Bundesamt
für Sicherheit in der
Informationstechnik

Technical Guideline TR-03110

Advanced Security Mechanisms for Machine Readable Travel Documents –

Extended Access Control (EAC),
Password Authenticated Connection Establishment (PACE),
and Restricted Identification (RI)

Version 2.01

History

Version	Date	Comment
1.00	2006-02-08	Initial public version.
1.01	2006-11-02	Minor corrections and clarifications.
1.10	2007-08-20	Revised version.
1.11	2008-02-21	Minor corrections and clarifications.
2.00	2008-10-27	Enhanced version.
2.01	2009-05-05	Minor corrections and clarifications. Additional Mapping for PACE.

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63
53133 Bonn

E-Mail: ExtendedAccessControl@bsi.bund.de
Internet: http://www.bsi.bund.de

Contents

List of Figures

List of Tables

1 Introduction

The International Civil Aviation Organization (ICAO) standardizes machine readable travel documents in ICAO Doc 9303. This standard consists of three parts:

Part 1: Machine Readable Passports

- Volume 1: Passports with Machine Readable Data Stored in Optical Character Recognition Format
- Volume 2: Specifications for Electronically Enabled Passports with Biometric Identification Capability

Part 2: Machine Readable Visa

Part 3: Machine Readable Official Travel Documents

- Volume 1: Official Travel Documents with Machine Readable Data Stored in Optical Character Recognition Format
- Volume 2: Specifications for Electronically Enabled Official Travel Documents with Biometric Identification Capability

This technical guideline mainly focuses on and extends the electronic security mechanisms for electronic travel documents described in Doc 9303 Part 1 Volume 2 [8] and Doc 9303 Part 3 Volume 2 [9] to protect the authenticity (including integrity), originality, and confidentiality of the data stored on the radio frequency chip embedded in the travel document (MRTD chip). In a nutshell the security mechanisms specified in [8], [9] are *Passive Authentication*, *Active Authentication*, and *Access Control* as summarized in Table 1.1.

Mechanism	**Protection**	**Cryptographic Technique**
Passive Authentication	Authenticity	Digital Signature
Active Authentication	Originality	Challenge-Response
Access Control	Confidentiality	Authentication & Secure Channels

Table 1.1: ICAO Security Mechanisms

While the implementation of Passive Authentication is mandatory, Active Authentication and Access Control are both optional. It directly follows that without implementing those or equivalent mechanisms the originality and confidentiality of the stored data cannot be guaranteed. This guideline focuses on those aspects and specifies supplementary mechanisms for authentication and access control that are important for a secure MRTD chip.

1.1 Passive Authentication

The ICAO ePassport application basically consists of 16 data groups (DG1-DG16) and a Security Object for Passive Authentication. An overview on the usage of those data groups is given in Table 1.2.

DG	Content	Read/ Write	Mandatory/ Optional	Access Control (ICAO/EAC version 1)		Access Control (EAC version 2)	
				BAC	EAC	PACE	EAC
DG1	MRZ	R	m	m	x	m	r
DG2	Biometric: Face	R	m	m	x	m	r
DG3	Biometric: Finger	R	o	m	m	m	m
DG4	Biometric: Iris	R	o	m	m	m	m
...		R	o	m	o	m	r
DG14	SecurityInfos	R	o	m	x	m	r
DG15	Active Authentication	R	o	m	x	m	r
DG16	...	R	o	m	x	m	r
SO_D	Document Security Object	R	m	m	x	m	r

DG14 is defined in Appendix A.1. The abbreviations (o,r,m,x) are described in Table 1.3.

Table 1.2: Data Groups of the ePassport Application

Passive Authentication uses a digital signature to authenticate data stored in the data groups on the MRTD chip. This signature is generated by a Document Signer (e.g. the MRTD producer) in the personalization phase of the MRTD chip over a Security Object containing the hash values of all data groups stored on the chip. For details on the Security Object, Document Signers, and Country Signing CAs the reader is referred to [8], [9].

To verify data stored on an MRTD chip using Passive Authentication the terminal has to perform the following steps:

1. Read the Security Object from the MRTD chip.
2. Retrieve the corresponding Document Signer Certificate, the trusted Country Signing CA Certificate, and the corresponding Certificate Revocation List.
3. Verify the Document Signer Certificate and the signature of the Security Object.
4. Compute hash values of read data groups and compare them to the hash values in the Security Object.

Passive Authentication enables a terminal to detect manipulated data groups, but it does not prevent cloning of MRTD chips, i.e. copying the complete data stored on one MRTD chip to another MRTD chip.

Note: Even with Passive Authentication, a terminal can detect a cloned MRTD chip by carefully comparing the machine readable zone (MRZ) printed on the datapage to the MRZ stored in data group DG1 on the MRTD chip. This test, however, assumes that it is impossible to physically copy the datapage – which exclusively relies on its physical security features.

1.2 Active Authentication

Active Authentication is a digital security feature that prevents cloning by introducing a chip-individual key pair:

- The public key is stored in data group DG15 and thus protected by Passive Authentication.
- The corresponding private key is stored in secure memory and may only be used internally by the MRTD chip and cannot be read out.

Thus, the chip can prove knowledge of this private key in a challenge-response protocol, which is called Active Authentication. In this protocol the MRTD chip digitally signs a challenge randomly chosen by the terminal. The terminal recognizes that the MRTD chip is genuine if and only if the returned signature is correct. Active Authentication is a straightforward protocol and prevents cloning very effectively, but introduces a privacy threat: Challenge Semantics (see Appendix I for a discussion on Challenge Semantics).

1.3 Access Control

Access Control is not only required for privacy reasons but also mitigates the risk of cloning attacks. The MRTD chip protects the stored data against unauthorized access by applying appropriate access control mechanisms as described below:

- Less-sensitive data (e.g. the MRZ, the facial image and other data that is relatively easy to acquire from other sources) required for global interoperable border crossing is protected by *Basic Access Control*. For the reader's convenience, Basic Access Control is described in Appendix H.
- Sensitive data (e.g. fingerprints and other data that cannot be obtained easily from other sources at a large scale) must only be available to authorized terminals. Such data is additionally protected by *Extended Access Control*.

Basic Access Control only checks that the terminal has physical access to the travel document by requiring the MRZ to be read optically. Extended Access Control should additionally check that the terminal is entitled to read sensitive data. Therefore, strong authentication of the terminal is required. However, as Extended Access Control is not required for global interoperable border crossing, this protocol is not (yet) specified by ICAO.

The Password Authenticated Connection Establishment (PACE) introduced in this specification may be used as a more secure and more convenient replacement for Basic Access Control. Extended Access Control in version 2 MUST be used in combination with PACE instead of Basic Access Control .

Note: If compliance to ICAO Doc 9303 [8], [9] is REQUIRED, Basic Access Control and Extended Access Control in version 1 MUST be used.

1.4 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [2]. The key word "CONDITIONAL" is to be interpreted as follows:

CONDITIONAL: The usage of an item is dependent on the usage of other items. It is therefore further qualified under which conditions the item is REQUIRED or RECOMMENDED.

When used in tables (profiles), the key words are abbreviated as shown in Table 1.3.

Key word		**Abbrev.**
MUST / SHALL	REQUIRED	m
MUST NOT / SHALL NOT	–	x
SHOULD	RECOMMENDED	r
MAY	OPTIONAL	o
–	CONDITIONAL	c

Table 1.3: Key words

1.5 Abbreviations

The following abbreviations are commonly used throughout this specification.

Name	**Abbreviation**
Binary Coded Digit	BCD
Card Verifiable	CV
Card Security Object	SO_C
Certification Authority	CA
Chip Identifier	ID_{PICC}
Chip Authentication Public Key	PK_{PICC}
Chip Authentication Private Key	SK_{PICC}
Country Signing CA	CSCA
Country Verifying CA	CVCA
Country Verifying CA Certificate	C_{CVCA}
Document Security Object	SO_D
Data Group	DG
Document Verifier	DV
Document Verifier Certificate	C_{DV}
Domain Parameters	D
Ephemeral Private Key	$\widetilde{SK}$
Ephemeral Public Key	$\widetilde{PK}$
Hash Function	**H**
International Civil Aviation Organization	ICAO
Key Agreement Function	**KA**
Key Derivation Function	**KDF**
Logical Data Structure	LDS
Machine Readable Travel Document	MRTD
Proximity Integrated Circuit Chip	PICC
Proximity Coupling Device	PCD
Restricted Identification Public Key	PK_{ID}
Restricted Identification Private Key	SK_{ID}
Sector Public Key	PK_{Sector}
Sector Private Key	SK_{Sector}
Sector-specific Identifier	I_{ID}^{Sector}
Terminal Authentication Public Key	PK_{PCD}
Terminal Authentication Private Key	SK_{PCD}
Terminal Certificate	C_T

2 Advanced Security Mechanisms

This document specifies three advanced security mechanisms for machine readable travel documents: *Password Authenticated Connection Establishment, Chip Authentication*, and *Terminal Authentication*.

2.1 Synopsis

While PACE and Chip Authentication can both be used as stand-alone protocol to replace Basic Access Control and Active Authentication, respectively, Terminal Authentication may only be used in combination with Chip Authentication.

2.1.1 Password Authenticated Connection Establishment (PACE)

PACE establishes Secure Messaging between an MRTD chip and a terminal based on weak (short) passwords. PACE is an alternative to Basic Access Control, i.e. it enables the MRTD chip to verify that the terminal is authorized to access stored data but has two advantages:

- Strong session keys are provided independent of the strength of the password.
- The entropy of the password(s) used to authenticate the terminal can be very low (e.g. 6 digits are sufficient in general).

Details on the security of Basic Access Control are described in Appendix H.

2.1.2 Chip Authentication (CA)

Chip Authentication establishes Secure Messaging between an MRTD chip and a terminal based on a static key pair stored on the MRTD chip. Chip Authentication is an alternative to the optional ICAO Active Authentication, i.e. it enables the terminal to verify that the MRTD chip is genuine but has two advantages over the original protocol:

- Challenge Semantics are prevented because the transcripts produced by this protocol are non-transferable.
- Besides authentication of the MRTD chip this protocol also provides strong session keys.

Details on Challenge Semantics are described in Appendix I.

2.1.3 Terminal Authentication (TA)

Terminal Authentication enables the MRTD chip to verify that the terminal is entitled to access sensitive data. As the terminal may access sensitive data afterwards, all further communication MUST be protected appropriately. Terminal Authentication therefore also authenticates an ephemeral public key chosen by the terminal that was or will be used to set up Secure Messaging with Chip Authentication. The MRTD chip MUST bind the terminal's access rights to Secure Messaging established by the authenticated ephemeral public key of the terminal.

2.1.4 Secure Messaging

Secure Messaging provides a secure channel (i.e. encrypted and authenticated) between MRTD chip and terminal. Secure Messaging can be set up by Chip Authentication, PACE, or Basic Access Control. The provided security level however depends on the mechanism used to set up Secure Messaging.

2.1.5 Restricted Identification (RI)

Restricted Identification provides a sector-specific identifier for the MRTD chip with the following properties:

- Within each sector the sector-specific identifier of every MRTD chip is unique.
- Across any two sectors, it is computationally impossible to link the sector-specific identifiers of any MRTD chip.[1]

The sector-specific identifier is used to (re-)identify the MRTD chip within each sector. Chip Authentication *and* Terminal Authentication MUST have been successfully performed before Sector Identification is used.

Note: Depending on the hash function used to create the sector-specific identifier, hash collisions may occur.

2.2 Public Key Infrastructure

Terminal Authentication requires the terminal to prove to the MRTD chip that it is entitled to access sensitive data. Such a terminal is equipped with at least one *Terminal Certificate,* encoding the terminal's public key and access rights, and the corresponding private key. After the terminal has proven knowledge of this private key, the MRTD chip grants the terminal access to sensitive data as indicated in the Terminal Certificate.

The PKI required for issuing and validating Terminal Certificates consists of the following entities:

1. Country Verifying CAs (CVCAs)
2. Document Verifiers (DVs)
3. Terminals

1 Depending on the generation of the sectors a trusted third party may or may not be able to link sector identifiers between sectors.

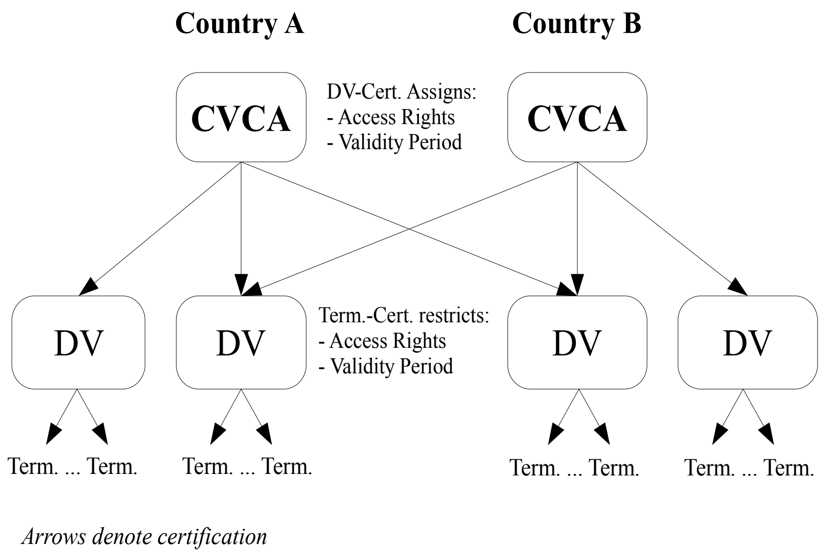

Figure 2.1: Public Key Infrastructure

This PKI forms the basis of Extended Access Control. It is illustrated in Figure 2.1.

2.2.1 Country Verifying CA

Every State is required to set up one trust-point that issues Document Verifier Certificates: the *Country Verifying CA* (CVCA).

Note: The Country Signing CA issuing certificates for Document Signers (cf. [8], [9]) and the Country Verifying CA MAY be integrated into a single entity, e.g. a Country CA. However, even in this case, separate key pairs MUST be used for different roles.

A CVCA determines the access rights to national MRTD chips for all DVs (i.e. official domestic DVs as well as the foreign/commercial DVs) by issuing certificates for DVs entitled to access some sensitive data. The conditions under which a CVCA grants a DV access to sensitive data is out of the scope of this document and SHOULD be stated in a certificate policy (cf. Appendix C.5).

Document Verifier Certificates MUST contain information, such as which data a certain DV is entitled to access. To diminish the potential risk introduced by lost or stolen terminals Document Verifier Certificates MUST contain a short validity period. The validity period is assigned by the issuing CVCA at its own choice and this validity period may differ depending on the Document Verifier the certificate is issued to.

