object with tag ´94´ (CID data object) and including this in the command data of the SET variant. The CID data object may also be empty (´94 00´). This enables you to reference key derivation data that was passed earlier.

When the key derivation data is stored it will clear any CIDs that were previously passed and any keys that were derived from them. If keys derived from the previous CIDs have been used to derive or negotiate further keys then these will also be cleared.

If it is not possible to successfully execute the command then the previously stored key derivation data and derived keys will remain unchanged.

Key derivation data is retained in memory until the chip card is reset or new data is successfully passed to the chip card using the SET variant of the command **MANAGE SECURITY ENVIRONMENT**.

Provided the active SE and the algorithm ID of the relevant master key permit, the stored key derivation data can subsequently be used to derive card-specific keys regardless of which directory is selected and which SE is active when the command was executed. Key derivation data is thus a component which can apply across all the SEs.

5.7.6 Active SEs of Inactive Directories

The following section describes how the active SE appears in a directory that is not currently selected.

If the directory in question

– has no data field EF_SE, then its active security environment contains only empty CRTs,

– does contain a data field EF_SE but this does not contain any records whose SE# matches that of the active SE, then its active SE contains only empty CRTs,

– does contain a data field EF_SE which does contain a record with a matching SE#, then the CRT data objects of this record apply in the active SE.
If the record does not contain all possible CRT data objects then those that are missing are considered to be empty.

If one record contains several correctly coded CRT data objects for the same purpose (for example, two authentication CRTs for the command **EXTERNAL AUTHENTICATE**), then that CRT data object which is closest to the beginning of the record will take precedence.

When a directory is selected, the contents of its active SE are exactly the same after it is selected as they were before.

The contents of the CRT data objects in the active SE of a selected directory can be modified using the command **MANAGE SECURITY ENVIRONMENT**.
In the active SE of the currently selected directory you can also negotiate the session keys and SSC using the commands **GET CHALLANGE** and **MUTUAL AUTHENTICATE**, and pass IcvSecureMessaging and IcvPso using the command **MANAGE SECURITY ENVIRONMENT**.

5.8 Security Status

MICARDO operates on objects. Of these, the object classes data field, password and key need particular protection because they contain ciphers which should not be generally accessible.

For security reasons, therefore, commands which wish to use the objects in the above classes must examine the access rules belonging to the particular object.

Access rules principally refer to security status values. A *security status* records the access rights acquired through authentication.

Security status values that apply to the master file are referred to as **global security status values**. Security status values that apply to a directory other than the master file are referred to as **DF-specific security status values**.

In order to modify a security status the external world must first authenticate itself to MICARDO. This may be done

– as a cardholder authentication, by furnishing proof of the knowledge of cardholder-specific data (passwords) using one of the commands **VERIFY**, **CHANGE REFERENCE DATA** or **RESET RETRY COUNTER**. As appropriate this may modify the security status of the directory relevant to the specific password.

– as a component authentication, by furnishing proof of the possession of a cryptographic key using the commands **External Authenticate** or **Mutual Authenticate**. As appropriate this may modify the security status of the directory relevant to the specific key.

To record which password was used for a cardholder authentication, or which key for a component authentication, it is sufficient to store the following information for each directory:

– The password number (PwdID) with which each cardholder authentication was performed,
– The key reference of the secret key with which each component authentication was performed.

MICARDO also performs data authentication, but the result of such an authentication is not retained and has no effect on the security status of the chip card.

5.9 Access Rules

Access rules are used to specify under which circumstances which types of access to the object classes file, password and key are to be permitted.

Access types:
The access type specifies which commands may access an object.
Examples: **Read Binary, Verify, Internal Authenticate**

Access conditions:
An access condition specifies which conditions have to be fulfilled before an object may be accessed.

Examples:
– Access is always permitted
– Access is permitted provided the accessing command is using *secure messaging* and a cardholder authentication with password number ´02´ has taken place.

We shall use the term access rules (ARs) to cover both elementary access rules and combinations of access rules using the OR operator.

5.9.1 Data Field EF_Rule

The defined access rules for a directory are stored in the data field EF_Rule in that directory. This field is always uniquely identifiable within the directory under the FID ´00 30´.
It is also permitted to store access rules in other data fields within a directory. These data fields taken together are then referred to as an AR EF.

One access rule is stored in each record of an AR EF. Depending on the length of the access rules the field type may be linear, variable or – if all the records are of the same length – linear, constant.

Within an AR EF access rules are referenced using the corresponding record number. Within a directory an access rule may either be identified only by a record number, in which case it is located in the data field EF_Rule, or by an FID and a record number, in which case access rule is located in a record of some data field other than EF_Rule.

5.9.2 ARRs per Transmission Type

A reference to an access rule is referred to as an ARR (access rule reference). This reference is stored together with the type of transmission (interface depending) in a data object that is also referred to as an ID ARR data object.

This is a compound data object with the tag ´A1´.

Tag	Length	Value
´91´	´01´	Interface byte optional
´8B´	´XX´	Access rule reference

The following table shows where the ID ARR data objects for operating system commands are located, depending on the command and object.

Command	Location of the relevant ID ARR data object
Activate File, **Deactivate File**	File control parameters (FCP) in the file header of the directory or the data field to which the command refers
Read Binary, **Read Record**	File control parameters (FCP) in the file header of the data field which should be read
Search Binary, **Search Record**	File control parameters (FCP) in the file header of the data field in which should be searched
Append Record, **Update Binary**, **Update Record**	File control parameters (FCP) in the file header of the data field in which should be written
Create File	File control parameters (FCP) in the file header of the current directory
Delete File (DF)	File control parameters (FCP) in the file header of the current directory
Delete File (EF)	File control parameters (FCP) in the file header of the data field that is to be deleted
Change Reference Data, **Reset Retry Counter**, **Verify**	SE data object of the relevant password
PSO commands, **Internal Authenticate**, **External Authenticate**, **Mutual Authenticate**, **Generate Public Key Pair**	SE data object of the relevant key

5.9.2.1 Interface Byte

The interface byte is encoded as follows:

b8	b7	b6	b5	b4	b3	b2	b1	Description
x	x	x	x	x	x	...	...	RFU, the status of these bits is ignored

b8	b7	b6	b5	b4	b3	b2	b1	Description
...	...	...	...	...	...	1	...	Contactless transfer (not implemented)
...	...	...	...	...	...	...	1	Contact-based transfer

5.9.2.2 Access Rule Reference

An access rule reference is stored in an access rule reference data object.

This is a simple data object with the tag ´8B´. It can assume the following lengths and values:

Length	Significance
´01´	Record number (binary coded) which references an access rule in EF_Rule; the access rule applies to all active SEs
´03´	FID and record number which references an access rule in a different data field within this directory; the access rule applies to all active SEs
2+<n>*2	FID and <n> pairs of SE# and record number If the SE# has the value ´00´, the referenced access rule applies to all SEs, unless this SE# is associated with some other record number in another such pair.

The TLV structure defines a length coding dependent on the length of the data object (see section "Data Objects" on page 15). In case of the ARR data object MICARDO differs from /ISO 8825-1/ and codes the length always in one byte.

To evaluate this data object we use the SE# of the directory in which the data object is located and also look in that directory for the AR EF and the access rule.

5.9.3 Access Modes

The access mode within an access rule is specified by a list of data objects, called AM data objects. Such a list may consist of one or more AM data objects.

An AM data object is a simple data object with a tag in the range ´80´ to ´8F´. Its value field describes the commands for which the access rules are defined. The value field is coded differently depending on the tag of the data object.

Tag	Length	Value
´80´	´01´	Bit map for operating system commands (AM byte)
´81´...´8F´	´XX...YY´	Command definition including CLA, INS, P1, P2

5.9.3.1 AM Bytes

Operating system commands are specified by the AM byte in the AM data object with tag ´80´.

The interpretation of the AM byte depends on which object the corresponding access rule applies to.

AM byte for access to directories:

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	...	...	...	...	...	...	...	b7..b1 according to /ISO 7816-9/
...	1	...	...	...	...	...	...	**DELETE FILE** (self)
...	...	x	...	...	...	...	...	RFU
...	...	...	1	...	...	...	...	**ACTIVATE FILE**
...	...	...	...	1	...	...	...	**DEACTIVATE FILE**
...	...	...	...	...	1	...	...	**CREATE FILE** (DF)
...	...	...	...	...	...	1	...	**CREATE FILE** (EF)
...	...	...	...	...	...	...	1	**DELETE FILE** (child DF)

AM byte for access to data fields:

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	...	...	...	...	...	...	...	b7..b1 according to /ISO 7816-9/
...	1	...	...	...	...	...	...	**DELETE FILE** (EF)
...	...	x	...	...	...	...	...	RFU
...	...	...	1	...	...	...	...	**ACTIVATE FILE**

b8	b7	b6	b5	b4	b3	b2	b1	Description
...	...	...	...	1	...	...	...	DEACTIVATE FILE
...	...	...	...	...	1	...	...	APPEND RECORD
...	...	...	...	...	...	1	...	UPDATE BINARY, UPDATE RECORD
...	...	...	...	...	...	...	1	READ BINARY, READ RECORD, SEARCH BINARY, SEARCH RECORD

AM byte for access to passwords and keys:

b8	b7	b6	b5	b4	b3	b2	b1	Description
1	...	...	...	...	...	...	...	b7..b1 proprietary use
...	x	x	...	...	...	...	...	RFU
...	...	...	1	...	...	...	...	INTERNAL AUTHENTICATE
...	...	...	...	1	...	...	...	EXTERNAL AUTHENTICATE
...	...	...	...	...	1	...	...	RESET RETRY COUNTER
...	...	...	...	...	...	1	...	CHANGE REFERENCE DATA
...	...	...	...	...	...	...	1	VERIFY

For keys that are accessed by PSO commands you will need to use the procedure described in the following section, because the usage of the key cannot be established until the parameters P1 and P2 are encountered.

A non-zero bit means that the rule applies to the specified command. If more than one bit is set then the rule applies to all those commands for which the corresponding bit is set.

At this point the current values of the class byte and the parameters P1 and P2 are not relevant for the purposes of checking validity.

5.9.3.2 Command Definition

If you need to specify the access type for a command more precisely, for example if you wish to specify different access rules for different values of P1 or P2, then you will have to specify the access type for the corresponding operating system command using a command definition (tags ´81´ to ´8F´).

An AM data object with a tag between ´81´ and ´8F´ contains a description of one or more commands specified by their class byte, instruction byte, P1, or P2.

Tag for an AM data object containing command definitions:

b8	b7	b6	b5	b4	b3	b2	b1	Description
1	...	...	...	...	...	...	...	all other values are erroneous
...	0	...	...	...	...	...	...	
...	...	0	...	...	...	...	...	
...	...	...	0	...	...	...	...	
...	...	...	...	1	...	...	...	CLA is present
...	...	...	...	...	1	...	...	INS is present
...	...	...	...	...	...	1	...	P1 is present
...	...	...	...	...	...	...	1	P2 is present

All the commands in any one AM data object with tag ´81´ to ´8F´ are described in the same way. The descriptive bytes must be specified in the following order.

CLA, INS, P1, P2.

Depending on whether the commands are described using 1, 2, 3 or 4 bytes, the length bytes of the AM data object indicate how many commands an access rule applies to.

For example, the value field of an AM data object with tag ´84´ and length ´05´ contains the instruction bytes of five commands.
An AM data object with tag ´87´ must have a length that is a multiple of 3. The value field of an AM data object with tag ´87´ and length ´0F´ contains instruction bytes, P1, and P2 for five commands.

5.9.4 Access Conditions

MICARDO provides a number of ways of protecting data from unauthorised access and protecting passwords and keys from unauthorised use.

This is effected by defining the following types of access conditions (security conditions) and storing them in an SC data object.

The following table shows the possible tags and values of an SC data object and specifies which access condition they code for.

Tag	Length	Value	Explanation
´90´	´00´		ALW
´97´	´00´		NEV
´A4´	´XX...YY´	UQ and CR data object	Access condition of type AUT or PWD
´B4´	´XX...YY´	UQ and CR data object	Access condition of type SM for MAC security, PRO SM SC
´B8´	´XX...YY´	UQ and CR data object	Access condition of type SM for encryption, ENC SM SC
´BA´	´XX...YY´	Command-specific	Access condition of type SPEC
´A0´	´XX...YY´		OR template
´AF´	´XX...YY´		AND template

Each SC data object contains exactly one of these data elements.

5.9.4.1 Access Condition of Type PWD

The value field of an access condition of type PWD contains the following data objects in the specified order:

Tag	Length	Value
´95´	´01´	´08´: Usage qualifier for cardholder authentication
´83´	´02´	Password reference: ´X0´ || PwdID

5.9.4.2 Access Condition of Type AUT

The value field of an access condition of type AUT contains the following data objects in the specified order:

Tag	Length	Value
´95´	´01´	´80´: Usage qualifier for external authentication
´83´	 ´02´ ´03´	Key reference for component authentication: ´X0´|| KID ´X0´|| KID || KV

5.9.4.3 Access Conditions of Type SM

Access conditions of type SM (*secure messaging*) specify how the command and response for the commands listed in the access type are to be secured under *secure messaging*.

The following conditions are possible:

– neither MAC security nor encryption
– only MAC security
– only encryption
– both MAC security and encryption

The independent allocation of these possibilities to the command and response gives us a total of 15 different types of *secure messaging*.

To represent the 15 different types of *secure messaging* and security conditions (SM SC) you can use an AND template to combine access conditions. The following abbreviations are defined for this purpose:

– PRO SM SC:
  An access condition which defines only MAC security for a command, a response, or both.
– ENC SM SC:
  An access condition which defines only encryption for a command, a response, or both.

Secure messaging according to ISO does not permit a command and/or a response to be secured with two or more MACs or two or more encryptions.

5.9.4.4 PRO SM SC (Tag ´B4´)

A PRO SM SC is coded using a CCT (CRT for data authentication) with tag ´B4´. The value field comprises data objects from the following list in the specified order, whereby optional data objects (o) may be omitted.

Tag	Length	Value	
´95´	´01	Usage qualifier, default = ´30´	o
´83´	 ´02´, ´03´	Key reference: ´X0´|| KID ´X0´|| KID || KV	
´8B´	´01´	Not relevant to responses; for commands: ´00´: header may be MAC-secured but need not ´01´: header must be MAC-secured (default)	o
´8E´	´01´	Minimum length of the MAC, default = ´08´	o

If a data object with tag ´8E´ is present it specifies the minimum length of a MAC calculated on the command message, or the precise length of a MAC calculated on the response message.
If fewer than 8 bytes of a calculated MAC are to be output or received these will be the most significant L bytes (length byte) of the MAC calculated from the message.

5.9.4.5 ENC SM SC (Tag ´B8´)

An ENC SM SC is coded using a CT (CRT for encryption and decryption) with tag ´B8´. The value field comprises data objects from the following list in the specified order, whereby optional data objects (o) may be omitted.

Tag	Length	Value	
´95´	´01´	Usage qualifier, default = ´30´	o
´83´	 ´02´ ´03´	Key reference: ´X0´|| KID ´X0´|| KID || KV	

5.9.4.6 Access Condition of Type SPEC (Tag ´BA´)

An access condition of type SPEC can be used for command-specific definitions in access rules. The value field of an access condition of type SPEC is not evaluated by MICARDO, it is passed for evaluation and processing to the commands listed in the access type.

Such a command can use the supplied information to check whether the necessary conditions for its execution are fulfilled.
For example, an access condition of type SPEC may also include CRT data objects containing key references, enabling the command to calculate MACs for the command data or response data using those referenced keys.

The specification of a command that uses an access condition of type SPEC must specify how to construct the return value for the SC data object of type SPEC.

5.9.4.7 OR Template (Tag ´A0´)

An SC data object of type OR template may contain any number of SC data objects of the types ALW, AUT, PWD, PRO SM SC, ENC SM SC, SPEC and AND templates in any desired order.

5.9.4.8 AND Template (Tag ´AF´)

An SC data object of type AND template may contain any number of SC data objects of the types AUT, PWD, PRO SM SC and ENC SM SC, but only one of type SPEC. These SC data objects may occur in any desired order.

5.9.5 Combinations and Evaluations

An access rule consists of a list of one or more AM data objects to which exactly one SC data object has been appended by concatenation (||).

AM data object_1 [|| AM data object_2 || .. || AM data object_n] || SC data object

In determining whether an access condition for a command is fulfilled, the program searches the list of AM data objects in the corresponding access rule from left to right for a reference to the command. Note that in general even a single AM data object amounts to a list of commands.

A given command may be referenced a number of times in the list of AM data objects, both within the same AM data object and in different ones.
For example, a command might be referenced in one AM data object by its instruction byte and in another AM data object by the class byte and the instruction byte, or it might be referenced within the same AM data object by its instruction byte and by two different values of P1.

The command may be executed if and only if the SC data object is fulfilled, otherwise the command will abort.

A command will be found in an AM data object with tag ´80´ if and only if bit b8 of the AM byte and the command code agree and the corresponding bit of the AM byte is set.