2.2.2 Document Verifiers

A *Document Verifier* (DV) is an organizational unit that manages terminals belonging together (e.g. terminals operated by a State's border police) by – inter alia – issuing Terminal Certificates. A Document Verifier is therefore a CA, authorized by at least the national CVCA to issue certificates for its terminals. The Terminal Certificates issued by a DV usually inherit both the access rights and the validity period from the Document Verifier Certificate, however, the Document Verifier MAY choose to further **restrict** the access rights or the validity period depending on the terminal the certificate is issued for.

If a Document Verifier requires its terminals to access sensitive data stored on other States' MRTD chips, it MUST apply for a DV Certificate issued by the CVCA of the respective States. The Document Verifier MUST also ensure that all received Document Verifier Certificates are forwarded to the terminals within its domain.

2.2.3 Card Verifiable Certificates

CVCA Link Certificates, DV Certificates, and Terminal Certificates are to be validated by MRTD chips. Due to the computational restrictions of those chips, the certificates MUST be in a card verifiable format:

- The certificate format and profile specified in Appendix C.1 SHALL be used.
- The signature algorithm, domain parameters, and key sizes to be used are determined by the CVCA of the issuing State, i.e. the same signature algorithm, domain parameters and key sizes MUST be used within a certificate chain.[2]
- CVCA Link Certificates MAY include a public key that deviates from the current parameters, i.e. the CVCA MAY switch to a new signature algorithm, new domain parameters, or key sizes.

2.2.4 Certificate Scheduling

Each certificate MUST contain a validity period. This validity period is identified by two dates, the *certificate effective date* and the *certificate expiration date.*

2 As a consequence Document Verifiers and terminals will have to be provided with several key pairs.

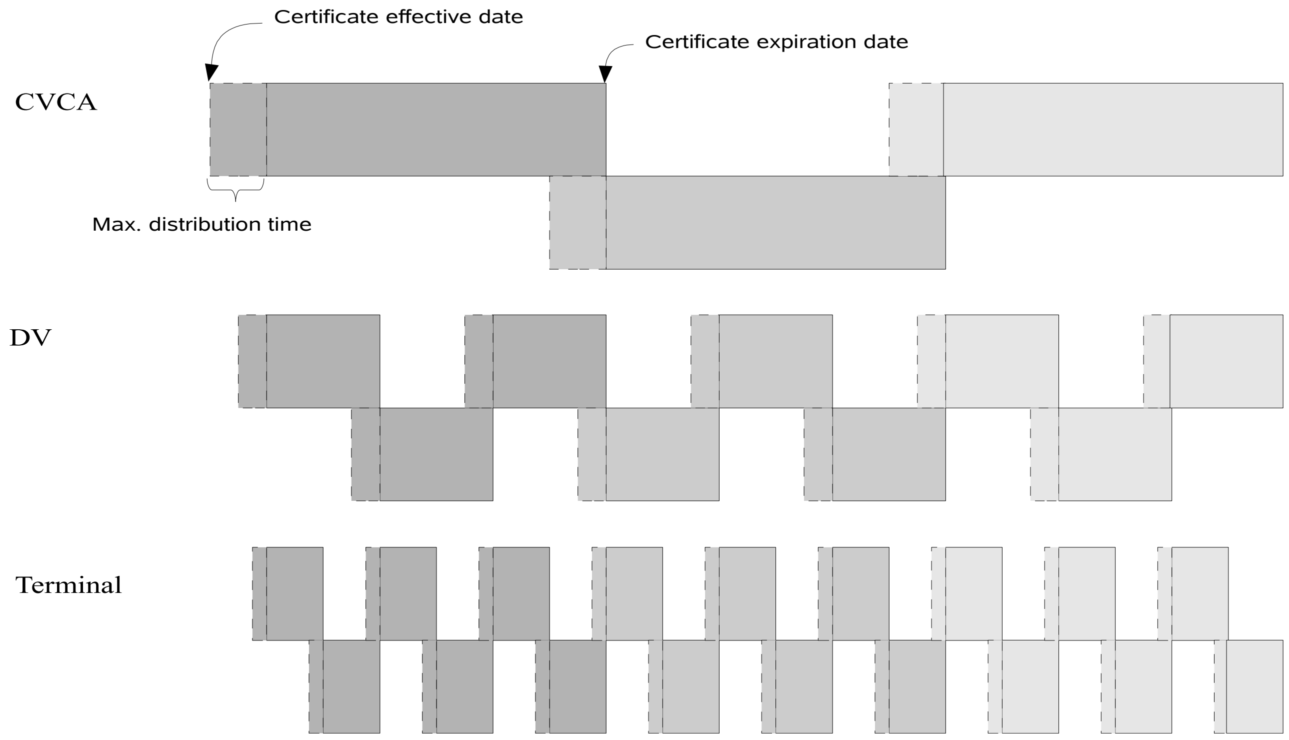

Figure 2.2: Certificate Scheduling

Certificate Effective Date: The certificate effective date SHALL be the date of the certificate generation.

Certificate Expiration Date: The certificate expiration date may be arbitrarily chosen by the certificate issuer.

When generating certificates the issuer MUST carefully plan the roll-over of certificates, as sufficient time for propagation of certificates and set up of certificate chains MUST be provided. Obviously, a new certificate must be generated before the current certificate expires. The resulting *maximum distribution time* equals the certificate expiration date of the old certificate minus the certificate effective date of the new certificate. Certificate scheduling is illustrated in Figure 2.2.

2.2.5 Certificate Validation

To validate a Terminal Certificate, the MRTD chip MUST be provided with a certificate chain starting at a trust-point stored on the MRTD chip. Those trust-points are more or less recent public keys of the MRTD chip's CVCA. The initial trust-point(s) SHALL be stored securely in the MRTD chip's memory in the production or (pre-) personalization phase.

As the key pair used by the CVCA changes over time, CVCA Link Certificates have to be produced. The MRTD chip is REQUIRED to internally update its trust-point(s) according to received valid link certificates.

Note: Due to the scheduling of CVCA Link Certificates (cf. Figure 2.2), at most two trust-points per application need to be stored on the MRTD chip.

The MRTD chip MUST accept expired CVCA Link Certificates but it MUST NOT accept expired DV and Terminal Certificates. To determine whether a certificate is expired, the MRTD chip SHALL use its *current date*.

Current Date: If the MRTD chip has no internal clock, the current date SHALL be approximated as described in the following. The current date stored on the MRTD chip is initially the date of the (pre-) personalization. This date is then autonomously approximated by the MRTD chip using the most recent certificate effective date contained in a valid CVCA Link Certificate, a DV Certificate or an *Accurate Terminal Certificate*.

Accurate Terminal Certificate: A Terminal Certificate is accurate, if the issuing Document Verifier is trusted by the MRTD chip to produce Terminal Certificates with the correct certificate effective date.

A terminal MAY send CVCA Link Certificates, DV Certificates, and Terminal Certificates to an MRTD chip to update the current date and the trust-point stored on the MRTD chip even if the terminal does not intend to or is not able to continue with Terminal Authentication.

Note: The MRTD chip only verifies that a certificate is *apparently* recent (i.e. with respect to the approximated current date).

2.2.5.1 General Procedure

The certificate validation procedure consists of two steps:

1. **Certificate Verification:** The signature MUST be valid and unless the certificate is a CVCA Link Certificate, the certificate MUST NOT be expired. If the verification fails, the procedure SHALL be aborted.
2. **Internal Status Update:** The current date MUST be *updated*, the public key and the attributes MUST be imported, new trust-points MUST be *enabled*, expired trust-points MUST be *disabled*.

The operation of *updating* the current date and the operations of *enabling* and *disabling* a trust-point MUST be implemented as an atomic operation.

Enabling a trust-point: The new trust-point SHALL be added to the list of trust-points.

Disabling a trust-point: Expired trust-points MUST NOT be used for the verification of DV Certificates but MUST remain usable for the verification of Link Certificates. Disabled trust-points MAY be deleted after the successful import of the successive Link Certificate.

2.2.5.2 Example Procedure

The following validation procedure, provided as an example, MAY be used to validate a certificate chain. For each received certificate the MRTD chip performs the following steps:

1. The MRTD chip verifies the signature on the certificate. If the signature is incorrect, the verification fails.
2. If the certificate is not a CVCA Link Certificate, the certificate expiration date is compared to the MRTD chip's current date. If the expiration date is before the current date, the verification fails.
3. The certificate is valid and the public key and the relevant attributes contained in the certificate are imported.

a) For CVCA, DV, and Accurate Terminal Certificates: The certificate effective date is compared to the MRTD chip's current date. If the current date is before the effective date, the current date is updated to the effective date.

b) For CVCA Link Certificates: The new CVCA public key is added to the list of trust-points stored securely in the MRTD chip's memory. The new trust-point is then enabled.

c) For DV and Terminal Certificates: The new DV or terminal public key is temporarily imported for subsequent certificate verification or Terminal Authentication, respectively.

4. Expired trust-points stored securely in the MRTD chip's memory are disabled and may be removed from the list of trust-points.

2.3 Terminal Sector

To support Restricted Identification terminals MUST be assigned a Terminal Sector. The Terminal Sector SHALL be contained in the Terminal Certificate and thus, it is RECOMMENDED that the Terminal Sector is generated by the certifying Document Verifier. In any case the Terminal Sector MUST NOT be chosen by the terminal itself.

The Terminal Sector is always a public key. It MAY be chosen either verifiably at random with an unknown private key to disable tracing completely (in this case linking sector-specific identifiers across sectors is computationally impossible) or as key pair to enable revocation based on sector-specific identifiers.

2.3.1 Sector Key Pair

Sector Key Pairs MUST be generated by all Document Verifiers that support sector-specific revocation of MRTD chips.

Each Document Verifier SHALL perform the following steps for every subordinated sector:

1. Generate a new Sector Key Pair based on the Revocation Sector Public Key.
2. Store the Sector Private Key securely (at the Document Verifier).
3. Include the Sector Public Key in every Terminal Certificate of all terminals belonging to the corresponding sector.

The Revocation Sector Key Pair SHALL be generated by the CVCA. The CVCA MAY delegate the revocation service to a service provider.

2.3.2 Sector-Specific Revocation of MRTD Chips

At the (pre-) personalization of the MRTD chip a key pair for Restricted Identification SHALL be generated. The private key SHALL be stored in the MRTD chip, the public key SHALL be stored in a database together with other data identifying the holder of the MRTD.

Note: The generation of the key pair for Restricted Identification MAY be performed within the MRTD chip or externally. The key pair MUST be chosen to be unique and MAY be either chip-specific or holder-specific (i.e. the same key pair will be used on subsequent MRTD chips). At least one key pair used for Restricted identification MUST be chip-specific.

To revoke the MRTD chip, the chip-specific public key of the MRTD chip is looked up in the database and transferred to the CVCA. The CVCA then transforms the public key using its Revocation Sector Private Key. The transformed public key is then transferred to all subordinated Document Verifiers. Each Document Verifier calculates the sector-specific identifiers using the Sector Private Keys for all subordinated Terminal Sectors. Finally, the sector-specific identifier is transferred to all terminals of the corresponding sector.

2.3.3 Validity Period

In contrast to the Terminal Key Pair (for Terminal Authentication), the Sector Key Pair is valid for a long time and MUST be chosen appropriately.

2.3.4 Migrating Terminals

To migrate a terminal from one Document Verifier to another Document Verifier, the sector key pair of the terminal MUST be transferred securely to the new Document Verifier.

Note: Migrating a terminal to a Document Verifier supervised by another CVCA is not possible.

3 MRTD Applications

3.1 Applications

This specification supports three applications: *ePassport*, *eID*, and *eSign*.

3.1.1 ePassport Application

The ePassport application is defined by ICAO [8], [9]. To read data from the ePassport application, the MRTD chip SHOULD require the terminal to be authenticated as inspection system. The different authentication procedures for the ePassport application are illustrated in Figure 3.1.

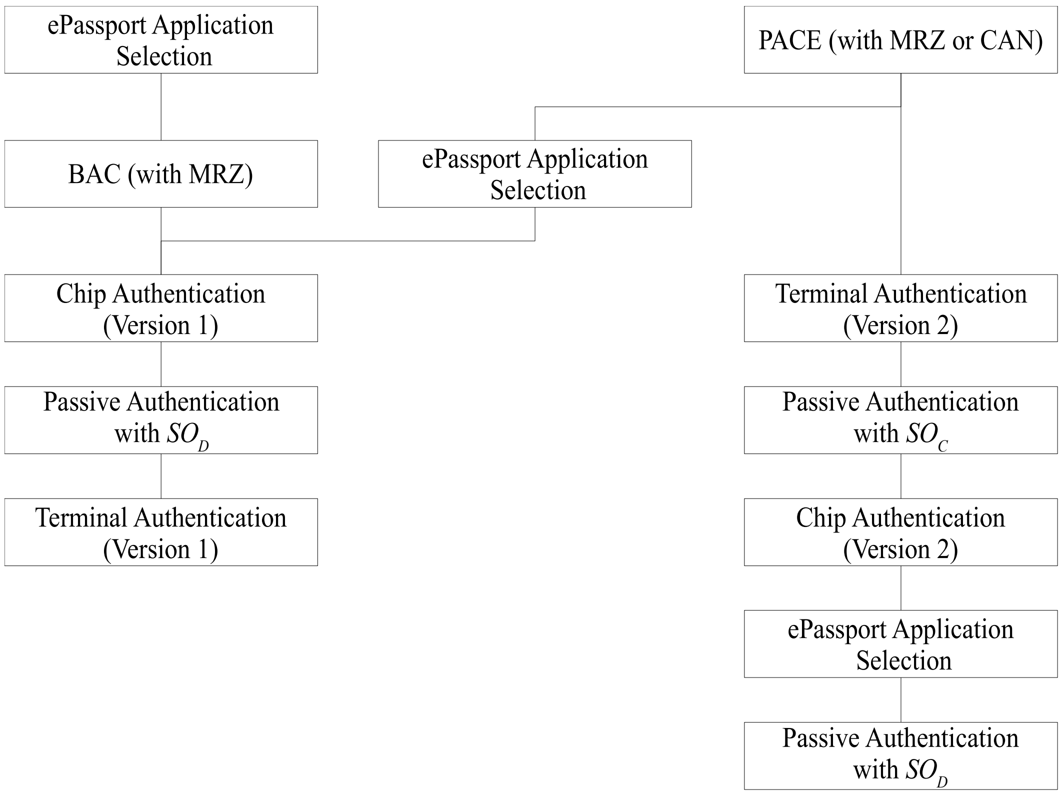
ePassport Application Selection
BAC (with MRZ)
Chip Authentication (Version 1)
Passive Authentication with SO_D
Terminal Authentication (Version 1)
PACE (with MRZ or CAN)
ePassport Application Selection
Terminal Authentication (Version 2)
Passive Authentication with SO_C
Chip Authentication (Version 2)
ePassport Application Selection
Passive Authentication with SO_D

Figure 3.1: Authentication procedures for the ePassport Application

Note: According to this specification it is RECOMMENDED to require Extended Access Control to be used even for less-sensitive data. If compatibility to ICAO [8], [9] is REQUIRED, the MRTD chip SHALL grant access to less-sensitive data to terminals authenticated by Basic Access Control. The relevant inspection procedures are described in Appendix G.

3.1.2 eID Application

The eID application is specified in Appendix E and requires the terminal to be authenticated as follows:

- To write to the eID application, the MRTD chip SHALL require the terminal to be authenticated as authentication terminal with authorization to write to the respective data groups of the eID application.
- To read from the eID application, the MRTD chip SHALL require the terminal to be authenticated as
  - authentication terminal with authorization to read from the respective data groups of the eID application or as
  - inspection system with authorization to read all data groups of the eID application.

To authenticate a terminal as authentication terminal or inspection system, the general authentication procedure (cf. Section 3.4) MUST be used.

3.1.3 eSign Application

This specification does not require compatibility to a certain standard, but requires the terminal to be authenticated as follows:

- To install the eSign application, the MRTD chip SHALL require the terminal to be authenticated as authentication terminal with special authorization to install the eSign application.
- To use the eSign application to generate signatures, the MRTD chip SHALL require the terminal to be authenticated as signature terminal.

To authenticate a terminal as authentication terminal or signature terminal, the general authentication procedure (cf. Section 3.4) MUST be used.

3.2 Terminal Types

This specification supports three terminal types: *inspection systems*, *authentication terminals*, and *signature terminals*.

3.2.1 Inspection System

An inspection system is an official terminal that is always operated by a governmental organization (i.e. a Domestic or Foreign Document Verifier). The MRTD chip SHALL require an inspection system to authenticate itself before access according to the effective authorization is granted. To authenticate a terminal as inspection system, the general authentication procedure (cf. Section 3.4) MUST be used. The authorization level of an inspection system SHALL be determined by the effective authorization calculated from the certificate chain.

3.2.2 Authentication Terminal

An authentication terminal is a terminal that may be operated by a governmental organization (Official Domestic Document Verifier) or by any other organization (Non-Official / Foreign Document Verifier). The MRTD chip SHALL require an authentication terminal to authenticate itself before access according to the effective authorization is granted. To authenticate a terminal as authentication terminal, the general authentication procedure (cf. Section 3.4) MUST be used. The authorization level of an authentication terminal SHALL be determined by the effective authorization calculated from the certificate chain.

3.2.3 Signature Terminal

A signature terminal MUST be approved by either the respective accreditation body or by a certification service provider. The MRTD chip SHALL require a signature terminal to authenticate itself before access according to the effective authorization is granted. To authenticate a terminal as signature terminal, the general authentication procedure (cf. Section 3.4) MUST be used. The authorization level of a signature terminal SHALL be determined by the effective authorization calculated from the certificate chain.

3.3 User Interaction

To allow the holder of the MRTD to control access to the applications implemented on the contactless MRTD chip, Basic and Extended Access Control have been specified. Due to the limitations of Basic Access Control, this specification introduces PACE as a secure and practical mechanism to restrict access to applications based on knowledge, i.e. based on passwords that are either printed on the document or only known to the legitimate holder of the document.

While Basic Access Control only supports one “password”, i.e. a symmetric key derived from the MRZ, PACE protocol supports multiple passwords. The different types of passwords used in this specification are:

CAN: The Card Access Number (CAN) is a short password that is printed or displayed on the document.

PIN: The Personal Identification Number (PIN) is a short secret password that SHALL be only known to the legitimate holder of the document.

PUK: The PIN Unblock Key (PUK) is a long secret password that SHALL be only known to the legitimate holder of the document.

MRZ: The MRZ-Password is a secret key that is derived from the machine readable zone and may be used for both PACE and BAC.

Note: While this Technical Guideline does not recommend any specific size for passwords, a long password MUST contain sufficient entropy to protect against brute-force attacks without blocking the password after a certain number of incorrect trials.

3.3.1 CAN

The CAN may be used to access all applications on the MRTD chip (ePassport, eID, and eSign application).

The CAN is a non-blocking password, i.e. the MRTD chip MUST NOT block the CAN after failed authentications. The CAN may be static (printed on the MRTD), semi-static (e.g. printed on a label on the MRTD) or dynamic (randomly chosen by the MRTD chip and displayed on the MRTD using e.g. ePaper, OLED or similar technologies).

3.3.2 PIN

The PIN is a short secret user password that may only be used to access the eID application. Usage of the PIN is REQUIRED for all authentication terminals, i.e. only the legitimate holder may allow an authentication terminal to access data stored on the eID application, unless the terminal has the effective access right to access the eID-application with the CAN.

The PIN is a blocking password, i.e. the PIN is associated with a retry counter (*RC*) that is decreased for every failed authentication. The following blocking procedure SHALL be enforced by the MRTD chip to prevent denial of service attacks:

$RC = 1$: The MRTD chip SHALL *suspend* the PIN, i.e. the MRTD chip MUST reject authentication attempts until the PIN is resumed. To *resume* the suspended PIN the CAN MUST be entered correctly. The resumed state SHALL be volatile, i.e. the PIN MUST be entered in the same session, otherwise (e.g. after a power down) the PIN SHALL remain suspended. The retry counter *RC* SHALL be set according to the entered PIN:

- If the PIN is entered correctly: The retry counter *RC* is reset to the initial value.
- If the PIN is entered incorrectly: The retry counter is decreased to $RC = 0$.

$RC = 0$: The MRTD chip SHALL *block* the PIN, i.e. the MRTD chip MUST NOT accept any further authentication attempt using the blocked PIN. To *unblock* the blocked PIN an unblocking procedure MUST be used to reset the corresponding retry counter and, where applicable, to set a new PIN.

3.3.3 PUK

The PUK is a long secret user password that may only be used to access the unblocking mechanism of the PIN.

The PUK is a non-blocking password, i.e. the MRTD chip MUST NOT block the PUK after failed authentications. It MAY however associate the PUK with a usage counter that is decreased for every successful authentication.

3.3.4 MRZ

The MRZ-Password may only be used for the ePassport application (for both PACE and BAC). The MRZ-Password is a non-blocking static symmetric key derived from the MRZ. The MRZ-Password is therefore only available if the MRTD contains a printed MRZ.

3.4 General Authentication Procedure

The general authentication procedure is REQUIRED for all terminals to access applications on the MRTD chip. It consists of the following steps:

1. **PACE** **(REQUIRED)**

   The terminal MUST indicate the terminal type and required access rights as part of PACE.

   - An inspection system SHALL use the CAN or the MRZ-Password.

- An authentication terminal SHALL use the PIN unless the effective authorization of the terminal allows to use the CAN (“CAN allowed”).
- A signature terminal SHALL use the CAN.

If successful, the MRTD chip performs the following:

- It SHALL start secure messaging.
- It SHALL provide trustpoints for Terminal Authentication.

2. **Terminal Authentication** **(REQUIRED)**

   The terminal SHALL authenticate the ephemeral public key *to be used* later on for Chip Authentication.

   If successful, the MRTD chip performs the following:

   - It SHALL grant read/write access to data groups according to the terminal’s access rights.
   - It SHALL restrict those access rights to Secure Messaging to be established by the authenticated ephemeral public key (except Card Security Object).

3. **Passive Authentication** **(REQUIRED)**

   The terminal SHALL read and verify the Card Security Object.

4. **Chip Authentication** **(REQUIRED)**

   The MRTD chip SHALL restart secure messaging.

An authenticated terminal may select and use the application(s) according to the effective authorization of the terminal.

Note: The terminal and the MRTD chip MUST use the established security context (i.e. secure messaging established by Chip Authentication) for all further communication.

3.4.1 Online Authentication

The eID application can also be used online, i.e. MRTD chip and authentication terminal are connected by a network. In this case a *local terminal* and a *remote terminal* are distinguished:

Remote Terminal: The remote terminal is authorized to access eID data. It provides the local terminal with a chain of Terminal Authentication certificates and a digital signature created on the MRTD chip’s challenge with the corresponding private key.

Local Terminal: The local terminal interacts with the user, the MRTD chip, and the remote terminal but is not authorized to access eID data. The chain of Terminal Authentication certificates received from the remote terminal are displayed to the user and only if the user accepts, the local terminal forwards the received certificates to the MRTD chip.

Note: Only after Chip Authentication when a secure end-to-end connection between MRTD chip and remote terminal is established, the MRTD chip grants access to eID data.

3.5 PIN Management

The PIN and the CAN are the only passwords (used for PACE) that may be changed. The PIN is the only password that may have the states suspended and blocked. Furthermore the PIN may have the states activated and deactivated. The remaining passwords (PUK and MRZ) are static and non-blocking. Details on the usage of the passwords are described in Section 3.3.

PIN management consists of the following operations:

- Change CAN
- Change PIN
- Resume PIN
- Unblock PIN
- Activate PIN
- Deactivate PIN

A mapping of the PIN management mechanisms Change CAN, Change PIN, Unblock PIN, Activate PIN, and Deactivate PIN to ISO 7816 commands is given in Appendix B.6. The operation Resume PIN is not mapped to an ISO 7816 command as this is implicitly performed by the MRTD chip.

3.5.1 Unauthenticated Terminals

A terminal is *unauthenticated* before successfully completing Terminal Authentication. Unauthenticated terminals may perform PIN management operations as follows:

1. **PACE** **(REQUIRED)**

   The terminal SHOULD NOT indicate the terminal type and required access rights as the terminal remains unauthenticated. The terminal may choose to use the CAN, PIN, or PUK as password for PACE.

   If successful, the MRTD chip performs the following:

   - It SHALL start secure messaging.
   - If the PIN is operational (i.e. activated, and neither suspended nor blocked) and the PIN was successfully used, the MRTD chip performs the following:
     - It SHALL reset the retry counter of the PIN.
     - It SHALL grant access to the following PIN management mechanism: Change PIN
   - If the CAN was successfully used:
     - It SHALL temporarily resume the PIN.
   - If the PUK was successfully used:
     - It SHALL grant access to the following PIN management mechanism: Unblock PIN
     - It MAY grant access to the following PIN management mechanism: Change PIN

2. **PACE with PIN** **(CONDITIONAL)**

   This step is REQUIRED in the following cases:

   - To resume the PIN.
   - To proceed with the general authentication procedure after PIN management. In this case the terminal MUST indicate the terminal type (authentication terminal) and required access rights.[3]

   If the PIN was successfully used and the PIN is operational or temporarily resumed, the MRTD chip performs the following:

   - If the PIN is temporarily resumed it SHALL resume the PIN.
   - It SHALL reset the retry counter of the PIN.
   - It SHALL grant access to the following PIN management mechanism: Change PIN

3. **PIN Management** **(CONDITIONAL)**

   This step is REQUIRED to change or unblock the PIN.

   The MRTD chip performs the following:

   - It SHALL allow the terminal to perform the following PIN management operations:
     - Change PIN, if the terminal has access to this operation.
     - Unblock PIN, if the terminal has access to this operation.

If the PUK is associated with a usage counter, the usage counter MUST NOT be expired and the MRTD chip SHALL decrease the usage counter either after a successful execution of the PACE protocol or after performing the operation Unblock PIN.

Note: Support for the PIN management operation Change PIN if PACE with PUK was used is OPTIONAL and implementation specific.

3.5.2 Authenticated Terminals

An Authentication Terminal with effective authorization for PIN management (cf. Appendix C.4.1.2) may perform PIN management operations as follows:

1. **General Authentication Procedure** **(REQUIRED)**

   If the terminal is authenticated as Authentication Terminal with effective authorization for PIN Management, the MRTD chip performs the following:

   - It SHALL grant access to the PIN management mechanisms.

2. **PIN Management** **(REQUIRED)**

   The MRTD chip performs the following:

   - It MAY allow the terminal to perform the following PIN management operations:
     - Change PIN

3 This is most likely the case if the CAN is used to resume the PIN as part of the general authentication procedure.

  - Change CAN
  - Unblock PIN
- It MUST allow the terminal to perform the following PIN management operations:
  - Activate PIN
  - Deactivate PIN

Note: Support for the PIN management operations Change PIN and Change CAN is OPTIONAL and implementation specific.

3.6 MRTDs with Display

If the MRTD is equipped with a display, the MRTD chip should use the display as follows:

- It SHALL display the selected application.
- It SHALL dynamically choose and display the CAN.
- It SHALL display the authenticated terminal's identification and effective access rights.
- It MAY display variable data of the eID application, e.g. the Normal Place of Residence.

4 Protocol Specifications

In this section cryptographic protocols for PACE, Chip Authentication and Terminal Authentication are specified assuming an arbitrary communication infrastructure. A mapping to ISO 7816 commands is given in Appendix B.

4.1 Cryptographic Algorithms and Notation

The protocols are executed between two parties: the MRTD chip (PICC) and the terminal (PCD). Table 4.1 gives an overview on the key pairs used. The following cryptographic operations and notations are used.

4.1.1 Hash Algorithms

The operation for computing a cryptographic hash is described in an algorithm-independent way.

4.1.1.1 Operations

- The operation for computing a hash over a message m is denoted by $\mathbf{H}(m)$.

4.1.2 Symmetric Key Algorithms

The keys and operations for symmetric key encryption and authentication are described in an algorithm-independent way.

Protocol	MRTD Chip	Terminal	Note
PACE	$\widetilde{PK}_{PICC}$, $\widetilde{SK}_{PICC}$	$\widetilde{PK}_{PCD}$, $\widetilde{SK}_{PCD}$	All key pairs are ephemeral key pairs.
Chip Authentication	PK_{PICC} , SK_{PICC}	$\widetilde{PK}_{PCD}$, $\widetilde{SK}_{PCD}$	The key pair used by the terminal is an ephemeral key pair different from the ephemeral PACE key pair.
Terminal Authentication	PK_{CVCA}	PK_{PCD} , SK_{PCD}	The MRTD chip verifies the certificate chain received from the terminal using the public key of the CVCA.
Restricted Identification	SK_{ID}	PK_{Sector}	The MRTD SHOULD NOT provide the corresponding public key PK_{ID}, the terminal MUST NOT be provided the corresponding private key SK_{Sector}. The keys PK_{ID} and SK_{Sector} are externally used to generate revocation lists (cf. Section 4.5.3)

Table 4.1: Overview of key pairs used

4.1.2.1 Keys

Symmetric keys are derived from a shared secret K and an OPTIONAL nonce r or from a password π using a Key Derivation Function (KDF):

- Deriving a key for message encryption is denoted by $K_{Enc}=\mathbf{KDF_{Enc}}(K,[r])$.
- Deriving a key for message authentication is denoted by $K_{MAC}=\mathbf{KDF_{MAC}}(K,[r])$.
- Deriving a key from a password is denoted by $K_{\pi}=\mathbf{KDF_{\pi}}(\pi)$.

4.1.2.2 Operations

The operations for encrypting and decrypting a message are denoted as follows:

- Encrypting a plaintext m with key K_{Enc} is denoted by $c=\mathbf{E}(K_{Enc},m)$.
- Decrypting a ciphertext c with key K_{Enc} is denoted by $m=\mathbf{D}(K_{Enc},c)$.