Example:
The AM data object = ´80 01 01´ applies to the command **DELETE FILE** (child DF) and the commands **READ** and **SEARCH,** but not to the command **VERIFY**.

5.9.6 Combining Access Rules with OR

One record of an AR EF can contain a number of access rules connected by ORs. Such an OR link is constructed by concatenation.

access rule_1 || … ||access rule_n

The command may be executed if and only if at least one access rule is fulfilled, otherwise the command will abort. The analysis is performed from the left to the right.

5.10 Algorithm IDs

The combination of cryptographic algorithms, modes and security features constitutes an unambiguous procedure that is identified by an algorithm ID.

An algorithm ID is of variable length. The individual digits of each algorithm ID are encoded as half-byte BCD. Only those algorithms whose IDs are shown with specific lengths in the following table are actually implemented.

Algorithm IDs for symmetrical procedures:

Procedure	Algorithm ID and abbreviation	ID length in bytes
Cryptographic algorithms	{ 1 }	
Encryption	{ 1 1 }	
DES	{ 1 1 1 }	
CBC, ICV = 0	{ 1 1 1 1 }	2
CBC, ICV ≠ 0	{ 1 1 1 2 }	2
DES3	{ 1 1 2 }	
CBC, ICV = 0	{ 1 1 2 1 }	2
CBC, ICV ≠ 0	{ 1 1 2 2 }	2
Cryptographic checksum	{ 1 2 }	
DES (simple MAC)	{ 1 2 1 }	
CBC, ICV = 0	{ 1 2 1 1 }	2
CFB, ICV ≠ 0	{ 1 2 1 2 }	2
DES3 (retail MAC)	{ 1 2 2 }	
CBC, ICV = 0	{ 1 2 2 1 }	2
CFB, ICV ≠ 0	{ 1 2 2 2 }	2
Authentication	{ 2 }	
One-way authentication	{ 2 1 }	
Internal authentication	{ 2 1 1 }	
DES	{ 2 1 1 1 }	2
DES3	{ 2 1 1 2 }	2
External authentication	{ 2 1 2 }	
DES	{ 2 1 2 1 }	2
DES3	{ 2 1 2 2 }	2
Mutual authentication	{ 2 2 }	
External authentication first	{ 2 2 1 }	
DES	{ 2 2 1 1 }	2
DES3	{ 2 2 1 2 }	2

Procedure	Algorithm ID and abbreviation	ID length in bytes
Key management	{ 3 }	
Key agreement	{ 3 1 }	
Key derivation according to HPC	{ 3 1 2 }	
DES3	{ 3 1 2 1 }	2
Key transport	{ 3 2 }	
Negotiation during authentication	{ 3 2 1 }	
DES3 (also for session keys)	{ 3 2 1 1 }	
1 session key without SSC	{ 3 2 1 1 1 0 }	3
2 session keys without SSC	{ 3 2 1 1 2 0 }	3
1 session key with SSC	{ 3 2 1 1 3 0 }	3
2 session keys with SSC	{ 3 2 1 1 4 0 }	3

Algorithm IDs for asymmetrical procedures (private keys):

Procedure	Algorithm ID and abbreviation	ID length in bytes
Cryptographic algorithms	{ 1 }	
Decryption	{ 1 1 }	
RSA	{ 1 1 3 0 }	2
Signature	{ 1 3 }	
RSA	{ 1 3 1 0 }	2
Authentication	{ 2 }	
One-way authentication	{ 2 1 }	
Internal authentication	{ 2 1 1 }	
RSA	{ 2 1 1 3 }	2

5.11 Security Features for the EEPROM

MICARDO possesses a procedure for protecting the contents of the EEPROM which is an extension to the ISO/CEN procedure. This enables sensitive data (program code, keys, passwords, administrative data, field contents etc.) to be protected from inadvertent alteration (e.g. defective memory cells). The chip card recognises inconsistencies both when it is loaded or personalised and also while running, and reports them to the external world. The available mechanisms are listed below.

5.11.1 Checking Header Information

All the headers of directories and data fields are provided with checksums (CRC). The internal flags and pointers are considered part of the header. This all takes place internally and is entirely transparent for the external world.

Since the headers contain data which is essential to the correct functioning of the card (file sizes, file types, etc.) defective header information makes it impossible for processing to continue.

The correctness of a header is checked for the relevant directories and data fields by the command **SELECT**. The header is also checked during an internal select performed within the card, when the execution of a command (such as **VERIFY, INTERNAL AUTHENTICATE, EXTERNAL AUTHENTICATE** etc.) call for data from a field that is not currently selected. Checksum errors return the code ´64 00´.

If the file header has been corrupted (checksum error occurs) the data contained in this file is no longer accessible.

5.11.2 Checking Transparent Data Fields

MICARDO calculates checksums (CRC) over transparent data fields. All commands which read or modify data verify these checksums and recalculate them as necessary.

The command **UPDATE BINARY** corrects checksums for new field contents. If there is an error in the field contents then **UPDATE BINARY** can be executed only with contents for the entire field; this enables the contents to be re-stored.

5.11.3 Checking Formatted Data Fields

Checksums (CRC) are also calculated for individual records. All commands which read or modify data verify these checksums and recalculate them as necessary.

The command **UPDATE RECORD** corrects the checksum of a record for new field contents.
If there is an error in the record contents then you can perform an **UPDATE RECORD** on the record; this enables the contents to be restored.

6 Operating System Commands

6.1 Layout of the Command Descriptions

All the command descriptions are laid out in the following way:

– explanation of the command
– command APDU
– response APDU if the command was completed successfully
– response APDU in the event of an error

Except where otherwise stated, all the information refers to unsecured transmission of the command and its response.

You will find information regarding the layout of APDUs on page 30.

6.2 Overview of all the Commands

The following table lists all MICARDO's operating system commands in alphabetical order, each with the code for the instruction byte (INS) in the command header and a brief description of the function.

Command	INS	Brief description	See page
ACTIVATE FILE	´44´	Unlocks a directory or data field	150
APPEND RECORD	´E2´	Creates new records in a formatted data field	152
CHANGE REFERENCE DATA	´24´	Changes the password for cardholder authentication	153
CREATE FILE	´E0´	Creates a directory or data field	156
DEACTIVATE FILE	´04´	Locks a directory or data field	158
DELETE FILE	´E4´	Deletes a data field or a directory including its subdirectories	160
EXTERNAL AUTHENTICATE /MUTUAL AUTHENTICATE	´82´	Authenticates the external world / external world and chip card	162
GENERATE PUBLIC KEY PAIR	'46'	Generates the public and the private part of a RSA key pair.	165
GET CHALLENGE	´84´	Generates and outputs a random number	166
GET RESPONSE	´C0´	Reads out the response data (T=0)	168

Command	INS	Brief description	See page
Internal Authenticate	´88´	Authenticates the chip card or application	169
Manage Security Environment	´22´	Passes on key references, random numbers, and data to be used for key derivation	171
Perform Security Operation	´2A´	Various functions using symmetrical and asymmetrical keys: – **Compute Digital Signature** – **Decipher** – **Hash**	174 176 178 180
Read Binary	´B0´	Reads from a transparent data field	182
Read Record	´B2´	Reads from a formatted data field	184
Reset Retry Counter	´2C´	Resets the counter of failed attempts	186
Search Binary	´A0´	Searches in a transparent data field	188
Search Record	´A2´	Searches in a formatted data field	190
Select File	´A4´	Selects a directory or data field	193
Update Binary	´D6´	Modifies a transparent data field	195
Update Record	´DC´	Modifies a formatted data field	197
Verify	´20´	Authenticates the cardholder	199

6.3 Command Classes

The following table lists the possible values for the class byte (CLA) in the command header.

b8...b5	b4	b3	b2	b1	Description
x					Type of command :
´00 00´	...	...	...	...	Command and response APDU as specified by ISO; in chaining mode, this is the only command or the last one
´00 01´	...	...	...	...	Indicates chaining mode as specified by ISO
...	x	x	...	...	*Secure messaging* format (SM):
...	0	0	...	...	No SM, or SM is not shown
...	0	1	...	...	RFU, treated as an error by MICARDO
...	1	0	...	...	SM, file header is not MAC-secured
...	1	1	...	...	SM, file header is MAC-secured
...	...	...	x	x	Logical channel number

6.4 ACTIVATE FILE

The command **ACTIVATE FILE** is the counterpart of the **DEACTIVATE FILE** command. It reversibly places the current file in the **operational state - activated**. The file can subsequently be processed in the usual way.

This command refers only to whichever file is currently selected. If, say, you wish to activate an entire application, you may find it appropriate to alternate this command with the command **SELECT FILE** for each of the files belonging to the application, one after another.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´44´	Command code

	Length	Contents	Description
P1	1	´00´ ´02´	Activate current DF Activate current EF
P2	1	´00´	Fixed value
Lc	0	-	Empty
Data	0	-	Empty
Le	0	-	Empty

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	File has been activated
		´63 CX´	As for ´90 00´, but problems with writing

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	Inconsistent or incorrect data
		´65 81´	Write error
		´67 00´	Response data has incorrect length, data field is superfluous, Le is superfluous
		´69 82´	Security conditions are not fulfilled
		´69 86´	No EF is selected
		´69 88´	Incorrect data objects for secure messaging
		´6A 86´	P1 or P2 possesses an illegal value
		´6E 00´	CLA and INS are inconsistent

6.5 Append Record

The command **Append Record** appends an additional record to a formatted data field. In the case of a cyclic data field, the record will be appended if there is still space available for an extra record; if all the space for records is already occupied then the oldest record will be overwritten.

The data field can either be selected previously, using the command **Select File**, or you can use the short file identifier (SFI) given in parameter P2 to make this data field the current data field.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´E2´	Command code
P1	1	´00´	Fixed value
P2	1	´XX´	Type of addressing (+)
Lc	1	´01´...´FF´	Length of the command data
Data	Lc	´XX...YY´	Contents of the new record
Le	0	-	Empty

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	No errors and no warnings
		´63 CX´	Repeated write attempts were necessary, X indicates the number of attempts

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty

	Length	Contents	Description
Trailer	2	´64 00´	Referenced EF is deactivated, inconsistent or incorrect data
		´65 81´	Write error
		´67 00´	Response data has incorrect length, command data is missing, Lc is not equal to the length of the command data, or the data length is not equal to the fixed record length, Le is superfluous
		´69 81´	Referenced EF is not formatted or already contains the maximum number of records
		´69 82´	Security conditions not fulfilled
		´69 86´	Command not allowed, no EF is selected
		´69 88´	Incorrect data objects for secure messaging
		´6A 82´	File not found from SFI
		´6A 84´	Too much data for the EF
		´6A 86´	P1 or P2 has an illegal value
		´6E 00´	CLA and INS are inconsistent

Coding of P2

b8	b7	b6	b5	b4	b3	b2	b1	Description
								SFI usage:
0	0	0	0	0	0	0	0	References the current EF, no SFI
x	x	x	x	x	0	0	0	5-bit SFI

6.6 CHANGE REFERENCE DATA

The command **CHANGE REFERENCE DATA** launches a comparison between the data that accompanied the command and a reference value stored in the chip card. The parameter P2 specifies which reference value is to be used for the comparison.

If the comparison is acceptable, the *security status* of the chip card is changed and the old password replaced by the new password contained in the command data.

In the command data, the old password is immediately followed by the new one, without any kind of separator. The additional information relating to the password, which is referenced by the parameter P2, tells the chip card the transfer format and the length of the old password. This information is sufficient to enable it to explicitly extract the old password and the new one from the command data.

Suppose that a user wishes to replace his old password *ABC* by the new one *123*. If he inadvertently enters *ABCD* for the old password, then the data field for the command will contain *ABCD123*. Since the beginning of the data field does correctly correspond to the old password, the command **CHANGE REFERENCE DATA** will be executed and will mistakenly set up the new password as *D123*.
Since no separator is used, the chip card has no way of recognising this situation as an error unless both the old password and the new one are of maximum length.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´24´	Command code
P1	1	´00´	Fixed value
P2	1	´XX´	Check byte (+)
Lc	1	´01´...´FF´	Length of the command data
Data	Lc	´XX...YY´	Old password/new password, both in the same transfer format
Le	0	-	Empty

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	No errors and no warnings
		´63 CX´	As for ´90 00´, but problems with writing

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	Inconsistent or incorrect data
		´65 81´	Write error
		´66 12´	Parity error in the cryptographic algorithm
		´67 00´	Response data has incorrect length, Lc is missing or incorrect, Le is superfluous
		´69 82´	Security conditions not fulfilled
		´69 83´	Retry counter expired
		´69 85´	Referenced password cannot be used
		´69 88´	Incorrect data objects for secure messaging
		´6A 80´	Password has incorrect transport format or old and new passwords are identical
		´6A 83‘	Record not found
		´6A 86´	P1 or P2 possesses an illegal value
		´6A 88´	Referenced password not found
		´6E 00´	CLA and INS are inconsistent
		*	Trailers from the **VERIFY** command are also possible

Coding of P2

b8	b7	b6	b5	b4	b3	b2	b1	Description
x	...	...	...	...	...	...	...	Password:
0	...	...	...	...	...	...	...	Global, all information in the MF
1	...	...	...	...	...	...	...	Local, all information under the MF
...	0	0	0	0	0	...	...	Other values are RFU
...	...	...	...	...	...	x	x	Password number (PwdID)

6.7 Create File

The command **Create File** enables you to create a lower level file within the current directory. All the necessary information regarding the directory or data field that is to be created is contained in the command data.
In general it will not be possible to accommodate all the information within one command message, so for creating new directories you can execute the command in chaining mode.

If a successful subcommand is followed by a reset or some other command, then the command **Create File** will be aborted and the reset or the other interrupting command will be executed. Any changes already carried out by the command **Create File** will be retained.

The command data may contain a variety of data objects:

1. File header for a DF that is to be created
2. File header for an EF that is to be created
3. Record data, tag ´73´ with the following data elements:

Tag	Length	Value
´83´	´02´	FID
´85´	´XX...YY´	Contents of the records
´85´	´XX...YY´	Continuation of record contents, if necessary
´85´	´XX......	....

The above-mentioned data objects can be used as follows:

For a DF:	Data object 1 at the beginning, followed by <n> occurrences of data objects 2 and 3 in an arbitrary order
For an EF:	One data object 2

Command APDU

	Length	Contents	Description
CLA	1	 ´0X´ ´1X´	Command class (see page 150) Final subcommand Subcommand, not the final command
INS	1	´E0´	Command code
P1	1	´XX´	First byte of the file descriptor
P2	1	´00´	Fixed value
Lc	1	´01´...´FF´	Length of the command data
Data	Lc	´XX...YY´	Data objects 1, 2 and/or 3 (see above)
Le	0	-	Empty

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	Command was performed correctly
		´63 CX´	As for ´90 00´, but problems with writing

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	Current DF is deactivated, inconsistent or incorrect data
		´65 81´	Write error
		´67 00´	Response data has incorrect length, Lc is missing or incorrect, Le is superfluous
		´69 82´	Security conditions not fulfilled
		´69 88´	Incorrect data objects for secure messaging
		´6A 80´	Error in the content of the command data
		´6A 84´	Not enough memory available
		´6A 86´	P1 or P2 possesses an illegal value or P1 varies within a chain
		´6A 89´	FID already present in the current DF
		´6A 8A´	Name of the DF is already present in the chip card
		´6E 00´	CLA and INS are inconsistent or the subcommands have different low half-bytes
		*	Trailers from the **APPEND RECORD** command are also possible

6.8 DEACTIVATE FILE

The command **DEACTIVATE FILE** is the counterpart of the **ACTIVATE FILE** command. It reversibly places the current file in the **operational state - deactivated**. The only commands that can subsequently be performed on such a deactivated file are **SELECT FILE**, **ACTIVATE FILE** and **DELETE FILE**.

This command refers only to whichever file is currently selected. If, say, you wish to deactivate an entire application, you may find it appropriate to alternate this command with the command **SELECT FILE** for each of the files belonging to the application, one after another.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´04´	Command code
P1	1	´00´ ´02´	Deactivate current DF Deactivate current EF
P2	1	´00´	Fixed value
Lc	0	-	Empty
Data	0	-	Empty
Le	0	-	Empty

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	File has been deactivated
		´63 CX´	As for ´90 00´, but problems with writing

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	Inconsistent or incorrect data
		´65 81´	Write error
		´67 00´	Response data has incorrect length, data field is superfluous, Le is superfluous
		´69 82´	Security conditions are not fulfilled
		´69 86´	No EF is selected

	Length	Contents	Description
		´69 88´	Incorrect data objects for secure messaging
		´6A 86´	P1 or P2 possesses an illegal value
		´6E 00´	CLA and INS are inconsistent

6.9 Delete File

The command **Delete File** has the effect of removing the referenced file from the file system.

If the command refers to a directory, all the contents of the entire directory will be irreversibly lost, including all its subdirectories and all the data fields contained in those directories. The memory previously occupied by the deleted directories and data fields may be used for other purposes.