The operation for computing an authentication code T on message m with key K_{MAC} is denoted as $T=\mathbf{MAC}(K_{MAC},m)$

4.1.3 Key Agreement

The keys and operations for key agreement are described in an algorithm-independent way. A mapping to DH and ECDH can be found in Appendix A.2.

4.1.3.1 Keys

The following key pairs are used for PACE and Chip Authentication:

- For PACE both the MRTD chip and the terminal generate ephemeral Diffie-Hellman key pairs based on the ephemeral domain parameters $\widetilde{D}$.
  - The MRTD chip's ephemeral public key is $\widetilde{PK_{PICC}}$, the corresponding private key is $\widetilde{SK_{PICC}}$.
  - The terminal's ephemeral public key is $\widetilde{PK_{PCD}}$, the corresponding private key is $\widetilde{SK_{PCD}}$.
- For Chip Authentication the MRTD chip uses a static Diffie-Hellman key pair and the terminal generates an ephemeral key pair based on the MRTD chip's static domain parameters D_{PICC}.
  - The MRTD chip's static public key is PK_{PICC}, the corresponding private key is SK_{PICC}.
  - The terminal's ephemeral public key is $\widetilde{PK_{PCD}}$, the corresponding private key is $\widetilde{SK_{PCD}}$.
  - The terminal's compressed ephemeral public key is denoted by $\mathbf{Comp}(\widetilde{PK_{PCD}})$.
- For Restricted Identification the MRTD chip uses a static Diffie-Hellman key pair and the terminals within a sector use an (almost) static Diffie-Hellman key pair where the private key is not known to the terminal.
  - The MRTD chip's static public key is PK_{ID}, the corresponding private key is SK_{ID}.

– The sector's static public key is PK_{Sector}, the corresponding private key is SK_{Sector}.

– The revocation sector public key $PK_{Revocation}$, the corresponding private key is $SK_{Revocation}$.

– The sector-specific identifier is $I_{ID}^{Sector.}$.

It is RECOMMENDED that the MRTD chip validates public keys received from the terminal.

Note: The terminal will have to use different ephemeral public keys for PACE and Chip Authentication. As the ephemeral keys are context specific, the same notation is used.

4.1.3.2 Operations

The operation for generating a shared secret K is denoted by $K = \mathbf{KA}(SK, PK, D)$, where SK is a (ephemeral or static) secret key, PK is a (ephemeral or static) public key and D are the (ephemeral or static) domain parameters.

4.1.4 Signatures

The keys and operations for signatures are described in an algorithm-independent way. A mapping to RSA and ECDSA can be found in Appendix A.6.

4.1.4.1 Keys

For Terminal Authentication the following key pair is used:

- The terminal has a static authentication key pair. The public key is PK_{PCD}, the corresponding private key is SK_{PCD}.

4.1.4.2 Operations

The operations for signing and verifying a message are denoted as follows:

- Signing a message m with private key SK_{PCD} is denoted by $s = \mathbf{Sign}(SK_{PCD}, m)$.
- Verifying the resulting signature s with public key PK_{PCD} is denoted by $\mathbf{Verify}(PK_{PCD}, s, m)$.

4.2 PACE

The PACE Protocol is a password authenticated Diffie-Hellman key agreement protocol that provides secure communication and implicit password-based authentication of the MRTD chip and the terminal (i.e. MRTD chip and terminal share the same password π).

MRTD Chip (PICC)		**Terminal (PCD)**
static domain parameters D_{PICC}		
choose random nonce $s \in_R Dom(E)$		
$z = \mathbf{E}(K_\pi, s)$	$\xrightarrow[z]{D_{PICC}}$	$s = \mathbf{D}(K_\pi, z)$
additional data required for $\mathbf{Map}()$	$\longleftrightarrow$	additional data required for $\mathbf{Map}()$
$\tilde{D} = \mathbf{Map}(D_{PICC}, s)$		$\tilde{D} = \mathbf{Map}(D_{PICC}, s)$
choose random ephemeral key pair $(\widetilde{SK_{PICC}}, \widetilde{PK_{PICC}}, \tilde{D})$		choose random ephemeral key pair $(\widetilde{SK_{PCD}}, \widetilde{PK_{PCD}}, \tilde{D})$
	$\overset{\widetilde{PK_{PCD}}}{\underset{\widetilde{PK_{PICC}}}{\longleftrightarrow}}$	
$K = \mathbf{KA}(\widetilde{SK_{PICC}}, \widetilde{PK_{PCD}}, \tilde{D})$		$K = \mathbf{KA}(\widetilde{SK_{PCD}}, \widetilde{PK_{PICC}}, \tilde{D})$
	$\xleftarrow{T_{PCD}}$	$T_{PCD} = \mathbf{MAC}(K_{MAC}, (\widetilde{PK_{PICC}}, \tilde{D}))$
$T_{PICC} = \mathbf{MAC}(K_{MAC}, (\widetilde{PK_{PCD}}, \tilde{D}))$	$\xrightarrow{T_{PICC}}$	

Figure 4.1: PACE

4.2.1 Protocol Specification

The following steps are performed by the terminal and the MRTD chip, a simplified version is also shown in Figure 4.1:

1. The MRTD chip randomly and uniformly chooses a nonce s, encrypts the nonce to $z = \mathbf{E}(K_\pi, s)$, where $K_\pi = \mathbf{KDF}_\pi(\pi)$ is derived from the shared password π, and sends the ciphertext z together with the static domain parameters D_{PICC} to the terminal.
2. The terminal recovers the plaintext $s = \mathbf{D}(K_\pi, z)$ with the help of the shared password π.
3. Both the MRTD chip and the terminal perform the following steps:
   a) They compute the ephemeral domain parameters $\tilde{D} = \mathbf{Map}(D_{PICC}, s)$.
   b) They perform an anonymous Diffie-Hellman key agreement based on the ephemeral domain parameters and generate the shared secret
   $$K = \mathbf{KA}(\widetilde{SK_{PICC}}, \widetilde{PK_{PCD}}, \tilde{D}) = \mathbf{KA}(\widetilde{SK_{PCD}}, \widetilde{PK_{PICC}}, \tilde{D}).$$
   c) They derive session keys $K_{MAC} = \mathbf{KDF_{MAC}}(K)$ and $K_{Enc} = \mathbf{KDF_{Enc}}(K)$.
   d) They exchange and verify the authentication token $T_{PCD} = \mathbf{MAC}(K_{MAC}, (\widetilde{PK_{PICC}}, \tilde{D}))$ and $T_{PICC} = \mathbf{MAC}(K_{MAC}, (\widetilde{PK_{PCD}}, \tilde{D}))$.

MRTD Chip (PICC)		**Terminal (PCD)**
static key pair $(SK_{PICC}, PK_{PICC}, D_{PICC})$		
	$\xrightarrow[D_{PICC}]{PK_{PICC}}$	
(CONDITIONAL: Version 1)		[choose random ephemeral key pair $(\widetilde{SK_{PCD}}, \widetilde{PK_{PCD}}, D_{PICC})$]
	$\xleftarrow{\widetilde{PK_{PCD}}}$	
$K = \mathbf{KA}(SK_{PICC}, \widetilde{PK_{PCD}}, D_{PICC})$		$K = \mathbf{KA}(\widetilde{SK_{PCD}}, PK_{PICC}, D_{PICC})$
[choose r_{PICC} randomly]		(CONDITIONAL: Version 2)
$[T_{PICC} = \mathbf{MAC}(K_{MAC}, (\widetilde{PK_{PCD}}, D_{PICC}))]$	$\xrightarrow[r_{PICC}]{T_{PICC}}$	

Figure 4.2: Chip Authentication

4.2.2 Security Status

If PACE was successfully performed then the MRTD chip has verified the used password. Secure Messaging is started using the derived session keys K_{MAC} and K_{Enc}. The MRTD chip MUST NOT accept more than one execution of PACE within the same session unless a suspended PIN has to be resumed using an unauthenticated terminal (cf. Section 3.5.1) with the CAN as password. In this case the second execution of PACE MUST be protected by Secure Messaging established by the first execution. If the second execution of PACE was successfully performed the MRTD chip has verified the PIN. Secure Messaging is restarted using the new derived session keys K_{MAC} and K_{Enc}. Otherwise, if the second execution of PACE was not successful, Secure Messaging is continued using the previously established session keys.

4.3 Chip Authentication

The Chip Authentication Protocol is an ephemeral-static Diffie-Hellman key agreement protocol that provides secure communication and unilateral authentication of the MRTD chip.

Version 1: The protocol provides implicit authentication of both the MRTD chip itself and the stored data by performing Secure Messaging using the new session keys.

Version 2: The protocol provides explicit authentication of the MRTD chip by verifying the authentication token and implicit authentication of the stored data by performing Secure Messaging using the new session keys.

4.3.1 Protocol Specification

The following steps are performed by the terminal and the MRTD chip, a simplified version is also shown in Figure 4.2:

4.3.1.1 Version 1

1. The MRTD chip sends its static Diffie-Hellman public key PK_{PICC}, and the domain parameters D_{PICC} to the terminal.
2. The terminal generates an ephemeral Diffie-Hellman key pair $(\widetilde{SK_{PCD}}, \widetilde{PK_{PCD}}, D_{PICC})$, and sends the ephemeral public key $\widetilde{PK_{PCD}}$ to the MRTD chip.
3. Both the MRTD chip and the terminal compute the following:
   a) The shared secret $K = \mathbf{KA}(SK_{PICC}, \widetilde{PK_{PCD}}, D_{PICC}) = \mathbf{KA}(\widetilde{SK_{PCD}}, PK_{PICC}, D_{PICC})$
   b) The session keys $K_{MAC} = \mathbf{KDF_{MAC}}(K)$ and $K_{Enc} = \mathbf{KDF_{Enc}}(K)$ derived from K for Secure Messaging.
   c) The terminal's compressed ephemeral public key $\mathbf{Comp}(\widetilde{PK_{PCD}})$ for Terminal Authentication.

To verify the authenticity of the PK_{PICC} the terminal SHALL perform Passive Authentication.

4.3.1.2 Version 2

In this version Terminal Authentication MUST be performed before Chip Authentication, as the terminal's ephemeral key pair $(\widetilde{SK_{PCD}}, \widetilde{PK_{PCD}}, D_{PICC})$, is generated as part of Terminal Authentication.

1. The MRTD chip sends its static Diffie-Hellman public key PK_{PICC}, and the domain parameters D_{PICC} to the terminal.
2. The terminal sends the ephemeral public key $\widetilde{PK_{PCD}}$ to the MRTD chip.
3. The MRTD chip computes the terminal's compressed ephemeral public key $\mathbf{Comp}(\widetilde{PK_{PCD}})$ and compares this to the compressed public key received in Terminal Authentication.
4. Both the MRTD chip and the terminal compute the following:
   a) The shared secret $K = \mathbf{KA}(SK_{PICC}, \widetilde{PK_{PCD}}, D_{PICC}) = \mathbf{KA}(\widetilde{SK_{PCD}}, PK_{PICC}, D_{PICC})$
5. The MRTD chip randomly chooses a nonce r_{PICC}, derives session keys $K_{MAC} = \mathbf{KDF_{MAC}}(K, r_{PICC})$ and $K_{Enc} = \mathbf{KDF_{Enc}}(K, r_{PICC})$ for Secure Messaging from K and r_{PICC}, computes the authentication token $T_{PICC} = \mathbf{MAC}(K_{MAC}, (\widetilde{PK_{PCD}}, D_{PICC}))$ and sends r_{PICC} and T_{PICC} to the terminal.
6. The terminal derives session keys $K_{MAC} = \mathbf{KDF_{MAC}}(K, r_{PICC})$ and $K_{Enc} = \mathbf{KDF_{Enc}}(K, r_{PICC})$ for Secure Messaging from K and r_{PICC} and verifies the authentication token T_{PICC}.

To verify the authenticity of the PK_{PICC} the terminal SHALL perform Passive Authentication.

MRTD Chip (PICC)		**Terminal (PCD)**
(CONDITIONAL: version 2)	$\left[\xleftarrow[A_{PCD}]{\mathbf{Comp}(\widetilde{PK}_{PCD})}\right]$	[Choose ephemeral key pair $(\widetilde{SK}_{PCD}, \widetilde{PK}_{PCD}, D_{PICC})$]
[choose r_{PICC} randomly]	$\xrightarrow{r_{PICC}}$	
	$\xleftarrow{s_{PCD}}$	$s_{PCD}=\mathbf{Sign}(SK_{PCD}, ID_{PICC}\|\|r_{PICC}\|\| \mathbf{Comp}(\widetilde{PK}_{PCD})\|\|A_{PCD})$
$\mathbf{Verify}(PK_{PCD}, s_{PCD}, ID_{PICC}\|\|r_{PICC}\|\| \mathbf{Comp}(\widetilde{PK}_{PCD})\|\|A_{PCD})$		

Figure 4.3: Terminal Authentication

4.3.2 Security Status

If Chip Authentication was successfully performed, Secure Messaging is restarted using the derived session keys K_{MAC} and K_{Enc}. Otherwise, Secure Messaging is continued using the previously established session keys (PACE or Basic Access Control).

Note: Passive Authentication MUST be performed . Only after a successful validation of the Security Object read from the MRTD chip using the new session keys the MRTD chip may be considered genuine.

4.4 Terminal Authentication

The Terminal Authentication Protocol is a two move challenge-response protocol that provides explicit unilateral authentication of the terminal.

In this protocol ID_{PICC} is an identifier of the MRTD chip:

- If BAC is used ID_{PICC} is the MRTD chip's Document Number as contained in the MRZ including the check digit.
- If PACE is used ID_{PICC} is computed using the MRTD chip's ephemeral PACE public key, i.e. $ID_{PICC}=\mathbf{Comp}(\widetilde{PK}_{PICC})$.

Note: All messages MUST be transmitted with Secure Messaging in Encrypt-then-Authenticate mode using session keys derived from PACE or Chip Authentication.

4.4.1 Protocol Specification

The following steps are performd by the terminal and the MRTD chip, a simplified version is also shown in Figure 4.3.

1. The terminal sends a certificate chain to the MRTD chip. The chain starts with a certificate verifiable with the CVCA public key stored on the chip and ends with the Terminal Certificate.

2. The MRTD chip verifies the certificates and extracts the terminal's public key PK_{PCD}.
3. Version 2 only:
   a) The terminal generates an ephemeral Diffie-Hellman key pair $(\widetilde{SK_{PCD}}, \widetilde{PK_{PCD}}, D_{PICC})$, and sends the compressed ephemeral public key $\mathbf{Comp}(\widetilde{PK_{PCD}})$ to the MRTD chip.

   b) The terminal may send auxiliary data A_{PCD} to the MRTD chip.
4. The MRTD chip randomly chooses a challenge r_{PICC} and sends it to the terminal.
5. The terminal responds with the signature
$$s_{PCD} = \mathbf{Sign}(SK_{PCD}, ID_{PICC} || r_{PICC} || \mathbf{Comp}(\widetilde{PK_{PCD}}) || A_{PCD}).$$
6. The MRTD chip checks that
$$\mathbf{Verify}(PK_{PCD}, s_{PCD}, ID_{PICC} || r_{PICC} || \mathbf{Comp}(\widetilde{PK_{PCD}}) || A_{PCD}) = \text{true}.$$

Note: In version 1 Chip Authentication MUST be performed before Terminal Authentication, i.e. $\mathbf{Comp}(\widetilde{PK_{PCD}})$ is calculated by both the MRTD chip and terminal as part of Chip Authentication. Since version 2 Chip Authentication may be performed after Terminal Authentication. In this case $\mathbf{Comp}(\widetilde{PK_{PCD}})$ MUST be calculated as part of Terminal Authentication. In addition the terminal MAY send authenticated auxiliary data A_{PCD} to the MRTD chip.

4.4.2 Security Status

If Terminal Authentication was successfully performed, the MRTD chip SHALL grant access to stored sensitive data according to the effective authorization level of the authenticated terminal. The MRTD chip SHALL however restrict the terminal's access rights to Secure Messaging established by the authenticated ephemeral public key, i.e. the MRTD chip SHALL compare the hash value of the terminal's ephemeral public key received as part of Terminal Authentication with the hash value of the ephemeral public key provided by the terminal as part of Chip Authentication. The MRTD chip MUST NOT accept more than one execution of Terminal Authentication within the same session.

Note: Secure Messaging is not affected by Terminal Authentication. The MRTD chip SHALL retain Secure Messaging even if Terminal Authentication fails (unless a Secure Messaging error occurs).

MRTD Chip (PICC)		**Terminal (PCD)**
unique chip identifier PK_{ID}		sector public key (PK_{Sector}, D)
	$\xleftarrow[D]{PK_{Sector}}$	
$I_{ID}^{Sector} = \mathbf{H}(\mathbf{KA}(SK_{ID}, PK_{Sector}, D))$	$\xrightarrow{I_{ID}^{Sector}}$	

Figure 4.4: Restricted Identification

4.5 Restricted Identification

The Restricted Identification Protocol is a static Diffie-Hellman key agreement protocol that generates a sector-specific identifier of the MRTD chip.

4.5.1 Protocol Specification

The following steps are performed by the terminal and the MRTD chip, a simplified version is also shown in Figure 4.4:

1. The terminal sends the static Sector Public Key PK_{Sector}, and the domain parameters D to the MRTD chip.
2. The MRTD chip verifies PK_{Sector}, computes and sends its sector-specific identifier $I_{ID}^{Sector} = \mathbf{H}(\mathbf{KA}(SK_{ID}, PK_{Sector}, D))$ to the terminal.
3. The terminal compares the received sector-specific identifier I_{ID}^{Sector} to the list of revoked sector identifiers received from the Document Verifier.

Restricted Identification SHALL only be used after Terminal Authentication *and* Chip Authentication have been successfully performed. Only then the authenticity of the sector public key PK_{Sector} and of the sector-specific identifier I_{ID}^{Sector} is guaranteed.

Note: The MRTD chip MUST verify the sector public key using the Terminal Sector extension (cf. Appendix C.3.2) contained in the Terminal Certificate.

4.5.2 Security Status

The MRTD chip's security status is not affected by Restricted Identification.

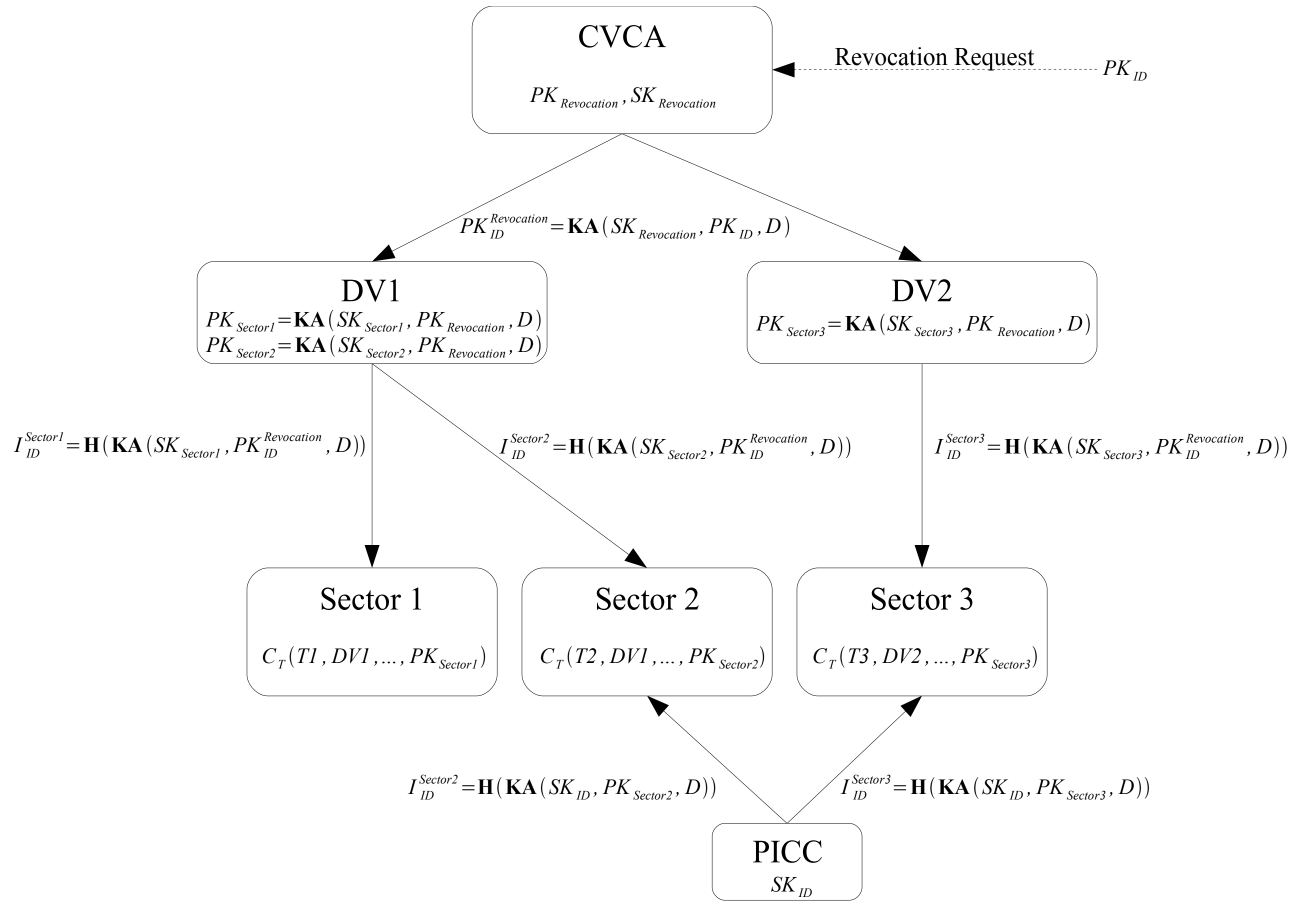
CVCA
$PK_{Revocation}, SK_{Revocation}$
Revocation Request
PK_{ID}
$PK_{ID}^{Revocation} = \mathbf{KA}(SK_{Revocation}, PK_{ID}, D)$
DV1
$PK_{Sector1} = \mathbf{KA}(SK_{Sector1}, PK_{Revocation}, D)$
$PK_{Sector2} = \mathbf{KA}(SK_{Sector2}, PK_{Revocation}, D)$
DV2
$PK_{Sector3} = \mathbf{KA}(SK_{Sector3}, PK_{Revocation}, D)$
$I_{ID}^{Sector1} = \mathbf{H}(\mathbf{KA}(SK_{Sector1}, PK_{ID}^{Revocation}, D))$
$I_{ID}^{Sector2} = \mathbf{H}(\mathbf{KA}(SK_{Sector2}, PK_{ID}^{Revocation}, D))$
$I_{ID}^{Sector3} = \mathbf{H}(\mathbf{KA}(SK_{Sector3}, PK_{ID}^{Revocation}, D))$
Sector 1
$C_T(T1, DV1, \ldots, PK_{Sector1})$
Sector 2
$C_T(T2, DV1, \ldots, PK_{Sector2})$
Sector 3
$C_T(T3, DV2, \ldots, PK_{Sector3})$
$I_{ID}^{Sector2} = \mathbf{H}(\mathbf{KA}(SK_{ID}, PK_{Sector2}, D))$
$I_{ID}^{Sector3} = \mathbf{H}(\mathbf{KA}(SK_{ID}, PK_{Sector3}, D))$
PICC
SK_{ID}

Figure 4.5: Revocation

4.5.3 Generation of Revocation Lists

The CVCA publishes the Revocation Sector Public Key $PK_{Revocation}$ and the domain parameters D. Each Document Verifier randomly chooses a Sector Private Key SK_{Sector} for every subordinated sector and calculates the Sector Public Key as $PK_{Sector} = \mathbf{KA}(SK_{Sector}, PK_{Revocation}, D)$.

To revoke an MRTD chip a revocation request is sent to the CVCA containing the Restricted Identification Public Key PK_{ID}. The sector-specific identities are calculates as follows:

1. The CVCA calculates $PK_{ID}^{Revocation} = \mathbf{KA}(SK_{Revocation}, PK_{ID}, D)$ using its private key $SK_{Revocation}$ and the Restricted Identification Public Key PK_{ID} received with the revocation request. The transformed public key $PK_{ID}^{Revocation}$ is forwarded to all subordinated Document Verifiers.
2. Each Document Verifier calculates the sector-specific identifier for all subordinated sectors. For each sector the Document Verifier calculates

$$I_{ID}^{Sector} = \mathbf{H}(\mathbf{KA}(SK_{Sector}, PK_{ID}^{Revocation}, D))$$

   using the corresponding Sector Private Key SK_{Sector} and the received public key $PK_{ID}^{Revocation}$ of the MRTD chip to be revoked. The sector-specific identifier I_{ID}^{Sector} is then forwarded to the terminals of the corresponding sector.

A. ASN.1 Specifications (Normative)

The object identifiers used in the following appendices are contained in the subtree of `bsi-de`:

```
bsi-de OBJECT IDENTIFIER ::= {
  itu-t(0) identified-organization(4) etsi(0)
  reserved(127) etsi-identified-organization(0) 7
}
```

A.1. Information on Supported Security Protocols

The ASN.1 data structure `SecurityInfos` SHALL be provided by the MRTD chip to indicate supported security protocols. The data structure is specified as follows:

```
SecurityInfos ::= SET OF SecurityInfo

SecurityInfo  ::= SEQUENCE {
  protocol      OBJECT IDENTIFIER,
  requiredData  ANY DEFINED BY protocol,
  optionalData  ANY DEFINED BY protocol OPTIONAL
}
```

The elements contained in a `SecurityInfo` data structure have the following meaning:

- The object identifier `protocol` identifies the supported protocol.
- The open type `requiredData` contains protocol specific mandatory data.
- The open type `optionalData` contains protocol specific optional data.

A.1.1. Supported Protocols

The ASN.1 specifications for the protocols provided in this specification are described in the following.

Note: MRTD chips implemented according to Version 1.0.x of this specification will only provide a `ChipAuthenticationPublicKeyInfo`.

In this case the terminal SHOULD assume the following:

- The MRTD chip supports Chip Authentication in version 1.
- The MRTD chip may support Terminal Authentication in version 1.

To determine whether or not sensitive data protected by Terminal Authentication is stored on the MRTD chip, the terminal may consult the Security Object and the elementary file EF.CVCA.

A.1.1.1. PACE

To indicate support for PACE `SecurityInfos` may contain the following entries:

- At least one `PACEInfo` MUST be present.
- At least one `PACEDomainParameterInfo` MUST be present.

PACEInfo: This data structure provides detailed information on an implementation of PACE.

- The object identifier `protocol` SHALL identify the algorithms to be used (i.e. key agreement, symmetric cipher and MAC).
- The integer `version` SHALL identify the version of the protocol. Currently, only version 1 is supported.
- The integer `parameterId` MAY be used to indicate the local domain parameter identifier. It MUST be used if the MRTD chip provides multiple domain parameters for PACE.

```
id-PACE OBJECT IDENTIFIER ::= {
  bsi-de protocols(2) smartcard(2) 4
}

id-PACE-DH-GM                     OBJECT IDENTIFIER ::= {id-PACE 1}
id-PACE-DH-GM-3DES-CBC-CBC        OBJECT IDENTIFIER ::= {id-PACE-DH-GM 1}
id-PACE-DH-GM-AES-CBC-CMAC-128    OBJECT IDENTIFIER ::= {id-PACE-DH-GM 2}
id-PACE-DH-GM-AES-CBC-CMAC-192    OBJECT IDENTIFIER ::= {id-PACE-DH-GM 3}
id-PACE-DH-GM-AES-CBC-CMAC-256    OBJECT IDENTIFIER ::= {id-PACE-DH-GM 4}

id-PACE-ECDH-GM                   OBJECT IDENTIFIER ::= {id-PACE 2}
id-PACE-ECDH-GM-3DES-CBC-CBC      OBJECT IDENTIFIER ::= {id-PACE-ECDH-GM 1}
id-PACE-ECDH-GM-AES-CBC-CMAC-128 OBJECT IDENTIFIER ::= {id-PACE-ECDH-GM 2}
id-PACE-ECDH-GM-AES-CBC-CMAC-192 OBJECT IDENTIFIER ::= {id-PACE-ECDH-GM 3}
id-PACE-ECDH-GM-AES-CBC-CMAC-256 OBJECT IDENTIFIER ::= {id-PACE-ECDH-GM 4}

id-PACE-DH-IM                     OBJECT IDENTIFIER ::= {id-PACE 3}
id-PACE-DH-IM-3DES-CBC-CBC        OBJECT IDENTIFIER ::= {id-PACE-DH-IM 1}
id-PACE-DH-IM-AES-CBC-CMAC-128    OBJECT IDENTIFIER ::= {id-PACE-DH-IM 2}
id-PACE-DH-IM-AES-CBC-CMAC-192    OBJECT IDENTIFIER ::= {id-PACE-DH-IM 3}
id-PACE-DH-IM-AES-CBC-CMAC-256    OBJECT IDENTIFIER ::= {id-PACE-DH-IM 4}

id-PACE-ECDH-IM                   OBJECT IDENTIFIER ::= {id-PACE 4}
id-PACE-ECDH-IM-3DES-CBC-CBC      OBJECT IDENTIFIER ::= {id-PACE-ECDH-IM 1}
id-PACE-ECDH-IM-AES-CBC-CMAC-128 OBJECT IDENTIFIER ::= {id-PACE-ECDH-IM 2}
id-PACE-ECDH-IM-AES-CBC-CMAC-192 OBJECT IDENTIFIER ::= {id-PACE-ECDH-IM 3}
id-PACE-ECDH-IM-AES-CBC-CMAC-256 OBJECT IDENTIFIER ::= {id-PACE-ECDH-IM 4}

PACEInfo ::= SEQUENCE {
  protocol     OBJECT IDENTIFIER(
               id-PACE-DH-3DES-CBC-CBC |
               id-PACE-DH-AES-CBC-CMAC-128 |
               id-PACE-DH-AES-CBC-CMAC-192 |
               id-PACE-DH-AES-CBC-CMAC-256 |
               id-PACE-ECDH-3DES-CBC-CBC |
               id-PACE-ECDH-AES-CBC-CMAC-128 |
               id-PACE-ECDH-AES-CBC-CMAC-192 |
               id-PACE-ECDH-AES-CBC-CMAC-256 |
               id-PACE-DH-IM-3DES-CBC-CBC |
               id-PACE-DH-IM-AES-CBC-CMAC-128 |
               id-PACE-DH-IM-AES-CBC-CMAC-192 |
               id-PACE-DH-IM-AES-CBC-CMAC-256 |
               id-PACE-ECDH-IM-3DES-CBC-CBC |
               id-PACE-ECDH-IM-AES-CBC-CMAC-128 |
               id-PACE-ECDH-IM-AES-CBC-CMAC-192 |
               id-PACE-ECDH-IM-AES-CBC-CMAC-256),
  version      INTEGER, -- MUST be 1
  parameterId INTEGER OPTIONAL
}
```

PACEDomainParameterInfo: This data structure provides one set of domain parameters for PACE of the MRTD chip.