If the command refers to a data field, the contents of the data field will be irreversibly lost. The memory previously occupied by the deleted data field may be used for other purposes.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´E4´	Command code
P1	1	´00´ ´01´ ´02´	Delete the current DF The command data contains the FID of the DF that is to be deleted The command data contains the FID of the EF that is to be deleted
P2	1	´00´	Fixed value
Lc	0 1	- 2	Empty Length of the command data

	Length	Contents	Description

Data	0
	Lc
		-
		´XX...YY´
			P1=´00´: Empty
			P1=´01´: FID
			P1=´02´: FID

Le	0	-	Empty

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	File has been deleted
		´63 CX´	As for ´90 00´, but problems with writing

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	File header and its checksum not consistent, inconsistent or incorrect data
		´65 81´	Write error
		´67 00´	Response data has incorrect length, Lc is not equal to the length of the command data, Le is superfluous
		´69 81´	MF as current DF may not be deleted
		´69 82´	Security conditions not fulfilled
		´69 88´	File to be deleted not specified, incorrect data objects for secure messaging
		´6A 82´	Transferred FID not present in the current DF
		´6A 86´	P1 or P2 possesses an illegal value
		´6A 87´	Data field is missing or is superfluous
		´6E 00´	CLA and INS are inconsistent

6.10 External Authenticate, Mutual Authenticate

The command **External Authenticate** modifies the *security status* of the chip card, provided the encrypted data contained in the command matches the token that was previously requested from the external world.
The key that is used for this purpose is either specified by the parameter P2 or referenced via the active *security environment* of the current directory. Which cryptographic algorithm is to be used can be derived from the additional information for the referenced key.

It is absolutely essential, before issuing the command **External Authenticate**, to use the command **Get Challenge** to pass a random number to the external world.

In the variant **Mutual Authenticate**, the MICARDO token that has been processed by the external world is received together with a random number that is then processed by MICARDO and sent back to the external world. This makes it possible to perform mutual authentication with only one command.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´82´	Command code
P1	1	´00´	Fixed value
P2	1	´00´	Fixed value or (+)
Lc	1	´01´...´FF´	Length of the command data
Data	Lc	´XX...YY´	Command data (+)
Le	0 1	- ´00´...´FF´	Empty for one-sided authentication Length of the response data

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0 var.	- ´XX...YY´	Empty for one-sided authentication Mutual authentication
Trailer	2	´90 00´	Authentication was successful
		´63 CX´	As for ´90 00´, but problems with writing, X indicates the number of attempts
		´63 CX´	Authentication not successful, X indicates the retry counter for the key

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´65 81´	Write error
		´67 00´	Lc is missing or incorrect, Le is missing or superfluous, Le is smaller than Lr
		´69 00´	Missing key reference
		´69 82´	Security condition for K0.IFD not fulfilled
		´69 83´	Retry counter has expired
		´69 84´	Usage counter has expired
		´69 85´	Incorrect algorithm ID or second key reference is missing, incorrect command sequence, referenced key cannot be used
		´69 98´	Error encountered while deriving the session key
		´6A 86´	P1 or P2 possesses an illegal value
		´6A 88´	Referenced key not found
		´6E 00´	CLA and INS are inconsistent

Coding of P2

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	...	...	...	...	...	...	...	Global key, all information in the MF
1	...	...	...	...	...	...	...	Local key, all information under the MF
...	0	0	...	...	...	...	...	Other values are RFU
...	...	...	x	x	x	x	x	Key reference: KID || KV = ´FF´

The following table shows how the command data is to be laid out depending on the algorithm ID, i.e. depending on the security feature:

Procedure/Algorithm ID	Command data
Authentication, one-sided external, DES { 2 1 2 1 }	eK(RND.ICC) Length: 8 bytes
Authentication, mutual external first, DES { 2 2 1 2}	eK(RND.ICC) || RND.ICC Lengths: 8 bytes || 8 bytes
Key management, key transport, negotiation DES3, 1 SK, without SSC { 3 2 1 1 1 0 } DES3, 1 SK, with SSC { 3 2 1 1 3 0 }	eK(RND.IFD || RND.ICC || K.IFD) Lengths: 8 bytes || 8 bytes || 16 bytes
Key management, key transport, negotiation DES3, 2 SK, without SSC { 3 2 1 1 2 0 } DES3, 2 SK, with SSC { 3 2 1 1 4 0 }	eK(RND.IFD || RND.ICC || K.IFD) Lengths: 8 bytes || 8 bytes || 32 bytes

As decribed in section "Negotiating the Session Keys" on page 111, the lower 16 or 32 bytes are used to create the session key.

6.11 GENERATE PUBLIC KEY PAIR

The command **GENERATE PUBLIC KEY PAIR** generates the private and the public part of a RSA key pair.

First, all required EF_Key data fields have to be created. The TLV structure in the data fields has to be defined, too.

The private part of the RSA key is stored in the EF_Key_RSA_Private data field. The command expects that the FID is referenced in the EF_KeyD data field.

The public part of the RSA key is stored in the EF_Key_RSA_Public data field. The command expects that the necessary information for key generation is available.

The appendix contains a detailed description of an RSA key generation. The section "Example of Generation of a Key Pair" on page 245 lists all necessary steps and commands.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´46´	Command code
P1	1	´00´	Fixed value
P2	1	´00´	Fixed value
Lc	1	´04´	Length of the command data
Data	Lc	´83 02 XX...YY´	FID of EF_Key_RSA_Public
Le	0	-	Empty

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0	-	Empty

	Length	Contents	Description
Trailer	2	´90 00´	Retry counter is set to its initial value (from the command)

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´65 81´	EF_Key_RSA_Public is incorrect
		´67 00´	Lc is missing or incorrect, Le exists
		´69 82´	Security condition for key not fulfilled
		´69 85´	Key reference is missing or incorrect
		´6A 80´	Command data are wrong
		’6A 82’	EF_Key_RSA_Public is missing
		´6A 86´	P1 or P2 possesses an illegal value
		´6E 00´	CLA and INS are inconsistent

The following table desribes the command data:

Byte number	Command data
1	Value: ’83’
2	Value: ’02’
3-4	FID of the EF_Key_RSA_Public

6.12 GET CHALLENGE

The command **GET CHALLENGE** returns a certain number of bytes. Since the contents of these bytes are determined by an internal hardware random number generator, the byte sequence can be considered as a random number.

The number of bytes required is specified in the command.

This command is generally used to prepare chip card input data that is to be encrypted, or transferred with the protection of a cryptogram. The random number supplied by the chip card is then used to prepare the input data as IcvSecureMessaging.

Command APDU

	Length	Contents	Description
CLA	1	´00´	Command class (see page 150)
INS	1	´84´	Command code
P1	1	´00´	Fixed value
P2	1	´00´	Fixed value
Lc	0	-	Empty
Data	0	-	Empty
Le	1	´00´ '01...´FF´	Length of the response data: Lr = 8 bytes Lr = Le

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	var.	´XX...YY´	Random number
Trailer	2	´90 00´	No errors and no warnings

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´67 00´	Lc is superfluous, Le is missing
		´69 85´	Could not create random number
		´6A 86´	P1 or P2 possesses an illegal value
		´6E 00´	CLA and INS are inconsistent

6.13 GET RESPONSE

The command **GET RESPONSE** is used to send response data from the chip card to the external world.

The command is relevant only for the chip card protocol T=0.

Command APDU

	Length	Contents	Description
CLA	1	´00´	Command class (see page 150)
INS	1	´C0´	Command code
P1	1	´00´	Fixed value
P2	1	´00´	Fixed value
Lc	0	-	Empty
Data	0	-	Empty
Le	1	´00´...´FF´	Response data expected

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0 var.	- ´XX...YY´	Empty, no data available Response APDU or part of it
Trailer	2	´90 00´	No errors and no warnings
		´61 XX´	Additional response data in the chip card

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´67 00´	Lc is superfluous, Le is missing

	Length	Contents	Description
		´69 85´	There is no data to be fetched
		´6A 86´	P1 or P2 possesses an illegal value
		´6E 00´	CLA and INS are inconsistent
		’6C XY’	Send same command with Le = ’XY’

6.14 INTERNAL AUTHENTICATE

The command **INTERNAL AUTHENTICATE** has the effect of generating authentication data from a token passed to the chip card. The key that is used for this purpose is either specified by the parameter P2 or referenced via the active *security environment* of the current directory.

The additional information regarding the referenced key indicates how the response data is to be derived from the token.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´88´	Command code
P1	1	´00´	Fixed value
P2	1	´00´ ´XY‘	Fixed value Key reference (+)
Lc	0 1	- ´01´...´FF´	No command data Length of the command data
Data	0 Lc	- ´XX...YY´	Token is already present in the chip card Token that is to be encrypted or signed
Le	1	´00´...´FF´	Length of the response data

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	var.	´XX...YY´	Encrypted or signed token
Trailer	2	´90 00´	No errors and no warnings
		´63 CX´	Repeated write attempts were necessary

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´65 81´	Write error
		´67 00´	Lc is missing or incorrect, Le is missing, Le is smaller than Lr
		´69 00´	Missing key reference
		´69 82´	Security condition for K0.ICC not fulfilled
		´69 83´	Retry counter has expired
		´69 84´	Usage counter has expired
		´69 85´	Incorrect algorithm ID or second key reference is missing, incorrect key length, referenced key cannot be used
		´69 98´	Error encountered while deriving the session key
		´6A 86´	P1 or P2 possesses an illegal value
		´6A 88´	Referenced key not found
		´6E 00´	CLA and INS are inconsistent

Coding of P2

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	...	...	...	...	...	...	...	Global key, all information in the MF
1	...	...	...	...	...	...	...	Local key, all information under the MF

b8	b7	b6	b5	b4	b3	b2	b1	Description
...	0	0	...	...	...	...	...	Other values are RFU
...	...	...	x	x	x	x	x	Key reference: KID || KV = ´FF´

The following table shows how the command data is to be laid out depending on the algorithm ID, i.e. depending on the security feature:

Procedure/Algorithm ID	Command data
Authentication, one-sided internal, DES { 2 1 1 1 } internal, DES3 { 2 1 1 2 }	RND.IFD Length: 8 bytes
Authentication, one-sided internal, RSA { 2 1 1 4 }	RND.IFD Length: 1 byte <= length RND.IFD <= 40 % modulus length

6.15 Manage Security Environment

There are two variants of the command **Manage Security Environment**:

1. The SET variant, in which P1=´X1´
   This variant modifies components of the active *security environment* in the current directory. The parameters P1 and P2 specify which CRT components are affected. New values for the affected components are passed on in the command.
   This variant can also be used to pass data for key derivation and initiate such a key derivation process.
2. The RESTORE variant, in which P1=´F3´
   This variant replaces the active *security environment* in the current directory by one which is permanently stored in the chip card. The parameter

P2 specifies the number of this *security environment*, which comes into effect straightaway.

Command APDU

	Length	Contents	Description
CLA	1	´00´	Command class (see page 150)
INS	1	´22´	Command code
P1	1	´X1´ ´F3´	SET Manipulates the active SE in the current DF (+) RESTORE Replaces the active SE in the current DF (+)

P2	1	´XY´	P1=´X1´: tag of a CRT component
			P1=´F3´: SE#

Lc	1 0	´XY´ -	P1=´X1´: length of the command data P1=´F3´: empty

Data	Lc
	0
		´XX...YY´
		-
			P1=´X1´: command data
			P1=´F3´: empty

Le	0	-	Empty

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	No errors and no warnings

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´67 00´	Lc is not equal to the length of the command data, Lc is missing or is superfluous, Le is missing or is superfluous

	Length	Contents	Description
		´69 85´	Chip card contains no data from which to derive a key, referenced key cannot be used
		´6A 80´	Incorrect command data
		´6A 86´	P1 or P2 possesses an illegal value
		´6A 88´	Referenced key not found
		´6E 00´	CLA and INS are inconsistent

Coding of P1

b8	b7	b6	b5	b4	b3	b2	b1	Description
x	x	x	x	0	0	0	1	SET variant:
1	...	...	...	0	0	0	1	Corresponds to bit b8 of the usage qualifier
...	1	...	...	0	0	0	1	Corresponds to bit b7 of the usage qualifier
...	...	1	...	0	0	0	1	Corresponds to bit b6 of the usage qualifier
...	...	...	1	0	0	0	1	Corresponds to bit b5 of the usage qualifier
1	1	1	1	0	0	1	1	RESTORE
All other values RFU								not supported

You will find detailed information regarding usage qualifiers on page 16.

Coding of P2

P2	Significance
	RESTORE variant:
´XY´	Number of the new SE, ´XY´=´01´...´FE´ except ´EF´
	SET:

P2	Significance
´A4´	AT components in the data portion of the command
´B4´	CCT components in the data portion of the command
´B6´	DST components in the data portion of the command
´B8´	CT components in the data portion of the command
All other values RFU	

6.16 Perform Security Operation

The command **Perform Security Operation** (PSO) can be used to call a variety of cryptographic algorithms. The following algorithms are offered:

1. Computation of a digital signature
2. Hashing of data
3. Decryption of data

These functions are specified as PSO commands. They access various cryptographic algorithms and keys, so you may need to adjust the *security environment* before you execute a PSO command. This will usually be done by calling the command **Manage Security Environment**.

The desired PSO command is specified using parameters P1 and P2. The contained data may vary depending on which PSO command is used.

Command APDU

	Length	Contents	Description
CLA	1	 ´0X´ ´1X´	Command class (see page 150) Final subcommand Subcommand, not the final command
INS	1	´2A´	Command code for PSO command

	Length	Contents	Description
P1	1	´XX...YY´	Tag of the data object that is used in the response data (+)
P2	1	´XX...YY´	Tag of the data object compiled from Lc and Data (+)
Lc	0 1	- ´XX...YY´	Empty Length of the command data (Length)
Data	0 Lc	- ´XX...YY´	Empty Command data (Value)
Le	0 1	- ´00´...´FF´	Empty Length of the response data

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´6A 86´	Illegal combination of P1 and P2 for PSO commands

Permitted combinations of P1 and P2

P1 P2	PSO Function
´9E 9A´	**COMPUTE DIGITAL SIGNATURE**
´80 86´	**DECIPHER**
´90 80´	**HASH**, command data is not TLV coded
´90 A0´	**HASH**, command data is TLV coded

6.16.1 COMPUTE DIGITAL SIGNATURE

The PSO command **COMPUTE DIGITAL SIGNATURE** calculates a digital signature using an asymmetrical cryptographic algorithm and a referenced private key. The calculation is performed according to the algorithm ID specified by the key reference.

The desired key for the calculation must previously have been set up using the command **MANAGE SECURITY ENVIRONMENT**, unless it has already been defined. The referenced key is located in the relevant data field EF_Key.

You can select the cryptographic algorithm:

- Digital signature using RSA, DSI according to PKCS#1

MICARDO formats the transferred data according to the format that is currently set up and returns the calculated digital signature in its response.

The program supports only algorithm IDs that are defined for the calculation of digital signatures.

This command can use a command specific access condition.
In this case, the command expects in the SC data object only one further data object with the tag ' A4'. It could be of the type AUT or of the type PWD. This indicates that prior to each signature creation a key or password based authentication must be performed.

The security state is reset during command execution.

You can find general information of the command specific access condition in section "Access Condition of Type SPEC (Tag ´BA´)" on page 141.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´2A´	Command code for PSO command
P1	1	´9E´	Create digital signature
P2	1	´9A´	Hash value, digest info or message, not TLV coded

	Length	Contents	Description
Lc	0 1	- ´XX´	Hash value already in the chip card Length of the message in bytes
Data	0 ´XX´	- ´XX...YY´	Empty Hash value, digest info or message
Le	1	´00´...´FF´	Length of the response data

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	var.	´XX...YY´	Digital signature
Trailer	2	´90 00´	Command was executed correctly

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	Hardware fault during the operation, inconsistencies in EF_KeyD
		´64 00´	Inconsistent or incorrect data
		´65 81´	Error encountered while writing the retry counter
		´67 00´	Response data has incorrect length, Lc is not equal to the length of the command data, Le is smaller than Lr
		´69 82´	Access conditions for the key not fulfilled
		´69 83´	Retry counter is zero
		´69 84´	Usage counter is zero
		´69 85´	Usage conditions for the key not fulfilled, referenced key cannot be used
		´69 88´	DSI has incorrect length, incorrect data objects for secure messaging

	Length	Contents	Description
		´6A 88´	Key or hash function not found, referenced key not found

6.16.2 Decipher

The PSO command **Decipher** decrypts a cryptogram that is passed together with the command. The following cryptographic algorithms can be used:

– DES and DES3 decryption in CBC mode
– RSA decryption using the private key

The command **Manage Security Environment** is used to specify the key that is to be used and hence the cryptographic algorithm. The length of the response data is always a multiple of the length of the key. The first byte of the command data is the padding indicator which shows whether the chip card processes the plaintext data at the end of the operation, and if so, how. If a defined padding indicator is present, MICARDO will remove the padding bytes and output only the relevant plaintext data.