- The object identifier `protocol` SHALL identify the type of the domain parameters (i.e. DH or ECDH).
- The sequence `domainParameter` SHALL contain the domain parameters.
- The integer `parameterId` MAY be used to indicate the local domain parameter identifier. It MUST be used if the MRTD chip provides multiple domain parameters for PACE.

```
PACEDomainParameterInfo ::= SEQUENCE {
  protocol        OBJECT IDENTIFIER(id-PACE-DH | id-PACE-ECDH),
  domainParameter AlgorithmIdentifier,
  parameterId     INTEGER OPTIONAL
}
```

A.1.1.2. Chip Authentication

To indicate support for Chip Authentication `SecurityInfos` may contain the following entries:

- At least one `ChipAuthenticationPublicKeyInfo` MUST be present.
- At least one `ChipAuthenticationInfo` MUST be present.
- At least one `ChipAuthenticationDomainParameterInfo` MUST be present for Chip Authentication in version 2.

If more than one Chip Authentication Public Key is present the optional `keyId` MUST be used in all three data structures to indicate the local key identifier. All public keys MUST have different domain parameters.

ChipAuthenticationInfo: This data structure provides detailed information on an implementation of Chip Authentication.

- The object identifier `protocol` SHALL identify the algorithms to be used (i.e. key agreement, symmetric cipher and MAC).
- The integer `version` SHALL identify the version of the protocol. Currently, versions 1 and 2 are supported.
- The integer `keyId` MAY be used to indicate the local key identifier. It MUST be used if the MRTD chip provides multiple public keys for Chip Authentication.

```
id-CA OBJECT IDENTIFIER ::= {
  bsi-de protocols(2) smartcard(2) 3
}

id-CA-DH                     OBJECT IDENTIFIER ::= {id-CA 1}
id-CA-DH-3DES-CBC-CBC        OBJECT IDENTIFIER ::= {id-CA-DH 1}
id-CA-DH-AES-CBC-CMAC-128    OBJECT IDENTIFIER ::= {id-CA-DH 2}
id-CA-DH-AES-CBC-CMAC-192    OBJECT IDENTIFIER ::= {id-CA-DH 3}
id-CA-DH-AES-CBC-CMAC-256    OBJECT IDENTIFIER ::= {id-CA-DH 4}
```

```
id-CA-ECDH                    OBJECT IDENTIFIER ::= {id-CA 2}
id-CA-ECDH-3DES-CBC-CBC       OBJECT IDENTIFIER ::= {id-CA-ECDH 1}
id-CA-ECDH-AES-CBC-CMAC-128 OBJECT IDENTIFIER ::= {id-CA-ECDH 2}
id-CA-ECDH-AES-CBC-CMAC-192 OBJECT IDENTIFIER ::= {id-CA-ECDH 3}
id-CA-ECDH-AES-CBC-CMAC-256 OBJECT IDENTIFIER ::= {id-CA-ECDH 4}

ChipAuthenticationInfo ::= SEQUENCE {
protocol   OBJECT IDENTIFIER(
           id-CA-DH-3DES-CBC-CBC |
           id-CA-DH-AES-CBC-CMAC-128 |
           id-CA-DH-AES-CBC-CMAC-192 |
           id-CA-DH-AES-CBC-CMAC-256 |
           id-CA-ECDH-3DES-CBC-CBC |
           id-CA-ECDH-AES-CBC-CMAC-128 |
           id-CA-ECDH-AES-CBC-CMAC-192 |
           id-CA-ECDH-AES-CBC-CMAC-256),
  version  INTEGER, -- MUST be 1 or 2
  keyId    INTEGER OPTIONAL
}
```

ChipAuthenticationDomainParameterInfo: This data structure provides one set of domain parameters for Chip Authentication of the MRTD chip.

- The object identifier `protocol` SHALL identify the type of the domain parameters (i.e. DH or ECDH).
- The sequence `domainParameter` SHALL contain the domain parameters.
- The integer `keyId` MAY be used to indicate the local key identifier. It MUST be used if the MRTD chip provides multiple public keys for Chip Authentication.

```
ChipAuthenticationDomainParameterInfo ::= SEQUENCE {
  protocol        OBJECT IDENTIFIER(id-CA-DH | id-CA-ECDH),
  domainParameter AlgorithmIdentifier,
  keyId           INTEGER OPTIONAL
}
```

ChipAuthenticationPublicKeyInfo: This data structure provides an Chip Authentication Public Key of the MRTD chip.

- The object identifier `protocol` SHALL identify the type of the public key (i.e. DH or ECDH).
- The sequence `chipAuthenticationPublicKey` SHALL contain the public key in encoded form.
- The integer `keyId` MAY be used to indicate the local key identifier. It MUST be used if the MRTD chip provides multiple public keys for Chip Authentication.

```
id-PK OBJECT IDENTIFIER ::= {
  bsi-de protocols(2) smartcard(2) 1
}

id-PK-DH                 OBJECT IDENTIFIER ::= {id-PK 1}
id-PK-ECDH               OBJECT IDENTIFIER ::= {id-PK 2}

ChipAuthenticationPublicKeyInfo ::= SEQUENCE {
  protocol                    OBJECT IDENTIFIER(id-PK-DH | id-PK-ECDH),
  chipAuthenticationPublicKey SubjectPublicKeyInfo,
  keyId                            INTEGER OPTIONAL
}
```

A.1.1.3. Terminal Authentication

To indicate support for Terminal Authentication `SecurityInfos` may contain the following entry:

- At least one `TerminalAuthenticationInfo` SHOULD be present.

TerminalAuthenticationInfo: This data structure provides detailed information on an implementation of Terminal Authentication.

- The object identifier `protocol` SHALL identify the *general* Terminal Authentication Protocol as the specific protocol may change over time.
- The integer `version` SHALL identify the version of the protocol. Currently, versions 1 and 2 are supported.
- The sequence `efCVCA` MAY be used to indicate a (short) file identifier of the file EF.CVCA. It MUST be used, if the default (short) file identifier is not used.

```
id-TA OBJECT IDENTIFIER ::= {
  bsi-de protocols(2) smartcard(2) 2
}

id-TA-RSA                OBJECT IDENTIFIER ::= {id-TA 1}
id-TA-RSA-v1-5-SHA-1     OBJECT IDENTIFIER ::= {id-TA-RSA 1}
id-TA-RSA-v1-5-SHA-256   OBJECT IDENTIFIER ::= {id-TA-RSA 2}
id-TA-RSA-PSS-SHA-1      OBJECT IDENTIFIER ::= {id-TA-RSA 3}
id-TA-RSA-PSS-SHA-256    OBJECT IDENTIFIER ::= {id-TA-RSA 4}

id-TA-ECDSA              OBJECT IDENTIFIER ::= {id-TA 2}
id-TA-ECDSA-SHA-1        OBJECT IDENTIFIER ::= {id-TA-ECDSA 1}
id-TA-ECDSA-SHA-224      OBJECT IDENTIFIER ::= {id-TA-ECDSA 2}
id-TA-ECDSA-SHA-256      OBJECT IDENTIFIER ::= {id-TA-ECDSA 3}

TerminalAuthenticationInfo ::= SEQUENCE {
  protocol OBJECT IDENTIFIER(id-TA),
  version  INTEGER, -- MUST be 1 or 2
  efCVCA   FileID OPTIONAL -- MUST NOT be used for version 2
}

FileID ::= SEQUENCE {
  fid  OCTET STRING (SIZE(2)),
  sfid OCTET STRING (SIZE(1)) OPTIONAL
}
```

A.1.1.4. Restricted Identification

To indicate support for Restricted Identification `SecurityInfos` may contain the following entry:

- At least one `RestrictedIdentificationInfo` MUST be present.
- Exactly one `RestrictedIdentificationDomainParameterInfo` MAY be present.

RestrictedIdentificationInfo: This data structure provides detailed information on an implementation of Restricted Identification.

- The object identifier `protocol` SHALL identify the algorithms to be used (i.e. key agreement).
- The integer `version` SHALL identify the version of the protocol. Currently, only version 1 is supported.
- The integer `keyId` SHALL identify the private key to be used.
- The boolean `authorizedOnly` SHALL indicate whether explicit authorization is REQUIRED to use the corresponding secret key.
- The integer `maxKeyLen` MAY be used to indicate the maximum length of the supported sector specific public keys.

```
id-RI OBJECT IDENTIFIER ::= {
  bsi-de protocols(2) smartcard(2) 5
}

id-RI-DH            OBJECT IDENTIFIER ::= {id-RI 1}
id-RI-DH-SHA-1      OBJECT IDENTIFIER ::= {id-RI-DH 1}
id-RI-DH-SHA-224    OBJECT IDENTIFIER ::= {id-RI-DH 2}
id-RI-DH-SHA-256    OBJECT IDENTIFIER ::= {id-RI-DH 3}

id-RI-ECDH          OBJECT IDENTIFIER ::= {id-RI 2}
id-RI-ECDH-SHA-1    OBJECT IDENTIFIER ::= {id-RI-ECDH 1}
id-RI-ECDH-SHA-224  OBJECT IDENTIFIER ::= {id-RI-ECDH 2}
id-RI-ECDH-SHA-256  OBJECT IDENTIFIER ::= {id-RI-ECDH 3}

RestrictedIdentificationInfo ::= SEQUENCE {
  protocol  OBJECT IDENTIFIER(
            id-RI-DH-SHA-1 |
            id-RI-DH-SHA-224 |
            id-RI-DH-SHA-256 |
            id-RI-ECDH-SHA-1 |
            id-RI-ECDH-SHA-224 |
            id-RI-ECDH-SHA-256),
  params    ProtocolParams,
  maxKeyLen INTEGER OPTIONAL
}

ProtocolParams ::= SEQUENCE {
  version         INTEGER, -- MUST be 1
  keyId           INTEGER,
  authorizedOnly  BOOLEAN
}
```

RestrictedIdentificationDomainParameterInfo: This data structure provides the set of domain parameters that have been used for the generation of the Restricted Identification Public Key PK_{ID} for revocation of the MRTD chip.

- The object identifier `protocol` SHALL identify the type of the domain parameters (i.e. DH or ECDH).
- The sequence `domainParameter` SHALL contain the domain parameters.

```
RestrictedIdentificationDomainParameterInfo ::= SEQUENCE {
    protocol        OBJECT IDENTIFIER(id-RI-DH | id-RI-ECDH),
    domainParameter AlgorithmIdentifier
  }
```

A.1.1.5. CardInfoLocator

To provide information about card capabilities and the structure of the card `SecurityInfos` may contain the following entry:

- Exactly one `CardInfoLocator` SHOULD be present.

CardInfoLocator: This data structure provides detailed information where to retrieve the CardInfo file [3].

- The STRING `url` SHALL define the location that provides the most recent CardInfo file.
- The sequence `efCardInfo` MAY be used to indicate a (short) file identifier of the file EF.-CardInfo.

```
id-CI OBJECT IDENTIFIER ::= {
  bsi-de protocols(2) smartcard(2) 6
}

CardInfoLocator ::= SEQUENCE {
  protocol   OBJECT IDENTIFIER(id-CI),
  url        IA5String,
  efCardInfo FileID OPTIONAL
}

FileID ::= SEQUENCE {
  fid  OCTET STRING (SIZE(2)),
  sfid OCTET STRING (SIZE(1)) OPTIONAL
}
```

A.1.1.6. Other Protocols

`SecurityInfos` MAY contain references to protocols that are not contained in this specification (including Active Authentication and Basic Access Control).

File Name	**EF.CardAccess**	**EF.CardSecurity**
File ID	0x011C	0x011D
Short File ID	0x1C	0x1D
Read Access	ALWAYS	PACE + TA
Write Access	NEVER	NEVER
Size	variable	variable
Content	DER encoded SecurityInfo	DER encoded SignedData

Table A.1: Elementary Files CardAccess and CardSecurity

A.1.2. Storage on the Chip

The MRTD chip SHALL provide `SecurityInfos` in two transparent elementary files *CardAccess* and *CardSecurity* contained in the master file (cf. Table A.1):

- The file CardAccess SHALL contain the relevant `SecurityInfos` that are required to access applications:
  - `PACEInfo`
  - `PACEDomainParameterInfo`
  - `ChipAuthenticationInfo`
  - `ChipAuthenticationDomainParameterInfo`
  - `TerminalAuthenticationInfo`
  - `CardInfoLocator`
- The file CardSecurity SHALL contain the full set of signed `SecurityInfos`.

Note: If compatibility to version 1.x of this specification is required, the MRTD chip SHALL also provide `SecurityInfos` in data group DG14 of the ePassport application.