DES/DES3:
In the case of DES/DES3, the input has to be a multiple of 8 bytes plus padding indicator. The only defined padding indicator is ´01´ (ISO padding).
If the key requires an ICV, you must set up one using the command **Manage Security Environment**. After the command call the current block is output decrypted. The decryption uses a key from the relevant data field EF_Key. This key must be authorised for the decryption.

RSA:
In the case of RSA the chip card decrypts the transferred cryptogram (length = modulus) using a private key from the relevant data field EF_Key. This key must be authorised for the decryption. MICARDO assumes that the plaintext data of the cryptogram is formatted according to PKCS#1 (block type 01).

The byte ´00´ is placed before the code in the command call as the padding indicator (no further indication, PKCS#1 is assumed). MICARDO interprets the plaintext data sufficiently that only data which is identified as plaintext data is output. When the decryption has been carried out, MICARDO also checks whether the data has the correct PKCS#1 format. If this is not the case, then no response data will be output and the retry counter for the private key will be decreased.

The program supports only algorithm IDs that are defined for decryption.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´2A´	Command code for PSO command
P1	1	´80´	Plaintext data object, not TLV coded
P2	1	´86´	Padding indicator followed by a cryptogram
Lc	1	´XX´	Length of the command data in bytes
Data	Lc	´XX...YY´	Command data, not TLV coded
Le	1	´00´...´FF´	Length of the response data

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	8	´XX...YY´	Plaintext data
Trailer	2	´90 00´	Command was performed correctly

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	Hardware fault during the operation, inconsistencies in EF_KeyD, inconsistent or incorrect data
		´65 81´	Error encountered while writing retry counter or usage counter
		´67 00´	Response data has incorrect length, Lc is not equal to the length of the command data, Le is smaller than Lr

	Length	Contents	Description
		´69 82´	Access conditions for the key not fulfilled
		´69 83´	Retry counter is zero
		´69 84´	Usage counter is zero
		´69 85´	Usage conditions for the key not fulfilled
		´69 85´	Referenced key cannot be used
		´69 88´	Contents of the cryptogram are not correct
		´69 88´	Incorrect data objects for secure messaging
		´6A 88´	Referenced key not found

6.16.3 HASH

The PSO command **HASH** calculates a 160-bit hash value from the data transferred in the command. The calculated hash value is output in the response to the PSO command, in chaining mode it is output in the response to the last block.

The result is also stored temporarily in RAM, to be used in the event that the next command is **COMPUTE DIGITAL SIGNATURE** or **INTERNAL AUTHENTICATE**.

We distinguish two versions of this PSO command:

Hash performed entirely within the chip card:
The plaintext data passed in the command is formatted by the chip card as specified by the SHA conventions (padding). The command can be used in chaining mode; in this case a plaintext block must always have a length of 64 bytes. The final block may be smaller, in which case it will be padded by MICARDO.

Hash of previous round(s):
If the task of hashing is shared between MICARDO and the external world, the data field is laid out as follows:

1. Without chaining mode or one single block
   (data object with tag ´90´, followed by one data object with tag ´80´):
   – data object ´90´ with 20 bytes of a hash value block, followed by 8 bytes of counter (number of bits already hashed)

– data object ´80´ containing the unpadded text remaining to be hashed (of length up to 64 bytes). The padding will be carried out by MICARDO, if necessary.

2. Using chaining mode:
   – First command: data object ´90´ (28 bytes, laid out as for 1)
   – Second to (n-1)th command: data object ´80´ and 64 bytes of text
   – nth command: data object ´80´ containing the final unpadded text block (of length up to 64 bytes), the padding will be carried out by the chip card, if necessary.

Command APDU

	Length	Contents	Description
CLA	1	´00´ ´10´	Final command in chaining mode Chaining mode, additional command calls will follow
INS	1	´2A´	Command code for PSO command
P1	1	´90´	Create hash value
P2	1	´80´ ´A0´	Plaintext data object, not TLV coded TLV coded data objects in the data field
Lc	1	´XX´	Length of the message in bytes
Data	´XX´	´XX...YY´ ´XX...YY´ ´XX...YY´ ´XX...YY´ ´XX...YY´	Message data, not TLV coded, or ´90 14´ - ´ XX...YY´ hash value ´90 1C´ - hash block + 8-bytes counter || ´80´ - L - text to be hashed ´80 40´ - text to be hashed in chaining mode ´80´ - L - final portion of text to be hashed (nth command)
Le	1	´00´...´FF´	Length of the response data

Response APDU if the command was completed successfully, CLA = '00'

	Length	Contents	Description
Data	20	´XX...YY´	Hash value

	Length	Contents	Description
Trailer	2	´90 00´	Command was performed correctly

Response APDU if the command was completed successfully, CLA = '10'

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	Command was performed correctly

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´67 00´	Lc is not equal to the length of the command data, Le is not equal to 20 bytes, or Le is superfluous
		´69 87´	Counter or hash value missing
		´69 88´	Length of data object is not 28 or 64 bytes

6.17 READ BINARY

The command **READ BINARY** outputs all or part of the contents of a transparent data field.

The data field can either be selected previously, using the command **SELECT FILE**, or you can use the short file identifier (SFI) given in parameter P1 to make this data field the current data field.

In the command, the parameters P1 and P2 specify the first byte to be read relative to the beginning of the data field and Le supplies the number of bytes that are to be read.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see “Command Classes” on page 150)
INS	1	´B0´	Command code
P1	1	´XX´	SFI or MSB of the offset (+)
P2	1	´00´...´FF´	LSB of the offset
Lc	0	-	Empty
Data	0	-	Empty
Le	1	´00´...´FF´	Length of the response data

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	var.	´XX...YY´	Data requested from user data
Trailer	2	´90 00´	No errors and no warnings
		´62 82´	Length of the response data is less than Le
		´62 81´	User data and checksum are inconsistent

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	EF to be read out is deactivated, inconsistent or incorrect data
		´67 00´	Response data has incorrect length, Lc is superfluous, Le is missing
		´69 81	EF to be read out is not transparent
		´69 82´	Security conditions are not fulfilled
		´69 86´	Command not allowed, no EF is selected

	Length	Contents	Description
		´69 88´	Incorrect data objects for secure messaging
		´6A 82´	File not found via SFI
		´6A 86´	Parameters P1 - P2 incorrect
		´6B 00´	Offset is outside the EF
		´6E 00´	CLA and INS are inconsistent

Coding of P1

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	x	x	x	x	x	x	x	Offset (MSB)
1	0	0	x	x	x	x	x	5-bit SFI

6.18 READ RECORD

The command **READ RECORD** outputs the entire contents of a record within a linear data field.

The data field can either be selected previously, using the command **SELECT FILE**, or you can use the short file identifier (SFI) given in parameter P2 to make this data field the current data field.

In the command, the parameters P1 and P2 specify which record is to be read and Le supplies the length of the response data.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´B2´	Command code
P1	1	´01´...´FE´	Number of the record that is to be read
P2	1	´XX´	Type of addressing (+)
Lc	0	-	Empty

	Length	Contents	Description
Data	0	-	Empty
Le	1	´00´...´FF´	Length of the response data

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	var.	´XX...YY´	Contents of a record
Trailer	2	´90 00´	No errors and no warnings
		´62 82´	Length of the response data is smaller than Le
		´62 81´	Record and its checksum are inconsistent

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	EF to be read out is deactivated, inconsistent or incorrect data
		´67 00´	Response data has incorrect length, Lc is superfluous, Le is missing, Le is smaller than Lr
		´69 81	EF to be read out is not formatted
		´69 82´	Security conditions not fulfilled
		´69 86´	Command not allowed, no EF is selected
		´69 88´	Incorrect data objects for secure messaging
		´6A 82´	File not found via SFI
		´6A 83´	Referenced record does not exist
		´6A 86´	Parameters P1 - P2 incorrect
		´6E 00´	CLA and INS are inconsistent

Coding of P2

b8	b7	b6	b5	b4	b3	b2	b1	Description
								SFI usage:
0	0	0	0	0	...	...	...	References the current EF, no SFI
x	x	x	x	x	...	...	...	5-bit SFI
...	...	...	...	...	x	x	x	Type of addressing:
...	...	...	...	...	1	0	0	Absolute; record number is given in P1

6.19 Reset Retry Counter

The command **Reset Retry Counter** has the effect of resetting the retry counter for a password to its initial value. Which password the operation refers to is specified by the parameter P2.

It is also possible to replace the old password by a new one (P1=´00´).The replacment of the old password is authorized by a resetting code (password) which is integrated in the command.

⚠ An unauthorized reset of the retry counter should be prevented for reasons of security. As the execution of the command is determined ny access rules, e. g. a preceding successful authentication procedure (which may be password or key based) can be demanded. Alternatively the reset can depend on a resetting code.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´2C´	Command code
P1	1	´00´ ´03´	Command with replace Command without replace
P2	1	´XX´	Check byte (+)
Lc	1 0	´01´...´FF´ -	P1=´00´: Length of the command data P1=´03´: Empty

	Length	Contents	Description

Data	Lc
	0
		´XX...YY´
		-
			P1=´00´: Resetting code || new password
			P1=´03´: Empty

Le	0	-	Empty

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	Retry counter is now set to its initial value
		´63 CX´	As for ´90 00´, but problems with writing

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	Inconsistent or incorrect data
		´65 81´	Write error
		´66 12´	Parity error in the cryptographic algorithm
		´67 00´	Response data has incorrect length, command data is superfluous, Le is superfluous
		´69 82´	Security conditions are not fulfilled
		´69 85´	Retry counter is not zero, referenced password cannot be used
		´69 88´	Incorrect data objects for secure messaging
		´6A 80´	Incorrect transport format for the password
		´6A 83´	Record not found
		´6A 86´	P1 or P2 possesses an illegal value, no reference to the resetting code
		´6A 88´	Referenced password not found
		´6E 00´	CLA and INS are inconsistent

	Length	Contents	Description
		*	Trailers from the **Verify** and **Change Reference Data** commands are also possible

Coding of P2

b8	b7	b6	b5	b4	b3	b2	b1	Description
x	...	...	...	...	...	...	...	Password:
0	...	...	...	...	...	...	...	Global, all information in the MF
1	...	...	...	...	...	...	...	Local, all information under the MF
...	0	0	0	0	0	...	...	Other values are RFU
...	...	...	...	...	...	x	x	Password number (PwdID)

6.20 Search Binary

The command **Search Binary** performs a search in a transparent data field.

The command specifies the position within the data field from which the search is to begin, and which byte sequence (search pattern) is being sought. If this search pattern is present in the data field its position relative to the beginning of the user data is returned.

The data field can either be selected previously, using the command **Select File**, or you can use the short file identifier (SFI) given in parameter P1 to make this data field the current data field.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´A0´	Command code
P1	1	´XX´	SFI or MSB of the offset (+)
P2	1	´00´...´FF´	LSB of the offset
Lc	1	´01´...´FF´	Length of the command data

	Length	Contents	Description
Data	Lc	´XX...YY´	Search pattern that is being searched for in the referenced EF
Le	1	´00´,´02´	Length of the response data

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0 2	- ´XX...YY´	Unsuccessful search Offset at which the pattern begins
Trailer	2	´90 00´	Search pattern located
		´62 82´	Search was unsuccessful
		´62 81´	User data and its checksum are inconsistent

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	Data field is deactivated, inconsistent or incorrect data
		´67 00´	Response data has incorrect length, Lc is missing or incorrect, Le is missing or incorrect
		´69 81	Data field is not transparent
		´69 82´	Security conditions are not fulfilled
		´69 86´	Command not allowed, no EF is selected
		´69 88´	Incorrect data objects for secure messaging
		´6A 82´	File not found via SFI
		´6A 86´	Parameters P1 - P2 incorrect
		´6B 00´	Offset is outside the EF
		´6E 00´	CLA and INS are inconsistent

Coding of P1

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	x	x	x	x	x	x	x	Offset (MSB)
1	0	0	x	x	x	x	x	5-bit SFI

6.21 Search Record

The command **Search Record** performs a search in a data field which contains one or more records.

The search pattern is passed with the command. In the general case, the search begins from the beginning of the records. In extended search mode you can optionally specify an offset, so that the search of each individual record begins at the indicated position.

The data field can either be selected previously, using the command **Select File**, or you can use the short file identifier (SFI) given in parameter P2 to make this data field the current data field.

The response data contains the numbers of all the records in which the search was successful.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´A2´	Command code
P1	1	´01´...´FE´	Record number
P2	1	´XX´	Search strategy and (if used) SFI (+)
Lc	1	´01´...´FF´	Length of the command data
Data	Lc	´XX...YY´	Command data
Le	1	´00´...´FF´	Length of the response data

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0 var.	- ´XX...YY´	Unsuccessful search, Record numbers if search was successful
Trailer	2	´90 00´	Search was successful
		´62 82´	Search was unsuccessful
		´62 81´	Record and its checksum are inconsistent

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	The EF that is to be read out is deactivated, inconsistent or incorrect data
		´64 00´	
		´67 00´	Response data has incorrect length, Le is smaller than Lr, Lc is missing or incorrect, the command makes no specification for Le
		´69 81	EF to be read out is not formatted
		´69 82´	Security conditions are not fulfilled
		´69 86´	Command not allowed, no EF is selected
		´69 88´	Incorrect data objects for secure messaging
		´6A 80´	Command data is inconsistent with P2, control byte or offset byte is incorrect
		´6A 82´	File not found via SFI
		´6A 83´	Referenced record does not exist
		´6A 86´	P1 or P2 has an illegal value
		´6E 00´	CLA and INS are inconsistent

Coding of P2

b8	b7	b6	b5	b4	b3	b2	b1	Description
								SFI usage:
0	0	0	0	0	...	...	...	References the current EF, no SFI
x	x	x	x	x	...	...	...	5-bit SFI
...	...	...	...	...	0	x	x	Is not supported
...	...	...	...	...	1	x	x	Search strategy:
...	...	...	...	...	1	0	0	Forwards from record with number P1
...	...	...	...	...	1	0	1	Backwards from record with number P1
...	...	...	...	...	1	1	0	Extended search mode (see coding of the control byte)
...	...	...	...	...	1	1	1	Not supported

Coding of the control byte

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	0	0	0	...	...	...	...	Other values are RFU and are not supported
...	...	...	...	x	x	x	x	Extended search strategy:
...	...	...	...	0	...	...	...	Begins in each record at the absolute binary position given by the offset
...	...	...	...	1	...	...	...	Begins in each record after the character which is identical to the offset byte
...	...	...	...	...	1	0	0	Forwards from record with number P1
...	...	...	...	...	1	0	1	Backwards from record with number P1
...	...	...	...	...	All other values RFU			Are not supported

Example:
If bits b3b2b1 of P2 have the value = ´110´ then the structure of the command

data is as follows:
`data = control byte || offset byte || search pattern`.

6.22 SELECT FILE

You can use the command **SELECT FILE** to select a file in the file system and make it the current file. Depending on the parameters, this command selects either a directory (MF or DF) or a data field (EF).

When this command has been successfully executed the situation is as follows:

– If a directory was selected, the directory pointer points to this selected directory and the field pointer is undefined.
– if a data field was selected, the directory pointer remains unchanged and the field pointer points to this selected data field.

If the command cannot be completed successfully, both the directory pointer and the field pointer remain unchanged.
The security status values, the active *security environments* (including volatile ones) and all derived and negotiated keys will also remain unchanged if the command is unsuccessful.

Command APDU

	Length	Contents	Description
CLA	1	´00´	Command class (see page 150)
INS	1	´A4´	Command code
P1	1	´0X´	Type of navigation (+)
P2	1	´0X´	Quantity of information in the response data (+)
Lc	0 1	- ´01´...´FF´	Empty Length of the command data
Data	0 Lc	- ´XX...YY´	Empty Optional command data containing the FID or the name of the DF

	Length	Contents	Description
Le	0 1	- ´00´...´FF´	No response data Length of the response data

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0 var.	- ´XX...YY´	Empty Information about the file (+)
Trailer	2	´90 00´	No errors and no warnings
		´62 83´	File has been deactivated

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	File header and checksum are inconsistent,
		´67 00´	Lc is not equal to the length of the com-mand data, Le is missing or is superfluous, Le is smaller than Lr
		´6A 80´	P1=´00´ yet FID is not ´3F00´
		´6A 82´	File not found
		´6A 86´	P1 or P2 possesses an illegal value
		´6A 87´	Lc is missing or is superfluous
		´6E 00´	CLA and INS are inconsistent

Coding of P1

b8	b7	b6	b5	b4	b3	b2	b1	Significance
0	0	0	0	0	0	0	0	Select MF
...	...	...	...	...	x	x	x	Selection of DF or EF
...	...	...	...	...	0	0	0	Selection of MF

b8	b7	b6	b5	b4	b3	b2	b1	Significance
…	…	…	…	…	0	0	1	Select DF in the current DF
…	…	…	…	…	0	1	0	Select EF in the current DF
…	…	…	…	…	0	1	1	Select the DF at next higher level above the current one
…	…	…	…	…	1	0	0	Make selection using the name of the DF
All other values RFU								Are not supported

Coding of P2

b8	b7	b6	b5	b4	b3	b2	b1	Significance
0	0	0	0	x	x	0	0	Contents of the response:
0	0	0	0	0	0	0	0	Return data object consists of FCP and FMD
0	0	0	0	0	1	0	0	Return FCP
0	0	0	0	1	0	0	0	Return FMD
0	0	0	0	1	1	0	0	No response data
All other values RFU								Are not supported

6.23 Update Binary

The command **Update Binary** overwrites all or part of the contents of a transparent data field.

The data field can either be selected previously, using the command **Select File**, or you can use the short file identifier (SFI) given in parameter P1 to make this data field the current data field.

In the command, the parameters P1 and P2 specify the first byte to be overwritten, relative to the beginning of the data field, and pass the new data.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´D6´	Command code
P1	1	´XX´	SFI or MSB of the offset (+)
P2	1	´00´...´FF´	LSB of the offset
Lc	1	´01´...´FF´	Length of the command data
Data	Lc	´XX...YY´	New data
Le	0	-	Empty

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	No errors and no warnings
		´63 CX´	Repeated write attempts were necessary

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	Referenced EF is deactivated, user data and checksum are inconsistent, inconsistent or incorrect data
		´65 81´	Write error
		´67 00´	Response data has incorrect length, command data is missing or Lc is not equal to the length of the command data, Le is superfluous
		´69 81	Referenced EF is not transparent
		´69 82´	Security conditions are not fulfilled

	Length	Contents	Description
		´69 86´	Command not allowed, no EF is selected
		´69 88´	Incorrect data objects for secure messaging
		´6A 82´	File not found via SFI
		´6A 84´	Length of user data is less than offset + data length
		´6A 86´	Parameters P1 - P2 incorrect
		´6B 00´	Offset lies outside the EF
		´6E 00´	CLA and INS are inconsistent

Coding of P1

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	x	x	x	x	x	x	x	Offset (MSB)
1	0	0	x	x	x	x	x	5-bit SFI

6.24 Update Record

The command **Update Record** overwrites the entire contents of a record.

The data field can either be selected previously, using the command **Select File**, or you can use the short file identifier (SFI) given in parameter P2 to make this data field the current data field.

In the command, parameters P1 and P2 specify which record is to be overwrit-ten.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´DC´	Command code
P1	1	´01´...´FE´	Record number

	Length	Contents	Description
P2	1	´XX´	Type of addressing (+)
Lc	1	´01´...´FF´	Length of the command data
Data	Lc	´XX...YY´	New contents of the record that is to be overwritten
Le	0	-	Empty

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	No errors and no warnings
		´63 CX´	Repeated write attempts were necessary

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	Referenced EF is deactivated, inconsistent or incorrect data
		´65 81´	Write error
		´67 00´	Response data has incorrect length, command data is missing, or Lc is not equal to the length of the command data, or the data length is not equal to the fixed record length, Le is superfluous
		´69 81	Referenced EF is not formatted
		´69 82´	Security conditions are not fulfilled
		´69 86´	Command not allowed, no EF selected
		´69 88´	Incorrect data objects for secure messaging
		´6A 82´	File not found via SFI

	Length	Contents	Description
		´6A 83´	Referenced record does not exist
		´6A 84´	Too much data for the EF
		´6A 86´	Parameters P1 - P2 incorrect
		´6E 00´	CLA and INS are inconsistent

Coding of P2

b8	b7	b6	b5	b4	b3	b2	b1	Description
								SFI usage:
0	0	0	0	0	...	...	...	References current EF, no SFI
x	x	x	x	x	...	...	...	5-bit SFI
...	...	...	...	...	x	x	x	Type of addressing:
...	...	...	...	...	1	0	0	Absolute; record number is given in P1

6.25 Verify

The command **Verify** launches a comparison between data that is sent with the command and a reference value stored in the chip card.

The parameter P2 specifies which reference value is to be used for the comparison. If the comparison is acceptable, the *security status* of the chip card is changed.

Command APDU

	Length	Contents	Description
CLA	1	´0X´	Command class (see page 150)
INS	1	´20´	Command code
P1	1	´00´	Fixed value
P2	1	´XX´	Check byte (+)
Lc	1	´01´...´FF´	Length of the command data

	Length	Contents	Description
Data	Lc	´XX...YY´	Password in transfer format
Le	0	-	Empty

Response APDU if the command was completed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	Correct password
		´63 CX´	As for ´90 00´, but problems with writing
		´63 CX´	Incorrect password

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´64 00´	Inconsistent or data is incorrect
		´65 81´	Write error
		´66 12´	Parity error in the cryptographic algorithm
		´67 00´	Response data has incorrect length, Lc is missing or incorrect, Le is superfluous
		´69 82´	Security conditions are not fulfilled
		´69 83´	Retry counter has expired
		´69 84´	Usage counter has expired
		´69 85´	Referenced password cannot be used
		´69 88´	Incorrect data objects for secure messaging
		´6A 80´	Password has incorrect transport format
		´6A 83´	Record not found
		´6A 86´	P1 or P2 has an illegal value
		´6A 88´	Referenced password not found

	Length	Contents	Description
	2	´6E 00´	CLA and INS are inconsistent

Coding of P2

b8	b7	b6	b5	b4	b3	b2	b1	Description
x	...	...	...	...	...	...	...	Password:
0	...	...	...	...	...	...	...	Global, all information in the MF
1	...	...	...	...	...	...	...	Local, all information under the MF
...	0	0	0	0	0	...	...	Other values are RFU
...	...	...	...	...	...	x	x	Password number (PwdID)

Protocols

7.1 Basics

This chapter describes the protocols which the chip card uses to communicate with the external world.
We need to distinguish the following terms:

Transmission protocols
A transmission protocol specifies the rules which govern the communications for transferring data from the chip card to the external world, for example T=0.

Application protocols
An application protocol specifies the commands and responses for a particular application.

7.2 Answer to Reset

When the external world has performed a reset, the chip card responds by transmitting a byte sequence. This response is the answer to reset (ATR).

The chip card distinguishes between a cold reset and a warm reset:

Cold reset
A cold reset consists of switching on all supply lines as specified by the ISO, or an interruption of the power supply. After a cold reset MICARDO transmits a cold ATR.

Warm Reset
After a warm reset, which is a reset effected during normal operation via the reset line, MICARDO sends a warm ATR.

Every warm reset automatically leads to the output of a warm ATR and initiates the status **Specific Mode**.

7.3 ROM ATR

A ROM ATR is output only for uninitialised chip cards. Initialised cards output an EEPROM-specific cold ATR or warm ATR.

You can find information about the EEPROM-specific ATR in section ”EF_ATR” on page 81.

The ROM ATR is specified as follows:

TS	Initial character
T0	Format character
TA1, TB1, TC1...	Interface character
HB01, HB02...	Historical character
TCK	Check character

Character	Value	Remarks
TS	´3B´	Initial character, specifies direct convention
T0	´FD´	Format characters TA1, TB1, TC1 and TD1 follow, 13 historical byte
TA1	´94´	Frequency conversion factor FI = 1001 corresponds to Fi = 512; Baud rate adaptation factor DI = 0100 corresponds to Di = 8 at maximally 5 MHz
TB1	‘00’	No external Vpp
TC1	‘FF’	Extra guardtime from 12 etu for T=0 and 11 etu for T=1
TD1	´80´	TD2 follows, first protocol offered is T=0

Character	Value	Remarks
TD2	´B1´	TA3, TB3, TD3 follow with specific parameters for T=1
TA3	´FE´	Maximum length of the information field (information field size card) IFSC = 254
TB3	´45´	Character waiting time (CWT) = 5 Block waiting time (BWT) = 4 (default)
TD3	‘1F’	TA4 follows with global interface bytes
TA4	‘03’	No clock stop, Vcc = 3 V or Vcc = 5 V
HB01	‘00’	ISO ATR
HB02	‘68’	Preissuing data object of length 8
HB03...HB07	‘D2 76 00 00 28’	International RID (Registered Identifier) of ORGA
HB08	´FF´	Mask ID = SY-SCA
HB09	´05´	Chip manufacturer = Infineon
HB10	´1E´	Chip number, 0x1E = 30 decimal
HB11	‘00’	Card life status
HB12, HB13	´90 00´	Status bytes
TCK	´90´	Checksum according to the formula: $T0 \oplus TA1 \oplus \ldots \oplus TCK \equiv$ ´00´

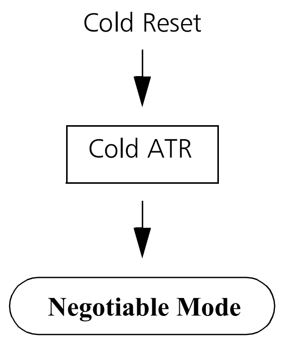
7.4 Protocol Parameter Selection

7.4.1 General Procedure

Following a cold reset, the chip card will transmit a cold ATR after between 400 and 40 000 clock pulses.
If no ATR has been output after 40 000 cycles the external world should switch off the power supply to the chip card.

In the cold ATR the chip card proposes its various parameters (Fi, Di and T) for the transfer. These parameters are standardly set to default values after every reset. The external world can use the procedure **Protocol Parameter Selection** (PPS) to modify these default parameters (Fd, Dd and T) to values which the chip card offers.

When the cold ATR has been transmitted, the chip card will automatically have the status **Negotiable Mode**.



The following four events may occur:

1. **PPS performed successfully**
   The PPS request sent from the external world is accepted and the chip card answers with a PPS response. The PPS is completed successfully. The chip card enters user-specific mode. The parameters the chip card offered in its cold ATR – frequency conversions factor Fi, Baud rate adaptation factor Di and transfer protocol T – have been accepted.
2. **PPS failed**
   The PPS failed. In this case the chip card does not send a PPS response, it waits for a reset.
3. **Warm ATR after cold ATR**
   A cold ATR is followed by a warm reset. The chip card outputs a warm ATR and enters the state **Specific Mode**. It is no longer possible to perform a PPS.
4. **No PPS**
   The external world does not perform a PPS. The chip card automatically enters default mode and sets up its default parameter values. The frequency conversions factor Fd, the baud rate adaptation factor Dd and T then have the following values:
   Fd = 372, Dd = 1 and T = 0

7.4.2 PPS Requests and PPS Responses

Only the external world can initiate a PPS and must do this immediately after a cold ATR.

PPS data is exchanged using the same parameters as were used to transfer the ATR.
Here: TS =´3B´, for direct convention, F=372 and D=1.

The PPS itself consists of a PPS request from the external world and, in a positive case, a PPS response from the chip card.

The PPS request which is sent by the external world is laid out as follows.

PPS char.	Value	Remarks
PPSS	´FF´	Initial character

PPS0	´XY´	b8 = 0
		b7 = 1, if PPS3 follows
		b6 = 1, if PPS2 follows
		b5 = 1, if PPS1 follows
		Y = requested protocol;
		only ´00´, ´01´, ´10´ or ´11´ will be accepted

PPS1	´XY´	Selection for the parameters F (=X) and D (=Y)
PPS2	´XY´	RFU, is never accepted by the chip card
PPS3	´XY´	RFU, is never accepted by the chip card
PCK	´XY´	Checksum according to the formula: PPSS ⊕ PPS0 ⊕ … ⊕ PCK ≡ ´00´

The chip card conforms to the following rules:

1. No PPS response is sent unless
   - PPS0 has one of the values ´00´, ´01´, ´10´, ´11´, and
   - the checksum is correct.
2. If PPS1 is not present, processing continues with Fd = 372 and Dd = 1.

3. If PPS1 is neither ´94´ (ATR proposal) nor ´11´ (default parameter), no PPS1 is included in the PPS response and processing continues with Fd = 372 and Dd =1.

If all the parameters are accepted, the PPS response from the chip card is identical to the PPS request.
If PPS1 is not accepted (rule 3) it is omitted from the PPS response.

In an error situation the chip card sends no PPS response.

The following table lists all possible PPS requests together with the corresponding reactions from the chip card.

PPS request						**PPS response**				**Active protocol after PPS response**
PPSS	**PPS0**	**PPS1**	**PPS2**	**PPS3**	**PCK**	**PPSS**	**PPS0**	**PPS1**	**PCK**	
´FF´	´00´	–	–	–	´FF´	´FF´	´00´	–	´FF´	T=0, F=372, D=1
	´01´	–	–	–	´FE´	´FF´	´01´	–	´FE´	T=1, F=372, D=1
	´10´	´11´	–	–	´FE´	´FF´	´10´	´11´	´FE´	T=0, F=372, D=1
		´94´	–	–	´7B´	´FF´	´10´	´94´	´7B´	T=0, F=512, D=8
		´XX´	–	–	´YY´	´FF´	´00´	–	´FF´	T=0, F=372, D=1
	´11´	´11´	–	–	´FF´	´FF´	´11´	´11´	´FF´	T=1, F=372, D=1
		´94´	–	–	´7A´	´FF´	´11´	´94´	´7A´	T=1, F=512, D=8
		´XX´	–	–	´YY´	´FF´	´01´	–	´FE´	T=1, F=372, D=1
all other values ...						no PPS response				Status: wait for reset

7.5 Transfer Protocols

The chip card supports a number of transfer protocols which can be used to send data to the card and receive data from it.

The basic protocol of the chip card is T=0. A protocol parameter selection (PPS) is implemented for the transfer protocols T=1 and T=0.

7.5.1 T=0

The implementation of the protocol T=0 is fully ISO-conform.
It uses the default value 10 (decimal) for the initial waiting time.

7.5.2 T=1

The implementation of the block transfer protocol T=1 is ISO-conform.

It supports the following protocol procedures:

- A waiting time extension request is sent within the specified block waiting time
- Chaining function at most up to the length of the input buffer
- IFS adjustment
- Resynchronisation

The chip card does not support the abort procedure or the status **Error on Vpp state** and treats them as protocol errors.

You will find a more detailed description of the individual procedures in the corresponding ISO standard.

Special features:

– MICARDO achieves compatibility with the EMV standard by mirroring the source address and destination address in the NAD byte, which leads to a formally correct response.
Layer 7 is provided with the NAD byte for control purposes, e.g. for application specific commands.
– MICARDO will neither send nor accept an i-block of length zero.
– As specified by ISO, MICARDO assumes that the maximum length of an information field has the initial value 254, which is confirmed by the external world.

7.6 Application Protocols

The chip card supports a layer 7 protocol as specified by ISO. Communications with the chip card always consist of a command from the external world and a response from the chip card. The card always has slave status, following a reset it waits for a command from the external world. After sending its response, the chip card goes back to executing the loop which waits for the next command.

Both the commands from the external world and the responses from the chip card are contained in APDUs (application protocol data units). The completed APDUs are then sent to the target addresses using the transfer protocols, e.g. APDU in the INF (information field) under T=1.

The layout of the command APDUs and response APDUs is described in detail in the section chapter "APDU Descriptions" on page 33.

7.6.1 Protocol Variant T=1

The protocol T=1 can transmit the APDUs completely and in a single step. This has the effect that the application protocol is entirely separated from the transfer protocol, but this is not critical and need not concern us further.

7.6.2 Protocol Variant T=0

Under T=0, the separation described for T=1 can be maintained to only a limited extent, because certain of the commands occur in more than one form, with or without data.

The following cases can arise:

Case	Command	Response
Case 1	No Data	No Data
Case 2	No Data	Data
Case 3	Data	No Data
Case 4	Data	Data

Under T=0 the basic communications procedure within an application for a command and the corresponding response look like this:

7.6.2.1 T=0, Case 1

The command APDU consists of a 4-byte header. The transfer protocol appends a fifth byte, P3 =´00´, and sends these 5 bytes to the chip card.

The chip card recognises from the command header that this is a case 1 command, and ignores P3.

The command is processed. The chip card then transmits the response APDU, which in this case consists only of a trailer.

7.6.2.2 T=0, Case 2

The command APDU consists of 5 bytes with P3 = Le, and is passed to the chip card in exactly that form. The chip card recognises from the command header that this is a case 2 command and interprets P3 as Le. The command is processed and the response APDU compiled.

1. If an error occurred while the command was being processed, the chip card returns only the trailer.
2. If Le equals the length of the response data, the chip card returns the response data.
3. If Le does not equal the length of the response data, the chip card returns the trailer ’61 XY’. ’XY’ is the length of the response data. The chip card then expects the command **GET RESPONSE**.

7.6.2.3 T=0, Case 3

The command APDU consists of more than 5 bytes. The first 5 bytes are sent to the chip card with P3 = Lc. The chip card recognises from the command header that this is a case 3 command and requests that the data be sent.

The command is processed, and the chip card then transmits the response APDU, which in this case consists only of a trailer.