The file CardSecurity SHALL be implemented as `SignedData` according to RFC 3852 [6] with content type `SecurityInfos`. The Card Security Object SHALL be signed by the Document Signer. The Document Signer Certificate MUST be included in `SignedData`. The following Object Identifier SHALL be used to identify the content type:

```
id-SecurityObject OBJECT IDENTIFIER ::= {
  bsi-de applications(3) eID(2) 1
}
```

The data structure `SignedData` is defined as follows; more details can be found in [6]:

```
SignedData ::= SEQUENCE{
  version CMSVersion,
  digestAlgorithms DigestAlgoritmsIdentifiers,
  encapContentInfo EncapsulatedContentInfo,
  certificates [0] IMPLICIT CertificateSet OPTIONAL,
  crls [1] IMPLICIT RevocationInfoChoices OPTIONAL
  signerInfos SignerInfos
}
```

Algorithm / Format	**DH**	**ECDH**
Key Agreement Algorithm	PKCS#3 [24]	ECKA [4]
X.509 Public Key Format	X9.42 [1]	ECC [4]
TLV Public Key Format	TLV, cf. Appendix D.3.2	TLV, cf. Appendix D.3.3
Public Key Compression	SHA-1 [21]	X-Coordinate
Ephemeral Public Key Validation	RFC 2631 [23]	ECC [4]

Table A.2: Algorithms and Formats for Key Agreement

```
DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

EncapsulatedContentInfo ::= SEQUENCE {
  eContentType ContentType,
  eContent [0] EXPLICIT OCTET STRING OPTIONAL
}

ContentType ::= OBJECT IDENTIFIER

SignerInfos ::= SET OF SignerInfo

SignerInfo ::= SEQUENCE {
  version CMSVersion,
  sid SignerIdentifier,
  digestAlgorithm DigestAlgorithmIdentifier,
  signatureAlgoritm SignatureAlgorithmIdentifier,
  signature SignatureValue
}

SignerIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  subjectKeyIdentifier [0] SubjectKeyIdentifier
}

SignatureValue ::= OCTET STRING
```

A.2. Key Agreement

PACE, Chip Authentication, and Restricted Identification are based on key agreement protocols. This appendix specifies the general algorithms, formats and protocols.

A.2.1. Domain Parameters

A.2.1.1. PACE and Chip Authentication

The domain parameters for PACE and Chip Authentication MUST be provided as `AlgorithmIdentifier` in the structures `PACEDomainParameterInfo` and `ChipAuthenticationDomainParameterInfo`, respectively.

The data structure `AlgorithmIdentifier` is defined as follows; more details can be found in [7]:

```
AlgorithmIdentifier ::= SEQUENCE {
  algorithm  OBJECT IDENTIFIER,
  parameters ANY DEFINED BY algorithm OPTIONAL
}
```

The MRTD chip MAY support more than one set of domain parameters (i.e. the chip may support different algorithms and/or key lengths). In this case the local identifier MUST be disclosed in the corresponding `PACEDomainParameterInfo` and `ChipAuthenticationDomainparameterInfo`.

A.2.1.2. Restricted Identification

The domain parameters for Restricted Identification are defined by the Document Verifier and MUST be provided together with the sector public key in a public key data object as part of Restricted Identification (cf. Appendix D.3 and Appendix B.4.1). The hash of this public key data object MUST be contained in the Terminal Certificate as Terminal Sector extension (cf. Appendix C.3.2). The MRTD chip MUST verify the sector public key using the Terminal Sector extension.

A.2.2. Ephemeral Public Keys

A.2.2.1. PACE and Chip Authentication

The domain parameters contained in `PACEDomainParameterInfo` and `ChipAuthenticationDomainParameterInfo` or `ChipAuthenticationPublicKeyInfo` MUST be used by the terminal for the generation of an ephemeral public key for PACE and Chip Authentication, respectively. Ephemeral public keys MUST be exchanged as plain public key values. More information on the encoding can be found in Appendix D.3.4.

Note: According to Section 4.1.3 the validation of ephemeral public keys is RECOMMENDED. For DH, the validation algorithm requires the MRTD chip to have a more detailed knowledge of the domain parameters (i.e. the order of the used subgroup) than usually provided by PKCS#3.

A.2.2.2. Restricted Identification

For Restricted Identification ephemeral public keys are not used.

Password	**Encoding**
MRZ	SHA-1(Serial Number || Date of Birth || Date of Expiry)
CAN	Character String (cf. Appendix D.2.1.4)
PIN	Character String (cf. Appendix D.2.1.4)
PUK	Character String (cf. Appendix D.2.1.4)

Table A.3: Encoding of Passwords

A.2.3. Key Derivation Function

Let $\mathbf{KDF}_{\mathbf{Enc}}(K,[r])=\mathbf{KDF}(K,[r],1)$, $\mathbf{KDF}_{\mathbf{MAC}}(K,[r])=\mathbf{KDF}(K,[r],2)$, be key derivation functions to derive encryption and authentication keys, respectively, from a shared secret K and an optional nonce r. Let $\mathbf{KDF}_{\pi}(\pi)=\mathbf{KDF}(f(\pi),3)$, be a key derivation function to derive encryption keys from a password π. The encoding of passwords, i.e. $K=f(\pi)$ is specified in Table A.3.

The key derivation function $\mathbf{KDF}(K,[r],c)$, is defined as follows:

Input: The following inputs are required:

- The shared secret value K **(REQUIRED)**
- A nonce r **(OPTIONAL)**
- A 32-bit, big-endian integer counter c **(REQUIRED)**

Output: An octet string keydata.

Actions: The following actions are performed:

1. $\text{keydata}=\mathbf{H}(K\|r\|c)$
2. Output octet string keydata

The key derivation function $\mathbf{KDF}(K,[r],c)$ requires a suitable hash function denoted by $\mathbf{H}()$, i.e the bit-length of the hash function SHALL be greater or equal to the bit-length of the derived key. The hash value SHALL be interpreted as big-endian byte output.

Note: The shared secret K is defined as an octet string. If the shared secret is generated with ECKA [4], the x-coordinate of the generated point SHALL be used.

A.2.3.1. 3DES

To derive 112-bit 3DES [19] keys the hash function SHA-1 [21] SHALL be used and the following additional steps MUST be performed:

- Use octets 1 to 8 of keydata to form keydataA and octets 9 to 16 of keydata to form keydataB; additional octets are not used.
- Adjust the parity bits of keydataA and keydataB to form correct DES keys (OPTIONAL).

A.2.3.2. AES

To derive 128-bit AES [20] keys the hash function SHA-1 [21] SHALL be used and the following additional step MUST be performed:

- Use octets 1 to 16 of keydata; additional octets are not used.

To derive 192-bit and 256-bit AES [20] keys SHA-256 [21] SHALL be used. For 192-bit AES keys the following additional step MUST be performed:

- Use octets 1 to 24 of keydata; additional octets are not used.

A.2.4. Authentication Token

The authentication token used in PACE and Chip Authentication in version 2 SHALL be computed over a public key data object (cf. Appendix D.3) containing the object identifier of the protocol used, i.e. PACE or Chip Authentication (as indicated in MSE:Set AT, cf. Appendix B.11.1,), the received ephemeral public key and the corresponding domain parameters using a message authentication and the key K_{MAC} derived from the key agreement.

Note: The domain parameters MUST be converted from an `AlgorithmIdentifier` contained in the structures `PACEDomainParameterInfo` and `ChipAuthenticationDomainParameterInfo` to the format specified in (cf. Appendix D.3). Data objects that are OPTIONAL in the public key data object SHALL be converted as indicated in the `AlgorithmIdentifier`.

A.2.4.1. 3DES

3DES [19] SHALL be used in Retail-mode according to ISO/IEC 9797-1 [16] MAC algorithm 3 / padding method 2 with block cipher DES and $IV=0$.

A.2.4.2. AES

AES [20] SHALL be used in CMAC-mode [22] with a MAC length of 8 bytes.

A.3. PACE

A.3.1. PACE with DH

For PACE with DH the respective algorithms and formats from Table A.1.2 and Table A.4 MUST be used.

OID	**Mapping**	**Sym. Cipher**	**Key Len**	**Secure Messaging**	**Auth. Token**
id-PACE-DH-GM-3DES-CBC-CBC	Generic	3DES	112	CBC / CBC	CBC
id-PACE-DH-GM-AES-CBC-CMAC-128	Generic	AES	128	CBC / CMAC	CMAC
id-PACE-DH-GM-AES-CBC-CMAC-192	Generic	AES	192	CBC / CMAC	CMAC
id-PACE-DH-GM-AES-CBC-CMAC-256	Generic	AES	256	CBC / CMAC	CMAC
id-PACE-DH-IM-3DES-CBC-CBC	Integrated	3DES	112	CBC / CBC	CBC
id-PACE-DH-IM-AES-CBC-CMAC-128	Integrated	AES	128	CBC / CMAC	CMAC
id-PACE-DH-IM-AES-CBC-CMAC-192	Integrated	AES	192	CBC / CMAC	CMAC
id-PACE-DH-IM-AES-CBC-CMAC-256	Integrated	AES	256	CBC / CMAC	CMAC

Table A.4: Object Identifiers for PACE with DH

A.3.2. PACE with ECDH

For PACE with ECDH the respective algorithms and formats from Table A.1.2 and Table A.5 MUST be used.

OID	**Mapping**	**Sym. Cipher**	**Key Len**	**Secure Messaging**	**Auth. Token**
id-PACE-ECDH-GM-3DES-CBC-CBC	Generic	3DES	112	CBC / CBC	CBC
id-PACE-ECDH-GM-AES-CBC-CMAC-128	Generic	AES	128	CBC / CMAC	CMAC
id-PACE-ECDH-GM-AES-CBC-CMAC-192	Generic	AES	192	CBC / CMAC	CMAC
id-PACE-ECDH-GM-AES-CBC-CMAC-256	Generic	AES	256	CBC / CMAC	CMAC
id-PACE-ECDH-IM-3DES-CBC-CBC	Integrated	3DES	112	CBC / CBC	CBC
id-PACE-ECDH-IM-AES-CBC-CMAC-128	Integrated	AES	128	CBC / CMAC	CMAC
id-PACE-ECDH-IM-AES-CBC-CMAC-192	Integrated	AES	192	CBC / CMAC	CMAC
id-PACE-ECDH-IM-AES-CBC-CMAC-256	Integrated	AES	256	CBC / CMAC	CMAC

Table A.5: Object Identifiers for PACE with ECDH

A.3.3. Encrypted Nonce

The MRTD chip SHALL randomly and uniformly select the nonce $s \in_R \{0 \ldots 2^k - 1\}$ as binary bit string of length k, where k is the blocksize of the used cipher, i.e. $k = 64$ for 3DES and $k = 128$ for AES.

- The nonce s SHALL be encrypted in ECB mode according to ISO 10116 [12] using the key $K_\pi = \mathbf{KDF}_\pi(\pi)$ derived from the password π.
- The nonce s SHALL be converted to a random generator using an algorithm-specific mapping function **Map**.

Note: Several different algorithms exist for implementing the mapping of the nonce to ephemeral domain parameters. Currently, all specified mappings implementing $\tilde{D} = \mathbf{Map}(D_{PICC}, s)$ map the nonce to an ephemeral generator. It is RECOMMENDED to implement the mapping as a randomized function.

A.3.4. ECDH Mapping

Let G and $\tilde{G}$ be the static and an ephemeral base point on the elliptic curve.

A.3.4.1. Generic Mapping

The function $\mathbf{Map}: G \mapsto \tilde{G}$ is defined as $\tilde{G} = s \cdot G + H$, where $H \in \langle G \rangle$ is chosen s.th. $\log_G H$ is unknown. The point H SHALL be calculated by an anonymous Diffie-Hellman Key Agreement [4].

Note: The key agreement algorithm ECKA prevents small subgroup attacks by using compatible cofactor multiplication.

A.3.4.2. Integrated Mapping

The function $\mathbf{Map}: G \mapsto \tilde{G}$ is defined as $\tilde{G} = f(\mathbf{E}(K, s))$, where $f()$ is a function that maps octet strings to points on the elliptic curve. The symmetric key K SHALL be chosen randomly by the terminal and sent to the MRTD chip.

The function $f()$ is defined in [26].

A.3.5. DH Mapping

Let g and $\tilde{g}$ be the static and an ephemeral generator.

A.3.5.1. Generic Mapping

The function $\mathbf{Map}: g \mapsto \tilde{g}$ is defined as $\tilde{g} = g^s \cdot h$, where $h \in \langle g \rangle$ is chosen s.th. $\log_g h$ is unknown. The group element h SHALL be calculated by an anonymous Diffie-Hellman Key Agreement.

Note: The public key validation method described in RFC 2631 [23] MUST be used to prevent small subgroup attacks.

A.3.5.2. Integrated Mapping

The function **Map**$: g \mapsto \tilde{g}$ is defined as $\tilde{g} = f(\mathbf{E}(K, s))$, where $f()$ is a function that maps octet strings to elements of $GF(p)$. The symmetric key K SHALL be chosen randomly by the terminal and sent to the MRTD chip.

The function $f()$ is defined as $f(x) = \mathrm{int}(x)^a \bmod p$, where $\mathrm{int}()$ is the conversion of the octet string to an unsigned integer in big-endian format using the binary representation and $a = (p-1)/q$ is the cofactor. Implementations MUST check that $\tilde{g} \neq 1$.

A.4. Chip Authentication

A.4.1. Chip Authentication Key Pair

The Chip Authentication Key Pair(s) MUST be stored on the MRTD chip.

- The private key SHALL be stored securely in the MRTD chip's memory.
- The public key SHALL be provided as `SubjectPublicKeyInfo` in the `ChipAuthenticationPublicKeyInfo` structure.
- The domain parameters MAY be additionally provided as `AlgorithmIdentifier` in the `ChipAuthenticationDomainParameterInfo` structure.

The data structures `SubjectPublicKeyInfo` and `AlgorithmIdentifier` are defined as follows; more details can be found in [7]:

```
SubjectPublicKeyInfo ::= SEQUENCE {
  algorithm        AlgorithmIdentifier,
  subjectPublicKey BIT STRING
}

AlgorithmIdentifier ::= SEQUENCE {
  algorithm  OBJECT IDENTIFIER,
  parameters ANY DEFINED BY algorithm OPTIONAL
}
```

The MRTD chip MAY support more than one Chip Authentication Key Pair (i.e. the chip may support different algorithms and/or key lengths). In this case the local key identifier MUST be disclosed in the corresponding `ChipAuthenticationInfo`, `ChipAuthenticationPublicKeyInfo`, and `ChipAuthenticationDomainParameterInfo`.

A.4.2. Chip Authentication with DH

For Chip Authentication with DH the respective algorithms and formats from Table A.1.2 and Table A.6 MUST be used. For Chip Authentication in version 1 PKCS#3 [24] MUST be used instead of X9.42 [1].

OID	**Sym. Cipher**	**Key Length**	**Secure Messaging**	**Auth. Token**
id-CA-DH-3DES-CBC-CBC	3DES	112	CBC / CBC	CBC
id-CA-DH-AES-CBC-CMAC-128	AES	128	CBC / CMAC	CMAC
id-CA-DH-AES-CBC-CMAC-192	AES	192	CBC / CMAC	CMAC
id-CA-DH-AES-CBC-CMAC-256	AES	256	CBC / CMAC	CMAC

Table A.6: Object Identifiers for Chip Authentication with DH

A.4.3. Chip Authentication with ECDH

For Chip Authentication with ECDH the respective algorithms and formats from Table A.1.2 and Table A.7 MUST be used. The elliptic curve domain parameters MUST be described explicitly in the `ECParameters` structure, i.e. named curves and implicit domain parameters MUST NOT be used.