7.6.2.4 T=0, Case 4

The command APDU consists of more than 5 bytes. The first 5 bytes are sent to the chip card with P3 = Lc. The chip card recognises from the command header that this is a case 4 command and requests that the data be sent.

This data does not, however, include the Le byte, which the chip card appends to the received command APDU with Le =´00´.

The command is processed and the response APDU is then compiled.

From now on for the response APDU the chip card reacts like described in case 2.

8 Initialisation and Personalisation

8.1 Introduction

A chip card for the MICARDO system has to perform the following routines before it can be used.

1. **Test routine**
   The test routine checks that the EEPROM is functioning correctly. It alternately writes and reads a particular area of the EEPROM.
2. **Initialisation routine**
   The initialisation routine creates the initial directory structure of the *file system* called minimal file system. This consists of the root directory MF and the data fields EF_Rule, EF_ATR.
   The minimal file system and the associated operating system elements, such as pointers, are referred to as the **standard image**.
   It is also possible to create an expanded customer specific file system.
3. **Personalisation routine**
   The personalisation routine can be subdivided into two processes.
   The first of these creates the application structures using the operating system command **CREATE FILE**.
   When this has been done, the personalisation procedure loads the personal data into the existing application using the operating system commands **UPDATE BINARY**, **APPEND RECORD** and **UPDATE RECORD**.

The operating system commands are described in detail in the appropriate chapter, beginning on page 147. The following diagram shows the special-purpose commands that are available for the various routines.

Testing	Tests prior to the initialisation - **WRITE EEPROM** - **COMPARE EEPROM**
Initialisation	- **VERIFY ROM PASSWORD** - **INIT EEPROM** Tests following the initialisation **- VERIFY EEPROM**
Personalisation	Operating system commands

The special commands belonging to these routines, plus an additional command for clearing the EEPROM, are always included in MICARDO's *command set*.

8.2 Overview of all the Commands

The following table lists all the commands, giving the code for the instruction byte (INS) in the command header and a brief explanation of the function.

Command	INS	Brief description
WRITE EEPROM	'B8'	Writes any desired value to one or more EEPROM pages
COMPARE EEPROM	'BA'	Compares one or more EEPROM pages with a specified value
VERIFY EEPROM	'B6'	Checks the integrity of the image

Command	INS	Brief description
INIT EEPROM	'02'	Creates data structures
VERIFY ROM PASSWORD	'20'	Verifies a supplied password by comparing it with the ROM password
DELETE EEPROM	'20'	Clears the EEPROM

8.3 Test Routines

8.3.1 Test Page

One specific page within the EEPROM area is a special case. This page is called the test page.

The following commands can be performed to the test page, even after the initialisation routine has been completed:

– **WRITE EEPROM**
– **COMPARE EEPROM**

This makes it possible to call these commands for test purposes. Such a call does not represent a breach of security, so no particular security conditions need to be fulfilled.

8.3.2 WRITE EEPROM

The command **WRITE EEPROM** is used to write a given value to EEPROM pages. This command always fills one or more complete pages with a byte value that is specified in the command, i.e. the same value is written to all the bytes of a page.

The absolute page numbers (first, last) are also specified in the command data, where '00 00' always addresses the first available page.

For all page numbers other than the test page, this command can be called only prior to the initialisation of the EEPROM or after it has been cleared.

Command APDU

	Length	Contents	Description
CLA	1	´D0´	Command class for personalisation commands
INS	1	´B8´	Command code
P1	1	´00´	Fixed value
P2	1	´00´	Fixed value
Lc	1	´05´	Fixed value
Data	Lc	´XX YY´ || ´XX YY´ || ´XX´	First page (2 bytes, high/low, start address), Last page (2 bytes, high/low, end address), Test byte that is to be written to all the cells in the specified pages
Le	0	-	Empty

Response APDU if the command was executed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	No errors and no warnings
		´63 CX´	Problems with writing

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´65 81´	Physical write error
		´67 00´	Lc is missing or incorrect, Le is superfluous
		´69 82´	Write not permitted
		´6A 80´	P1 or P2 is incorrect, incorrect start or end address
		´6E 00´	CLA is incorrect

8.3.3 Compare EEPROM

The command **Compare EEPROM** compares the contents of EEPROM pages with the one-byte value passed in the command. The command expects to find the same value in each byte of the pages that are being tested.

The absolute page numbers (first, last) are also specified in the command data, where '00 00' always addresses the first available page.

For all page numbers other than the test page, this command can be called only prior to the initialisation of the EEPROM or after it has been cleared.

Command APDU

	Length	Contents	Description
CLA	1	´D0´	Command class for personalisation commands
INS	1	´BA´	Command code
P1	1	´00´	Fixed value
P2	1	´00´	Fixed value
Lc	1	´05´	Fixed value
Data	Lc	´XX YY´ ‖ ´XX YY´ ‖ ´XX´	First page (2 bytes, high/low, start address), Last page (2 bytes, high/low, end address), Test byte that is checked for all the cells in the specified pages
Le	0	-	Empty

Response APDU if the command was executed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	Comparison was successful

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´67 00´	Lc is missing or incorrect, Le is superfluous
		´69 82´	Comparison not permitted
		´6A 80´	P1 or P2 is incorrect, data contains incorrect addresses
		´6E 00´	CLA is incorrect
		´6F 00´	Comparison failed

8.3.4 VERIFY EEPROM

The command **VERIFY EEPROM** is used to subsequently check the integrity of the image that was loaded into the EEPROM using the command **INIT EEPROM**. This makes it possible to detect any bit errors which arise during the interval between initialisation and personalisation.

The start and end addresses of that part of the EEPROM for which the CRC value should be computed is stored within the image. MICARDO calculates the CRC value over just this area and outputs it in the response APDU. The external world can then compare the supplied CRC value with a reference value in order to establish whether this EEPROM area is correct. The developer responsible for the image makes this reference value available to users (see section "Customer Information about the Initialisation" on page 244).

Command APDU

	Length	Contents	Description
CLA	1	´D0´	Command class for personalisation commands
INS	1	´B6´	Command code
P1	1	´00´	Fixed value
P2	1	´00´	Fixed value

	Length	Contents	Description
Lc		-	Empty
Data		-	Empty
Le	1	´00´, ´02´	Expected length of the CRC value

Response APDU if the command was executed successfully

	Length	Contents	Description
Data	0	‘XX...YY’	CRC value
Trailer	2	´90 00´	No errors and no warnings

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´67 00´	Lc is superfluous, Le is not ´00´ or ´02´
		´69 82´	Data access not permitted
		´6A 80´	P1 or P2 is incorrect
		´6E 00´	CLA is incorrect

8.4 Initialisation Routines

The term initialisation routine denotes a specific sequence of command calls each with its specific command data.

This routine is executed by an initialisation unit which sends a load data record to the chip card. This load data record contains the complete image and a MAC. The initialisation unit divides up the load data record into blocks (specific to the chip) and sends these to the chip card one at a time using the command **INIT EEPROM**.

The following points should be noted in relation to the command **INIT EEPROM**:

- The command transfers a certain (chip-specific) quantity of image data, indicated by P1='00', and writes this into the EEPROM beginning at the specified address.
  When writing is carried out page-wise, the initialisation process will be completed most efficiently if each call of the command transfers data for one or more whole pages.
- The RAM's maximum I/O buffer size (specific to the chip) determines the volume of data that can be transferred at any one time, so it will be necessary to call the command repeatedly, according to the actual quantity of data involved.
- The end of the initialisation routine is signalled to the command by setting P1='01'. In this case the command data does not contain any data for writing to the chip.
- Every image is secured by means of a MAC. When the final command has been received MICARDO compiles the MAC and checks the authenticity of the loaded image by comparing the MACs. The operating system is not ready for use until this process has been completed successfully.
- The command can be executed only so long as no correct initialisation has yet taken place. Either the EEPROM has previously been completely cleared using the command **DELETE EEPROM** or the chip has never been loaded.
  In either case, before an initialisation can be carried out in any session the ROM password must have been successfully checked using the command **VERIFY ROM PASSWORD**.
- You are prevented from calling the command again once there has been a successful initialisation. A renewed initialisation will not be possible until the command **DELETE EEPROM** has been executed.
  If the image has the command **DELETE EEPROM** disabled, then a renewed initialisation will not be possible **at all**. This is the setting for E4 evaluated MICARDO chip cards.

8.4.1 VERIFY ROM PASSWORD

The command **VERIFY ROM PASSWORD** checks the password given in the command data by comparing it with the ROM password.

The ROM password has a length of 8 bytes. It protects the chip card system from being inadvertently deleted or loaded with incorrect data.

Successful verification authorises the external world to call the following commands:

– **DELETE EEPROM**
– **INIT EEPROM**

The command is accompanied by 8 bytes of data. The chip card compares this data with the ROM data and sets an internal flag if the verification is successful. In the event of an error the execution is aborted and the flag reset.

⚠ The execution of the command is monitored by a non-volatile retry counter. If the password is entered incorrectly this counter is decremented by 1, if the entry is correct the counter is reset to its initial value.
If the password verification is not correct after three consecutive attempts then it will not be possible to call the command again, nor will the commands **DELETE EEPROM** or **INIT EEPROM** be available.

Command APDU

	Length	Contents	Description
CLA	1	´D0´	Command class for personalisation commands
INS	1	´20´	Command code
P1	1	´00´	Fixed value
P2	1	´11´	Fixed value
Lc	1	´08´	Length des password
Data	Lc	´XX…YY´	Password
Le	0	-	Empty

Response APDU if the command was executed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	Successful verification

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´65 81´	Write error in EEPROM
		´67 00´	Lc is missing or incorrect, Le is superfluous
		´69 83´	Retry counter has expired
		´6A 80´	P1 or P2 is incorrect
		´6E 00´	CLA is incorrect

8.4.2 INIT EEPROM

The command **INIT EEPROM** can be used to load variable data structures into the EEPROM. The contents of the data that is to be loaded is not checked and is therefore quite arbitrary.
This means that the command can be used to load operating system extensions, data fields, or entire applications.

This command can be executed only after the ROM password has been successfully verified.

Since the image data are DES3 encrypted, they always must have a length which can be devided by 8. Therefore the valid value of Lc is: Lc = 2+n*8.

Command APDU

	Length	Contents	Description
CLA	1	´D0´	Command class for personalisation commands

	Length	Contents	Description
INS	1	´02´	Command code
P1	1	´00´ ´01´	Write command data to EEPROM Validate image
P2	1	´00´	Fixed value
Lc	1 0	´XX´ -	P1=’00’: length of the command data P1=’01’: empty

Data	Lc	’XX…YY’	P1=’00’: MSB offset; LSB offset;
			Image data
			P1=’01’: empty

Le	0	-	Empty

Response APDU if the command was executed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	EEPROM written successfully
		´63 CX´	Problems with writing

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´65 81´	Write error in EEPROM
		´67 00´	Lc is missing, incorrect or superfluous, Le is superfluous
		´69 82´	Access condition not fulfilled
		´6A 80´	Parameter is P1 or P2 is incorrect, Lc is too small, EEPROM size less than offset + length of data
		´6E 00´	CLA is incorrect

8.4.3 Delete Eeprom

The command **Delete Eeprom** writes a defined (chip-specific) value throughout the whole of the EEPROM and thus deletes its previous contents.

This command can be executed only after the ROM password has been successfully verified.

Command APDU

	Length	Contents	Description
CLA	1	´D0´	Command class for personalisation commands
INS	1	´04´	Command code
P1	1	´00´	Fixed value
P2	1	´00´	Fixed value
Lc	0	-	Empty
Data	0	-	Empty
Le	0	-	Empty

Response APDU if the command was executed successfully

	Length	Contents	Description
Data	0	-	Empty
Trailer	2	´90 00´	EEPROM cleared successfully
		´63 CX´	As for ´90 00´, but problems with writing

Response APDU in the event of an error

	Length	Contents	Description
Data	0	-	No response data
Trailer	2	´65 81´	Write error in EEPROM
		´67 00´	Lc is superfluous, Le is superfluous
		´69 82´	Access condition not fulfilled

	Length	Contents	Description
		´6A 80´	Parameter P1 or P2 is incorrect
		´6E 00´	CLA is incorrect

8.4.4 Minimal File System

The minimal file system is created by the initialisation routine. The MF directory and the data fields EF_Rule and EF_ATR are created.

MF

The header of the directory MF is created with the following values:

File Control Parameter			
Tag	Length	Value	Comment
´82´	´01´	’38’	File description
´83´	´02´	’3F 00’	File identifier (FID)
´84´	’02’	’4D 46’	Name MF
´8A´	’01’	’05’	LCSI activated
´A1´	’03’	’8B 01 02’	Access Rule Reference Record number 2 in EF_Rule

File Management Data			
Tag	Length	Value	Comment
´85´	´08´	’D2 76 00 00 28 FF 05 1E’	Technical information: ORGA RID || SY-SCA mask ID || Infineon 30 chip
´85´	´03´	’00 00 01’	Image ID: Product Version Release

EF_Rule

The data field EF_Rule is created with the following values:

Data content: record 1			
Tag	**Length**	**Value**	**Comment**
´80´	´01´	’1F’	Activate, deactivate, append, update, read
’90’	’00’		Always

The record number 1 contains the access rules for the data fields EF_Rule and EF_ATR. Activate, deactivate, append, update and read and a change of LCSI are always allowed, all other accesses are not valid.

The other records have to be created and appended later by the MICARDO user. Especially the access rules for the MF in record number 2 have to be entered.

EF_ATR

This data field is described in detail in section ”EF_ATR” on page 81.

The described values are the ones that are entered when the EF_ATR data field is created as a part of the minimal file system.

8.4.5 Checking Initialised Cards

Identity

Cards, which are initialised with the standard image, can be identified using special identifiers.

The FMD of the MF contains the following identifiers:

– ORGA identifier
– Identifier of the semiconductor manufacturer
– Semiconductor identifier
– Identifier of the ROM mask
– Identifier of the standard image

– Version and release of the standard image

These identifiers are loaded during initialisation and are unchangeable. They can be read with the command **SELECT**.

If the standard image is extended with a customer specific part, the data field EF_TIN is created in the MF. The following identifiers are stored in the FMD there:

– Identifier of the customer
– Identifier of the customer specific part of the image (version and release)

The entries can not be changed afterwards and can be read with the command **SELECT**, too.

Integrity

The integrity of the image can be checked with the command **VERIFY EEPROM.**

You will find further information about this command in the section "VERIFY EEPROM" on page 218.

Authentication

The authentication of the cards can be checked with the command **INTERNAL AUTHENTICATE** with a dedicated authentication key pair.

The command is described in the section "INTERNAL AUTHENTICATE" on page 169. You will find information about the origin of the key pair in the section "Customer Information about the Initialisation" on page 244.

8.5 Personalisation Routines

8.5.1 Loading Applications

In this context we understand the loading of applications to mean the introduction of application structures, that is directories and data fields, using the operating system commands.

Once a MICARDO chip card has its master file set up it is possible to create any desired directories as the basis for applications. The command **CREATE FILE** is used for this purpose.
Within these application directories you can then create various data fields to accommodate the application data, keys and passwords. These data fields are also created using the command **CREATE FILE**.

The actual data contents are loaded into the corresponding data fields using the commands **UPDATE BINARY** and **UPDATE RECORD**. However, only fixed data values are generally loaded into the chip card, not personal data. It merely provides the supporting framework of applications, which are then rendered usable by personalising the appropriate data fields.

When creating application structures it is important to make sure that you provide for self-consistent assignment of security conditions for the new directories and data fields. The application vendor is responsible for ensuring that his applications are usable and personalisable. MICARDO does not check the contents of **any** data while performing the commands **CREATE FILE**, **UPDATE BINARY**, **APPEND RECORD** and **UPDATE RECORD**.

Since MICARDO provides for encrypted and MAC-secured transmission of data to the chip card, data can also be loaded on-line via unsecure computer networks.

8.5.2 Personalisation of Applications

An application that has been loaded into the chip card will not generally be able to function at a terminal until it has been provided with specific personal data, such as keys, passwords and user IDs.
We use the term personalisation to mean the loading of such data into a previously loaded application.

This personalisation can therefore not be performed unless the corresponding directories (DF) and data fields (EF) are already present on the chip card. The data, security parameters and features necessary for operating the application are then loaded into these data fields using the operating system commands **UPDATE BINARY**, **APPEND RECORD** and **UPDATE RECORD**.
These commands can be called only if the security conditions for the specific data field are fulfilled.

Personalisation can also be carried out on-line via unsecured computer networks, because MICARDO provides for encrypted and MAC-secured transmission of data to the chip card.

The processes of loading and personalising an application can be carried out separately or one after another, or can be performed as one combined action by a terminal unit.

The individual application specifications will give you further information about the loading and personalisation procedures for the applications as well as the structure of their directories and data fields.

Appendix

A.1 Abbreviations

AC	access condition
ACK	acknowledge
ADF	application dedicated file
AEF	application elementary file
ALW	always
APDU	application protocol data unit
ATR	answer to reset
AUT	authentication
BCD	binary coded digits
BWT	block waiting time
C	cryptogram
CBC	cipher block chaining mode
CC	cryptographic checksum
CCT	cryptographic checksum template
CFB	cipher feedback mode
CLA	class byte

CRC	cyclic redundancy check
CRT	control reference template or chinese remainder theorem
CT	cryptogram template
CWT	character waiting time
Dd	default baud rate adaptation
DE	data element
DER	distinguished encoding rules
DES	data encryption standard
DF	dedicated file
DI	baud rate adaptation shown in the ATR
DO	data object
DSI	digital signature input
EEPROM	electrically erasable programmable read only memory
EF	elementary file
ENC	encrypted
ETU	elementary time unit
Fd	default frequency conversion factor
FI	frequency conversion factor shown in the ATR
FID	file identifier
ICC	integrated circuit card
ICCSN	ICC serial number
ID	identifier
IFD	interface device
IFSC	information field size card
IFSD	information field size device

INS	instruction byte
ISO	international standards organization
K	key
LCSI	life cycle status integer
MAC	message authentication code
MF	master file
MSE	manage security environment
NEV	never
NIST	US National Institute of Standards and Technology
OID	object identifier
P1, P2	control parameters
PIN	personal identification number
PK	public key
PKCS	public key cryptography standards
PPS	protocol parameter selection
PRO	protected
PSO	perform security operation
RFU	reserved for future use
RND	random number
RSA	algorithm created by Rivest, Shamir and Adleman
SHA	secure hash algorithm
SIGN	signature
SK	session key
SM	secure messaging
SSC	send sequence counter

SW1	status value 1
SW2	status value 2
TLV	tag, length, value
Vpp	programming voltage input
WI	initial waiting time
WTX	waiting time extension

A.2 Glossary

Access Conditions

The access conditions govern who by which means may have what access to MICARDO’s directories, data, password or key.

Additional Information

Additional information is available for passwords and keys.
The additional information for passwords specifies, among other things, access rules and transfer formats.
The additional information for a key covers, for instance, the type of the key and how it may be used.

Algorithm ID

MICARDO uses the algorithm ID to recognise a → security feature.

Application

An application is a problem-oriented function complex for solving specific tasks. A chip card may contain a number of applications, e.g. home banking, videotex access and electronic purse.
Structurally, a MICARDO application consists of an application directory, the application definition file (ADF) and one or more → data fields, also known as application elementary files (AEF).

ATR

The ATR (answer to reset) is a byte sequence that is transmitted by the chip card following a reset.

Authentication Method

An instance uses an authentication method to prove to MICARDO that it is genuine. In the case of a cardholder authentication it is the password that is checked, in the case of a component authentication a → cryptographic key.

Card Number

The card number is the identification of the card as defined by the standard /ISO7812-1/.

Chaining Mode

A command may be equipped with a chaining mode which enables it to transmit large volumes of data. The data is divided up among a number of command calls and transmitted one piece at a time by executing the command repeatedly.

Chip Card Protocol

Chip card protocols are also called → transfer protocols.

Command APDU

A command APDU (application protocol data unit) is a data structure which contains the data pertaining to a command.

Command Structure

A command consists of a command header and optionally some command data. This structure is also known as the → command APDU.

Cryptographic Algorithm

The various methods used for encryption, decryption and the formation of → signatures are all referred to as cryptographic algorithms or crypto-algorithms.
We distinguish between symmetrical and asymmetrical cryptographic algorithms. Symmetrical methods use a single key, asymmetrical ones use a → key pair.

Cryptographic Keys

The term cryptographic keys denotes the keys that are used by → cryptographic algorithms.

Cryptographic Services

The term cryptographic services is used to cover all those services which use cryptographic algorithms.

Data Field Types

There are formatted data fields and unformatted ones. Formatted data fields are subdivided into → records, unformatted fields consist simply of a sequence of bytes.
The following types of data field are used:
- unformatted transparent data fields
- formatted fixed length records
- formatted variable length records
- formatted fixed length records arranged cyclically

Data Fields

MICARDO uses the term data field in two different contexts.

A data field is a structure in MICARDO’s file system. Data fields contain information, such as data, keys, passwords. They can be read and written by the → external world, provided the → access conditions permit.
Data fields are referred to as elementary files (EF). If the information they contain belongs to a particular → application they are called application elementary files (AEF).

That part of a → command APDU which contains the actual data is also termed a data field.

Directory

Directories are used to structure the data held on data media. A directory contains information which logically belongs together, and may itself contain other directories (subdirectories) forming a tree-like structure. MICARDO's directory structure begins with the root directory, which is called the master file (MF). The directories below this level, which are used to provide for additional structuring, are called dedicated files (DF). A directory which contains an → application is called an application dedicated file (ADF).

Directory Pointer

MICARDO maintains an internal directory pointer which indicates which is the current directory. This directory pointer is set afresh by commands such as **SELECT FILE**.

External World

External world is the general term used for the outgoing interface. It may be specified more precisely as appropriate, for instance as a card reader.

Field Pointer

MICARDO maintains an internal field pointer which points to the current data field. The field pointer is set afresh directly by commands such as **SELECT FILE** and is modified indirectly by commands such as **READ RECORD** when a short file identifier (SFI) is used to successfully make a selection.

File Layout

A file consists of a file header and a data body. The header contains the administrative data and the body contains the user data.

Identifier

An identifier is a sequence of two bytes used to provide a unique identification. All MICARDO’s directories and data fields possess such an identifier.

Image

The term image is used for the entire file system consisting of the root directory (MF), the initial directory structure, and the optional operating system areas in the initialisation context.

Initialisation

The initialisation loads the image into the chip card.

Key Management

The term key management is used to cover all the methods used for the creation, storage, addressing and destruction of cryptographic keys.

Key Pair

A key pair is made up of two parts, a public key and a private (secret) key.

Key Reference

The term key reference is used for a combination of the search type, key number, key version and FID which provide an unambiguous reference to the key.

Page

A certain number of memory cells in a storage area on a chip card is called a page.

Password

A password is a secret character sequence which is used to identify a user. A cardholder uses his password to prove that he is the rightful user of the card.

A password consists of 6 to 8 characters. The permitted characters include digits and letters as well as a number of special characters.

Personalisation

A chip card is personalised by creating the specific structures belonging to one or more → applications and filling them with personal data.

PIN

A PIN (personal identification number) is a secret sequence of digits which is used to identify a user. A cardholder uses his PIN to prove that he is the rightful user of the card.
A PIN consists of at least 4 and at most 12 digits.

Public Key Methods

The encryption and decryption procedures which use a → key pair are called public key methods.

Record

A record is the smallest accessible unit in a formatted data field.

Response APDU

A response APDU (application protocol data unit) is a data structure which contains the data belonging to a command response.
The chip card transmits such a response when it is prompted to do so by the command contained in a → command APDU.

Secure Messaging

The term secure messaging is used for data transfers which are protected from manipulation or tapping.

Security Environment

A security environment comprises the prespecified → security features together with the → key references that are used for specific security commands and for → secure messaging.

Security Feature

Keys are assigned certain security features which specify the purposes for which a key may be used.
MICARDO can enable a key for use with a → cryptographic algorithm, an → authentication method or for → key management.

Signature

Here the term is used for an electronic signature which is calculated using a cryptographic algorithm.

Transmission Protocols

A transmission protocol specifies the rules which govern the communications which transfer data between the chip card and the → external world.

A.3 Trailer/Return Codes

Description in alphabetical order:

Description	Code
AuthenticationBlockedError	'69 83'
AuthenticationFailedWarning	'63 CX'
ClassNotSupportedError	'6E 00'
CommandExecutionOrderError	'69 85'
CommandNotAllowedError	'69 86'
ConditionOfUseNotSatisfiedError	'69 85'
CorruptDataError	'64 00'
CorruptDataWarning	'62 81'
DfNameExistError	'6A 8A'
EndOfFileWarning	'62 82'

Description	Code
EndOfRecordWarning	’62 82’
ExecutionError	’64 00’
FileExistError	’6A 89’
FileInvalidError	’64 00’
FileInvalidWarning	’62 83’
FileNotFoundError	’6A 82’
IncompatibleFileStructureError	’69 81’
InconsistentDataError	’64 00’
IncorrectParameterError	’6A 86’
IncorrectParametersDatafieldError	’6A 80’
InstructionCodeNotSupportedError	’6D 00’
LcInkonsistentWithP1P2Error	’6A 87’
MemoryFailureError	’65 81’
NoChvReferenceError	’69 00’
NoError	’90 00’
NoKeyreferenceError	’69 00’
NoPreciseDiagnosisError	’6F 00’
NotEnoughMemorySpaceError	’6A 84’
RecordNotFoundError	’6A 83’
ReferencedDataInvalidatedError	’69 84’
ReferencedDataNotFoundError	’6A 88’
ResponseDataAvailable	’61 XX’
SecurityStatusNotSatisfiedError	’69 82’

Description	Code
SmDataObjectsIncorrectError	’69 88’
SmDataObjectsMissingError	’69 87’
SmWithoutSessionkeysError	’69 89’
TinyLeError	’67 00’
UndefinedTechnicalProblemError	’64 00’
UpdateRetryWarning	’63 CX’
WrongLeError	’6C XY’
WrongLengthError	’67 00’
WrongParametersError	’6B 00’

Numerical order:

Code	Description
’61 XX’	ResponseDataAvailable
’62 81’	CorruptDataWarning
’62 82’	EndOfFileWarning
’62 82’	EndOfRecordWarning
’62 83’	FileInvalidWarning
’63 CX’	AuthenticationFailedWarning
’63 CX’	UpdateRetryWarning
’64 00’	CorruptDataError
’64 00’	ExecutionError
’64 00’	FileInvalidError
’64 00’	InconsistentDataError

Code	Description
'64 00'	UndefinedTechnicalProblemError
'65 81'	MemoryFailureError
'67 00'	TinyLeError
'67 00'	WrongLengthError
'69 00'	NoChvReferenceError
'69 00'	NoKeyreferenceError
'69 81'	IncompatibleFileStructureError
'69 82'	SecurityStatusNotSatisfiedError
'69 83'	AuthenticationBlockedError
'69 84'	ReferencedDataInvalidatedError
'69 85'	CommandExecutionOrderError
'69 85'	ConditionOfUseNotSatisfiedError
'69 86'	CommandNotAllowedError
'69 87'	SmDataObjectsMissingError
'69 88'	SmDataObjectsIncorrectError
'69 89'	SmWithoutSessionkeysError
'6A 80'	IncorrectParametersDatafieldError
'6A 82'	FileNotFoundError
'6A 83'	RecordNotFoundError
'6A 84'	NotEnoughMemorySpaceError
'6A 86'	IncorrectParameterError
'6A 87'	LcInkonsistentWithP1P2Error
'6A 88'	ReferencedDataNotFoundError

Code	Description
’6A 89’	FileExistError
’6A 8A’	DfNameExistError
’6B 00’	WrongParametersError
’6C XY’	WrongLeError
’6D 00’	InstructionCodeNotSupportedError
’6E 00’	ClassNotSupportedError
’6F 00’	NoPreciseDiagnosisError
’90 00’	NoError

A.4 Customer Information about the Initialisation

Each customer receives together with the ordered MICARDO chip cards an initialisation specification and, if a personalisation was performed too, a personalisation specification.

The initialisiation specification contains among other things the following information:

– FMD of the MF
– FMD of EF_TIN (if a customer specific image has been loaded)
– Public part of the authentication key pair
– Description of the data field for the private part of the authentication key pair: EF_Key_RSA_Private (description without key code)

The public part of the authentication key pair and the CRC of the image for the integrity check of the image are delivered in an authentic way.

A.5 Example of Generation of a Key Pair

The command **Generate Public Key Pair** transmits only few pieces of information regarding the key. The other necessary information has to be stored previously in the chip card.

The following sections describe the sequence of commands to provide all information necessary to generate a key pair. These are:

– Create a new directory with data fields
– Fill new data fields
– Change access rules
– Generate keys
– Read public keys

The description starts with the selection of a directory in which the creation of a new directroy is allowed without secure messaging.

A.5.1 Create a New Directory with Data Fields

The following section describes the creation of a new directory (DF) with all necessary files. The records of the data files (EFs) are filled with the appropriate structure information.

A.5.1.1 Create GPK Directory

```
Command APDU = 10e03800 18
62148201388302df00840347504b8a0105a1038b01006400
```

```
Response APDU = 9000
```

Remark for Command APDU:
```
62 14           // FCP
    82 01 38     //   DF
    83 02 df00   //   FID
    84 03 47504b //   AID  = GPK
    8a 01 05     //   LCSI = activated
    a1 03        //   ID ARR data object
       8b 01 00  //     Record 0
  6400           // FMD
```

A.5.1.2 Create EF_Rule

```
Command APDU = 10e03800 38
621782050441000e028302003085020013 8a0105a1038b010164
00731b83020030850e800103800190840146 84012a9000850580
01029000
```

```
Response APDU = 9000
```

Remark for Command APDU:

```
62 17                // FCP
   82 05 0441000e02 //   File descriptor: linear, var.
    83 02 0030       //   FID
    85 02 0013       //   File size
    8a 01 05         //   LCSI = activated
    a1 03            //   ID ARR data object
      8b 01 01       //     Use record 1 in EF_Rule
  64 00              // FMD
  73 1b              // Record data
    83 02 0030       //   EF_Rule
    85 0e            //   Record 01:
      80 01 03       //     Update read \
      80 01 90       //     Internal Authenticate \
      84 01 46       //     Generate Public Key Pair \
      84 01 2a       //     PSO \
      90 00          //     Always
    85 05            //   Record 02:
      80 01 02       //     Update \
      90 00          //     Always
```

A.5.1.3 Create EF_Key_RSA_Private for Authentication Key

```
Command APDU = 10e03800 17
62138201018302100185020 15a8a0105a1038b01026400
```

```
Response APDU = 9000
```

Remark for Command APDU:

```
  62 13         // FCP
    82 01 01    //   File descriptor: transparent
```

```
      83 02 1001 //   FID
      85 02 015a //   File size
      8a 01 05   //   LCSI = activated
      a1 03      //   ID ARR data object
        8b 01 02 //     Use record 2 in EF_Rule
    64 00        // FMD
```

A.5.1.4 Create EF_Key_RSA_Public for Authentication Key

```
Command APDU = 10e03800 17
62138201018302c0018502011e8a0105a1038b01016400
```

```
Response APDU = 9000
```

Remark for Command APDU:

```
  62 13          // FCP
    82 01 01     //   File descriptor: transparent
    83 02 c001   //   FID
    85 02 011e   //   File size
    8a 01 05     //   LCSI = activated
    a1 03        //   ID ARR data object
      8b 01 01   //     Use record 1 in EF_Rule
  64 00          // FMD
```

A.5.1.5 Create EF_Key_RSA_Private for Signature Key

```
Command APDU = 10e03800 17
62138201018302100285020115a8a0105a1038b01026400
```

```
Response APDU = 9000
```

Remark for Command APDU:

```
  62 13         // FCP
    82 01 01    //   File descriptor: transparent
    83 02 1002 //   FID
    85 02 015a //   File size
    8a 01 05    //   LCSI = activated
    a1 03       //   ID ARR data object
      8b 01 02 //     Use record 2 in EF_Rule
  64 00         // FMD
```

A.5.1.6 Create EF_Key_RSA_Public for Signature Key

```
Command APDU = 10e03800 17
62138201018302c002850201208a0105a1038b01016400
```

```
Response APDU = 9000
```

Remark for Command APDU:
```
  62 13         // FCP
    82 01 01    //   File descriptor: transparent
    83 02 c002 //   FID
    85 02 0120 //   File size
    8a 01 05    //   LCSI = activated
    a1 03       //   ID ARR data object
      8b 01 01 //     Use record 1 in EF_Rule
  64 00         // FMD
```

A.5.1.7 Create EF_KeyD

```
Command APDU = 00e03800 5b
621382050241001d02830200138a0105a1038b01016400
73 42 83020013
851d830401001001c00281807b11800100a1038b0101e4079501
4089022113851d830402001002c00281807b11800100a1038b01
01f60795014089021310
```

```
Response APDU = 9000
```

Remark for Command APDU:
```
  62 13               // FCP
    82 05 0241001d02 //   File descriptor: linear, fix
    83 02 0013       //   FID
    8a 01 05          //   LCSI = activated
    a1 03             //   ID ARR data object
      8b 01 01       //     Use record 1 in EF_Rule
  64 00               // FMD
  73 42               // Record data
    83 02 0013       //   EF_KeyD
    85 1d             //   Record 01:
      83 04           // Key identifier
      0100          //   KID | KV
      1001          //   FID
    c0 02 8180      // RSA 1024 bit
    7b 11           // SE data object
      80 01 00      //   All SE
      a1 03         //   ID ARR data object
        8b 01 01    //     Record 01
      e4 07         //   AT
        95 01 40    //     Internal Authenticate
        89 02 2113 //     Algorithm ID = AutIntRsa
  85 1d             //   Record 02:
    83 04           // Key identifier
      0200          //   KID | KV
      1002          //   FID
    c0 02 8180      // RSA 1024 bit
    7b 11           // SE data object
      80 01 00      //   All SE
      a1 03         //   ID ARR data object
        8b 01 01    //     Record 01
      f6 07         //   DST
        95 01 40    //     Compute Digital Signature
        89 02 1310 //     Algorithm ID = KrySigRsa
```

A.5.2 Fill New Data Fields

The following section describes the selection of the previously created directory (DF) and contains the information which is written into the transparent data fields (EFs).

A.5.2.1 Select GPK Directory

```
Command APDU = 00a4040c0347504b
```

```
Response APDU = 9000
```

A.5.2.2 Select EF_Key_RSA_Private for Authentication Key

```
Command APDU = 00a4020c021001
```

```
Response APDU = 9000
```

A.5.2.3 Update EF_Key_RSA_Public for Authentication Key 1/2

```
Command APDU = 00d60000 d0
81 44 00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000 00000000
82 42 00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000 0000
83 44 00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000 00000000
```

```
Response APDU = 9000
```

Remark for Command APDU:
`Refer to section ”EF_Key_RSA_Private” on page 66`

A.5.2.4 Update EF_Key_RSA_Private for Authentication Key 2/2

```
Command APDU = 00d600d0 8a
84 42 00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000 0000
85 44 00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000 00000000
```

```
Response APDU = 9000
```

Remark for Command APDU:
`section ”EF_Key_RSA_Private” on page 66`

A.5.2.5 Select EF_Key_RSA_Public for Authentication Key

```
Command APDU = 00a4020c02c001

Response APDU = 9000
```

A.5.2.6 Update EF_Key_RSA_Public for Authentication Key 1/2

```
Command APDU = 00d60000 80
b682011a8304012345677f49818a80010281818000000000000000
000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000

Response APDU = 9000
```

Remark for Command APDU:

```
Refer to section ”EF_Key_RSA_Public” on page 68, here

Register number          = 01234567
Modulus n length         = 1024 bit
Public exponent e length = 16 bit
```

A.5.2.7 Update EF_Key_RSA_Public for Authentication Key 2/2

```
Command APDU = 00d60080 9e
000000000000000000000000000000000000000082020000800111
9e8180000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000

Response APDU = 9000
```

Remark for Command APDU:

```
Refer to section ”EF_Key_RSA_Public” on page 68
```

A.5.2.8 Select EF_Key_RSA_Private for Signature Key

```
Command APDU = 00a4020c021002

Response APDU = 9000
```

A.5.2.9 Update EF_Key_RSA_Private for Signature Key 1/2

```
Command APDU = 00d60000 d0
81 44 00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000 00000000
82 42 00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000 0000
83 44 00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000 00000000

Response APDU = 9000
```

Remark for Command APDU:
`Refer to section ”EF_Key_RSA_Private” on page 66`

A.5.2.10 Update EF_Key_RSA_Private for Signature Key 2/2

```
Command APDU = 00d600d0 8a
84 42 00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000 0000
85 44 00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000
      00000000000000000000000000000000 00000000

Response APDU = 9000
```

Remark for Command APDU:
```
Refer to section ”EF_Key_RSA_Private” on page 66
```

A.5.2.11 Select EF_Key_RSA_Public for Signature Key

```
Command APDU = 00a4020c02c002

Response APDU = 9000
```

A.5.2.12 Update EF_Key_RSA_Public for Signature Key 1/2

```
Command APDU = 00d60000 80
b682011c8304765432107f49818c80010281818000000000000000
000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000

Response APDU = 9000
```

Remark for Command APDU:
```
Refer to section ”EF_Key_RSA_Public” on page 68, here

Register number          = 76543210
Modulus n length         = 1024 bit
Public exponent e length = 32 bit
```

A.5.2.13 Update EF_Key_RSA_Public for Signature Key 2/2

```
Command APDU = 00d60080 a0
000000000000000000000000000000000000000082040000000080
01119e818000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000

Response APDU = 9000
```

1 2 3 4 5 6 7 8 A I

A.5.3 Change Access Rules

The following section describes the changes of the access rules in the data field EF_Rule. After this, the access to private key information is not possible.

A.5.3.1 Select EF_Rule

```
Command APDU = 00a4020c020030
```

```
Response APDU = 9000
```

A.5.3.2 Update EF_Rule Record 2

```
Command APDU = 00dc0204058001009700
```

```
Response APDU = 9000
```

Remark for Command APDU:
`If this step is not performed, private keys can be changed!`

```
  80 01 00 // Nothing \
  97 00    // Never
```

A.5.3.3 Update EF_Rule Record 1

```
Command APDU = 00dc01040e8001018001908401468401 2a9000
```

```
Response APDU = 9000
```

Remark for Command APDU:
`If this step is not performed, access rules and private keys can be read.`

```
  80 01 01 // Read \
  80 01 90 // Internal Authenticate \
  84 01 46 // Generate Public Key Pair \
  84 01 2a // PSO \
  90 00    // Always
```

A.5.4 Generate Keys

A.5.4.1 Manage Security Environment: SET for AT

```
Command APDU = 002241a4058303800100
```

```
Response APDU = 9000
```

Remark for Command APDU:
The key reference for authentication is set (AT), all other CRT in this SE are empty.

A.5.4.2 Generate Public Authentication Key Pair

```
Command APDU = 00460000048302c001
```

```
Response APDU = 9000
```

Remark for Command APDU:
MICARDO first searches for a key reference in CT, this CRT is empty, therefore MICARDO continues the search for a key reference in DST, this CRT is also empty, last MICARDO searches for a key reference in AT.

Because of the previous MSE command a key reference exists. Thus this key is affected by the command Generate Public Key Pair. A public key pair is generated, the public key is written to the file with the FID = C0 01 and signed by the key referenced in AT (here the key signs itself).

A.5.4.3 Manage Security Environment: SET for DST

```
Command APDU = 002241b6058303800200
```

```
Response APDU = 9000
```

Remark for Command APDU:
The key reference for signing is set (DST), the key reference for AT was given in a previous MSE command, all other CRT in this SE are empty.

A.5.4.4 Generate Public Signature Key Pair

```
Command APDU = 00460000048302c002
```

```
Response APDU = 9000
```

Remark for Command APDU:
MICARDO first searches for a key reference in CT, this CRT is empty, therefore MICARDO continues the search for a key reference in DST.

Because of the previous MSE command a key reference exists. Thus this key is affected by the command Generate Public Key Pair. A public key pair is generated, the public key is written to the file with the FID = C002 and signed by the key referenced in AT (here the key is signed by the key genereted by the first GeneratePublicKeyPair–command).

A.5.5 Read Public Keys

A.5.5.1 Select EF_Key_RSA_Public for Authentication Key

```
Command APDU = 00a4020c02c001
```

```
Response APDU = 9000
```

A.5.5.2 Read EF_Key_RSA_Public for Authentication Key 1/2

```
Command APDU = 00b0000080
```

```
Response APDU = 9000
b682011a8304012345677f49818a80010281818096d929cfab5a
0d5ef5c2158e6aa8244f75ffc32234bcba5cc144ea671c327290
6a1f4505c5cae704219512f8467662f03cd5b2ec3d9080488979
41b11fbdac9a885aa5d4b9ba254116317f354e149915adb0388d
31ae11575bf66ab089031bddabbddaa07b19340f6ab1e4d2
```

A.5.5.3 Read EF_Key_RSA_Public for Authentication Key 2/2

```
Command APDU = 00b0008000
```

```
Response APDU = 9000
47e6fc9c1a9dc3f762c85f395d29b460fa884c2b8202ca6f8001
119e81805cde0b496c2735074ac55a4bc8539f5be520e5af62c8
852136a85a5688b7217ba975eb1bb8727aef2531b7669261a914
e77988922cafe51ad709e84e19c906919a69d46381fbffc551a9
9b909ef814a414b284a987f63fa1e8845940be66388107c056f9
31a921cfe298ff99a07cfca11ccc298f4b38ba61f5d3429422e2
a7fd
```

A.5.5.4 Select EF_Key_RSA_Public for Signature Key

```
Command APDU = 00a4020c02c002
```

```
Response APDU = 9000
```

A.5.5.5 Read EF_Key_RSA_Public for Signature Key 1/2

```
Command APDU = 00b0000080
```

```
Response APDU = 9000
b682011c8304765432107f49818c800102818180db982ed3926a
d22ccb7e75932bbf92795aae6ecad5b846c36f6d0ba8d56e5c4a
01312c4132f62e86ea82706915274765b881a49cd2394f19ea28
fe44701e7b65578557b3333e3e3989f46cf35263b8374f04f90f
63cf969b503392e7cc77a952800c2577cdc8e359bcb5da02
```

A.5.5.6 Read EF_Key_RSA_Public for Signature Key 2/2

```
Command APDU = 00b0000080
```

```
Response APDU = 9000
41a9cade380503a5d9d8bd3d4756b9b5a7ccb8fb8204ce953053
8001119e8180023fab145875e0a4247f206f1b278b437af75d31
f2b49949c2fba0cbf6a9e8ae73b0245ff183b98785d42e079f28
5768eaa0cfffc41f0ea75ce950f2dd4ca358823cb2426e1585b7
16f4a14a398042533548d90fc6bcc61b712a2578879a3661cb8e
637ef54a65c1dcd7739ab0f4b572ad8c7f2e8fb9cf23176da8ef
01359c07
```

A.6 Data Objects

The following table contains a list of all the compound data objects used by MICARDO and their component data elements. A data element which is itself a data object is flagged DO.

DO designation (field/location)	Data elements	
File control parameters (file header)	- File type (file descriptor) - File identifier (FID) - Name of a DF or ADF - Free memory space - Name of an EF (SFI) - Life cycle status integer - ARRs for each transfer type	DO
File management data (file header)	not evaluated	
SE data object for passwords (EF_PwdD)	- SE reference data object - ARRs for each transfer type - Transfer formats	DO
ARR per transmission type (EF_PwdD, EF_KeyD, KTM data object)	- AM data object_1 to AM data object_<n> plus one SC data object	DO
AM data object	- AM byte - Command description	
SC data object	- An access condition or a combination of access conditions	DO

DO designation (field/location)	Data elements	
SM data objects	- Data field data object (plaintext data object or cryptogram data object) - Le data object - Status information data object - Response descriptor data object - CRT data object	DO
Response descriptor data object	- CRT for authentication (CCT) or/and - CRT for data confidentiality (CT)	
SE data object for keys (EF_KeyD)	- SE reference - ARRs for each transfer type - CRT for authentication (AT) - CRT for data authentication(CCT) - CRT for digital signature (DST) - CRT for data confidentiality (CT) - Key management template (KTM)	 DO DO DO DO DO DO
AT data object (within an SE data object)	- Usage qualifier - Algorithm ID or object ID - DSI format	
CCT data object (within an SE data object)	- Usage qualifier - Algorithm ID - ICV indicator - Padding indicator	
DST data object (within an SE data object)	- Usage qualifier - Algorithm ID or object ID - DSI format	
CT data object (within an SE data object)	- Usage qualifier - Algorithm ID - ICV indicator	

DO designation (field/location)	Data elements	
KTM data object (within an SE data object)	- Algorithm ID - ARRs for each transfer type - Initial value for operation pointer - CRT for authentication (AT) - CRT for data authentication (CCT) - CRT for data confidentiality (CT) - Key management template (KTM)	 DO DO DO DO DO
KTM data object (within a KTM data object)	- Algorithm ID - CRT for data confidentiality (CT)	 DO
AT data object (EF_SE)	- Usage qualifier - Key reference - DSI format	
CCT data object (EF_SE)	- Usage qualifier - Key reference	
DST data object (EF_SE)	- Usage qualifier - Key reference - DSI format	
CT data object (EF_SE)	- Usage qualifier - Key reference	

A.7 Published Standards

A.7.1 ISO

/ISO 7812-1/

Title:	Identification Cards - Identification of Issuers - Part 1: Numbering System

Identification:	ISO/IEC 7812-1
Version:	Second edition
Date:	2000-09-15
Published by:	International Organization for Standardization/International Electrotechnical Commission

/ISO 7816-1/

Title:	Integrated circuit(s) cards with contacts. Part 1: Physical characteristics
Identification:	ISO/IEC 7816-1
Version:	Second edition
Date:	1998-10-15
Published by:	International Organization for Standardization/International Electrotechnical Commission

/ISO 7816-2/

Title:	Integrated circuit(s) cards with contacts. Part 2: Dimension and location of the contacts
Identification:	ISO/IEC 7816-2
Version:	First edition
Date:	1999-03-01
Published by:	International Organization for Standardization/International Electrotechnical Commission

/ISO 7816-3/

Title:	Integrated circuit(s) cards with contacts. Part 3: Electronic signals and transmission protocols
Identification:	ISO/IEC 7816-3
Version:	Second edition

Date:	December 15, 1997
Published by:	International Organization for Standardization/International Electrotechnical Commission

/ISO 7816-4/

Title:	Integrated circuit(s) cards with contacts. Part 4: Interindustry commands for interchange
Identification:	ISO/IEC 7816-4
Version:	First edition
Date:	01.09.1995
Published by:	International Organization for Standardization/International Electrotechnical Commission

/ISO 7816-4A1/

Title:	Integrated circuit(s) cards with contacts. Part 4: Interindustry commands for interchange. AMENDMENT 1: Impact of secure messaging on the structures of APDU messages
Identification:	ISO/IEC 7816-4
Date:	15.12.1997
Published by:	International Organization for Standardization/International Electrotechnical Commission

/ISO 7816-5/

Title:	Integrated circuit(s) cards with contacts. Part 5: Numbering system and registration procedures for application identifiers
Identification:	ISO/IEC 7816-5
Version:	First edition
Date:	15.06.1994
Published by:	International Organization for Standardization/International Electrotechnical Commission

/ISO 7816-6/

Title:	Integrated circuit(s) cards with contacts. Part 6: Interindustry data elements
Identification:	ISO/IEC 7816-6
Version:	First edition
Date:	15.05.1996
Published by:	International Organization for Standardization/International Electrotechnical Commission

/ISO 7816-8/

Title:	Integrated circuit(s) cards with contacts. Part 8: Interindustry commands for interchange
Identification:	ISO/IEC FDIS 7816-8
Date:	June 25, 1998
Published by:	International Organization for Standardization/International Electrotechnical Commission

/ISO 7816-9/

Title:	Integrated circuit(s) cards with contacts. Part 9: Enhanced inter-industry commands
Identification:	ISO/IEC 7816-9
Version:	First edition
Date:	2000-09-01
Published by:	International Organization for Standardization/International Electrotechnical Commission

/ISO 8825-1/

Title:	Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)

Identification:	ISO/IEC 8825-1
Version:	First edition
Date:	15.10.1995
Published by:	International Organization for Standardization/International Electrotechnical Commission

/ISO 9564-1/

Title:	Banking - Personal Identification Number Management and Security, Part 1: PIN protection principles and techniques
Identification:	ISO/IEC 9564-1
Date:	15.12.1991
Published by:	International Organization for Standardization/International Electrotechnical Commission

A.7.2 Others

/ANSIX392/

Title:	Data Encryption Algorithm
Identification:	X3.92
Date:	1981
Published by:	American National Standards Institute

/EMV2000/

Title:	EMV '96 Integrated Circuit Card Specification for Payment Systems
Version:	4.0
Date:	December 2000
Published by:	Europay Int. S.A., Mastercard Int. Inc., Visa Int. Service Association

/DINV66291-1/

Title:	Spezifikation der Schnittstelle zu Chipkarten mit Digitaler Signatur-Anwendung/Funktion nach SigG und SigV (specification of the interface to chip cards and digital signature applications or functions according to SigG and SigV)
Identification:	DIN V66291-1
Version:	1.0
Date:	15.12.1998
Author:	NI-17.4
Published by:	Deutsches Institut für Normung e.V.

/FIPS180-1/

Title:	Secure Hash Standard
Identification:	FIPS PUB 180-1
Date:	11.05.1993
Author:	Computer Systems Laboratory
Published by:	National Institute of Standards and Technology (NIST)

/HPC1.0/

Title:	German Health Professional Card - Physicians
Version:	1.0
Date:	July 1999
Published by:	National Association of Office Based Physicians German Medical Association

/ITSEC/

Title:	Kriterien für die Bewertung der Sicherheit von Systemen der Informationstechnik (ITSEC)
Identification:	ISBN 92-826-3003-X

Version:	1.2
Date:	Juni 1991

/PKCS1/

Title:	PKCS #1: RSA Encryption Standard
Identification:	PKCS #1
Version:	Version 2.0
Date:	1 October 1998
Published by:	RSA Laboratories

/ZKA41/

Title:	Schnittstellenspezifikation für die ZKA-Chipkarte - Kommandos und Datenstrukturen (interface specification for the ZKA chip card - commands and data structures)
Version:	4.1
Date:	01.07.1999
Published by:	ZKA (Germany's central credit reference agency)

Index