OID	**Sym. Cipher**	**Key Length**	**Secure Messaging**	**Auth. Token**
id-CA-ECDH-3DES-CBC-CBC	3DES	112	CBC / CBC	CBC
id-CA-ECDH-AES-CBC-CMAC-128	AES	128	CBC / CMAC	CMAC
id-CA-ECDH-AES-CBC-CMAC-192	AES	192	CBC / CMAC	CMAC
id-CA-ECDH-AES-CBC-CMAC-256	AES	256	CBC / CMAC	CMAC

Table A.7: Object Identifiers for Chip Authentication with ECDH

A.5. Restricted Identification

A.5.1. MRTD Chip Private Key

The generation of the chip private key SK_{ID} is out of the scope of this specification. It is however RECOMMENDED to generate SK_{ID} as encrypted sequential counter or as encrypted document number. In this case the secret encryption key SHALL be generated and stored securely by a third party.

A.5.2. Sector Public Keys

The sector public keys MUST be generated by a (trusted) third party.

- If the third party MUST be able to link sector-specific identifier across sectors, then the third party SHALL generate sector key pairs and store the sector private keys securely.
- If the third party MUST NOT be able to link sector-specific identifier across sectors, then the third party SHALL generate sector public keys in a way that the corresponding private keys are unknown.

A.5.3. Restricted Identification with DH

For Restricted Identification with DH the respective algorithms and formats from Table A.1.2 and Table A.8 MUST be used.

OID	**Hash**
id-RI-DH-SHA-1	SHA-1
id-RI-DH-SHA-224	SHA-224
id-RI-DH-SHA-256	SHA-256

Table A.8: Object Identifiers for Restricted Identification with DH

A.5.4. Restricted Identification with ECDH

For Restricted Identification with ECDH the respective algorithms and formats from Table A.1.2 and Table A.9 MUST be used. Input to the hash function SHALL be the x-coordinate of the point generated by ECKA [4].

OID	**Hash**
id-RI-ECDH-SHA-1	SHA-1
id-RI-ECDH-SHA-224	SHA-224
id-RI-ECDH-SHA-256	SHA-256

Table A.9: Object Identifiers for Restricted Identification with ECDH

A.6. Terminal Authentication

A.6.1. Public Key References

Public keys to be used for Terminal Authentication MUST be contained in CV Certificates according to the certificate profile defined in Appendix C.1. Each CV Certificate MUST contain two public key references, a *Certificate Holder Reference* and a *Certification Authority Reference:*

Certificate Holder Reference: The Certificate Holder Reference is an identifier for the public key provided in the certificate that SHALL be used to reference this public key.

Certification Authority Reference: The Certification Authority Reference is a reference to the (external) public key of the certification authority that SHALL be used to verify the signature of the certificate.

Note: As a consequence the Certification Authority Reference contained in a certificate MUST be equal to the Certificate Holder Reference in the corresponding certificate of the issuing certification authority.

	Encoding	**Length**
Country Code	ISO 3166-1 ALPHA-2	2F
Holder Mnemonic	ISO/IEC 8859-1	9V
Sequence Number	ISO/IEC 8859-1	5F

F: fixed length (exact number of octets), V: variable length (up to number of octets)

Table A.10: Certificate Holder Reference

The Certificate Holder Reference SHALL consist of the following concatenated elements: *Country Code*, *Holder Mnemonic*, and *Sequence Number*. Those elements MUST be chosen according to Table A.10 and the following rules:

1. **Country Code**

   The Country Code SHALL be the ISO 3166-1 ALPHA-2 code of the certificate holder's country.

2. **Holder Mnemonic**

   The Holder Mnemonic SHALL be assigned as unique identifier as follows:

   - The Holder Mnemonic of a CVCA SHALL be assigned by the CVCA itself.
   - The Holder Mnemonic of a DV SHALL be assigned by the *domestic* CVCA.
   - The Holder Mnemonic of an IS SHALL be assigned by the supervising DV.

3. **Sequence Number**

   The Sequence Number SHALL be assigned by the certificate holder.

   - The Sequence Number MUST be numeric or alphanumeric:
     – A numeric Sequence Number SHALL consist of the characters "0"..."9".
     – An alphanumeric Sequence Number SHALL consist of the characters "0"..."9" and "A"..."Z".
   - The Sequence Number MAY start with the ISO 3166-1 ALPHA-2 country code of the certifying certification authority, the remaining three characters SHALL be assigned as alphanumeric Sequence Number.
   - The Sequence Number MAY be reset if all available Sequence Numbers are exhausted.

A.6.2. Public Key Import

Public keys imported by the certificate validation procedure (cf. Section 2.2.5) are either *permanently* or *temporarily* stored on the MRTD chip. The MRTD chip SHOULD reject to import a public key, if the Certificate Holder Reference is already known to the MRTD chip.

A.6.2.1. Permanent Import

Public keys contained in CVCA Link Certificates SHALL be permanently imported by the MRTD chip and MUST be stored securely in the MRTD chip's memory. A permanently imported public key and its metadata SHALL fulfill the following conditions:

- It MAY be overwritten *after expiration* by a subsequent permanently imported public key.
- Either it MUST be overwritten by a subsequent permanently imported public key with the same Certificate Holder Reference or the import MUST be rejected.
- It MUST NOT be overwritten by a temporarily imported public key.

Note: It is RECOMMENDED to reject to import a public key, if the Certificate Holder Reference is already known to the MRTD chip.

Enabling and disabling a permanently imported public key MUST be an atomic operation.

A.6.2.2. Temporary Import

Public keys contained in DV and Terminal Certificates SHALL be temporarily imported by the MRTD chip. A temporarily imported public key and its metadata SHALL fulfill the following conditions:

- It SHALL NOT be selectable or usable after a power down of the MRTD chip.
- It MUST remain usable until the subsequent cryptographic operation is successfully completed (i.e. PSO:Verify Certificate or External Authenticate).
- It MAY be overwritten by a subsequent temporarily imported public key.

A terminal MUST NOT make use of any temporarily imported public key but the most recently imported.

A.6.2.3. Imported Metadata

For each permanently or temporarily imported public key the following additional data contained in the certificate (cf. Appendix C.1) MUST be stored:

- Certificate Holder Reference
- Certificate Holder Authorization (effective role and effective authorization)
- Certificate Effective Date
- Certificate Expiration Date
- Certificate Extensions

File Name	**EF.CVCA**
File ID	0x011C (default)
Short File ID	0x1C (default)
Read Access	PACE
Write Access	NEVER (internally updated only)
Size	36 bytes (fixed) padded with octets of value 0x00
Content	[CAR_i][||CAR_{i-1}][||0x00..00]

Table A.11: Elementary File EF.CVCA

The calculation of the effective role (CVCA, DV, or Terminal) and the effective authorization of the certificate holder is described in Appendix C.4.

Note: The format of the stored data is operating system dependent and out of the scope of this specification.

A.6.2.4. EF.CVCA

Support for the elementary file EF.CVCA is CONDITIONAL. If the MRTD chip supports Terminal Authentication in version 1 it MUST make the references of CVCA public keys suitable for inspection systems available in a transparent elementary file EF.CVCA contained in the ePassport application as specified in Table A.11.

This file SHALL contain a sequence of Certification Authority Reference (CAR) data objects (cf. Appendix D.2) suitable for Terminal Authentication.

- It SHALL contain at most two Certification Authority Reference data objects.
- The most recent Certification Authority Reference SHALL be the first data object in this list.
- The file MUST be padded by appending octets of value 0x00.

The file EF.CVCA has a default file identifier and short file identifier. If the default values cannot be used, the (short) file identifier SHALL be specified in the OPTIONAL parameter `efCVCA` of the `TerminalAuthenticationInfo`. If `efCVCA` is used to indicate the file identifier to be used, the default file identifier is overridden. If no short file identifier is given in `efCVCA`, the file EF.CVCA MUST be explicitly selected using the given file identifier.

A.6.3. Terminal Authentication with RSA

For Terminal Authentication with RSA the following algorithms and formats MUST be used.

OID	**Signature**	**Hash**	**Parameters**
id-TA-RSA-v1-5-SHA-1	RSASSA-PKCS1-v1_5	SHA-1	N/A
id-TA-RSA-v1-5-SHA-256	RSASSA-PKCS1-v1_5	SHA-256	N/A
id-TA-RSA-PSS-SHA-1	RSASSA-PSS	SHA-1	default
id-TA-RSA-PSS-SHA-256	RSASSA-PSS	SHA-256	default

Table A.12: Object Identifiers for Terminal Authentication with RSA

A.6.3.1. Signature Algorithm

RSA [18], [25] as specified in Table A.12 SHALL be used. The default parameters to be used with RSA-PSS are defined as follows:

- Hash Algorithm: The hash algorithm is selected according to Table A.12.
- Mask Generation Algorithm: MGF1 [18], [25] using the selected hash algorithm.
- Salt Length: Octet length of the output of the selected hash algorithm.
- Trailer Field: 0xBC

A.6.3.2. Public Key Format

The TLV-Format [15] as described in Appendix D.3.1 SHALL be used.

- The object identifier SHALL be taken from Table A.12.
- The bit length of the modulus SHALL be 1024, 1280, 1536, 2048, or 3072.
- The bit length of the exponent SHALL be at most 32.

A.6.4. Terminal Authentication with ECDSA

For Terminal Authentication with ECDSA the following algorithms and formats MUST be used.

OID	Signature	Hash
id-TA-ECDSA-SHA-1	ECDSA	SHA-1
id-TA-ECDSA-SHA-224	ECDSA	SHA-224
id-TA-ECDSA-SHA-256	ECDSA	SHA-256

Table A.13: Object Identifiers for Terminal Authentication with ECDSA

A.6.4.1. Signature Algorithm

ECDSA with plain signature format [4] as specified in Table A.13 SHALL be used.

A.6.4.2. Public Key Format

The TLV-Format [15] as described in Appendix D.3.3 SHALL be used.

- The object identifier SHALL be taken from Table A.13.
- The bit length of the curve SHALL be 160, 192, 224, or 256.
- Domain Parameters SHALL be compliant to [4].

Data Object
Authentication
Discretionary Data Template
Object Identifier
Discretionary Data
Discretionary Data Template
Object Identifier
Discretionary Data
...

Table A.14: Authenticated Auxiliary Data

A.6.5. Authenticated Auxiliary Data

Usage of auxiliary data in Terminal Authentication is CONDITIONAL. It MUST be used if further operations performed by the terminal require authenticated auxiliary data (details can be found in the following sections):

- For age verification, the terminal MUST commit to the required date of birth.
- For document validity verification the terminal MUST commit to the current date.
- For community ID verification the terminal MUST commit to (parts of) the community ID.

Authenticated auxiliary data MUST be structured as specified in Table A.14:

- An authentication data object that contains a sequence of discretionary data templates.
- Each discretionary data template contains an object identifier and a discretionary data object. The content of the discretionary data object is defined by the object identifier.

Only after a successful authentication of the terminal the MRTD chip SHALL interpret and make the data contained in the discretionary data object available for further operations.

Note: If the authentication data object contains more than one discretionary data template with the same object identifier, the data of the last discretionary data template SHALL be made available for further operations.

A.6.5.1. Object Identifier

The following object identifier SHALL be used to identify authenticated auxiliary data:

```
id-auxiliaryData OBJECT IDENTIFIER ::= {
  bsi-de applications(3) mrtd(1) 4
}
```

A.6.5.2. Age Verification

The following object identifier SHALL be used for age verification:

```
id-DateOfBirth OBJECT IDENTIFIER ::= {id-AuxiliaryData 1}
```

The discretionary data object SHALL contain the date of birth encoded as `Date` (cf. Appendix E.2) that is *required* by the terminal. The MRTD chip SHALL compare the stored date of birth to the required date of birth. Age Verification is successful if the stored date of birth is not after the required date of birth.

A.6.5.3. Document Validity Verification

The following object identifier SHALL be used for document validity verification:

```
id-DateOfExpiry OBJECT IDENTIFIER ::= {id-AuxiliaryData 2}
```

The discretionary data object SHALL contain the current date of the terminal encoded as `Date` (cf. Appendix E.2). The MRTD chip SHALL compare the stored date of expiry to the current date. Document Validity Verification is successful if the stored date of expiry is not before the given current date.

A.6.5.4. Community ID Verification

The following object identifier SHALL be used for community ID verification:

```
id-CommunityID OBJECT IDENTIFIER ::= {id-AuxiliaryData 3}
```

The discretionary data object SHALL contain (parts of) the community ID encoded as `Octet-String` (cf. Appendix E.2). The MRTD chip SHALL compare the leftmost octets of the stored community ID to the transmitted (part of the) requested community ID. Community ID Verification is successful if the leftmost octets of the stored data are identical to the transmitted data.

B. ISO 7816 Mapping (Normative)

In this Appendix the protocols for PACE, Chip Authentication and Terminal Authentication are mapped to ISO 7816 APDUs (Application Program Data Units).

B.1. PACE

The following sequence of commands SHALL be used to implement PACE:

1. MSE:Set AT
2. General Authenticate

The protocol specific data objects SHALL be exchanged in a chain of General Authenticate commands as shown below:

Step	Description	Protocol Command Data		Protocol Response Data	
1.	Encrypted Nonce	-	Absent[4]	0x80	Encrypted Nonce
2.	Map Nonce	0x81	Mapping Data	0x82	Mapping Data
3.	Perform Key Agreement	0x83	Ephemeral Public Key	0x84	Ephemeral Public Key
4.	Mutual Authentication	0x85	Authentication Token	0x86	Authentication Token
				0x87	Certification Authority Reference (CONDITIONAL)
				0x88	Certification Authority Reference (CONDITIONAL)

The Certificate Authority Reference(s) are REQUIRED if PACE is used with a CHAT. In this case the data object 0x87 SHALL contain the most recent Certificate Authority Reference with respect to the terminal type indicated in the CHAT. The data object 0x88 MAY contain the previous Certificate Authority Reference.

B.1.1. Encrypted Nonce

The encrypted nonce (cf. Appendix A.3.3) SHALL be encoded as octet string.

B.1.2. Mapping Data

The exchanged data is specific to the used mapping.

4 This implies an empty Dynamic Authentication Data Object.

B.1.2.1. Generic Mapping

The ephemeral public keys (cf. Appendix A.2.2 and Appendix D.3.4) SHALL be encoded as elliptic curve point (ECDH) or unsigned integer (DH).

B.1.2.2. Integrated Mapping

The symmetric key SHALL be encoded as octet string.

Note: The context specific data object 0x82 is not used.

B.1.3. Authentication Token

The authentication token (cf. Appendix A.2.4) SHALL be encoded as octet string.

B.1.4. Certification Authority Reference

The MRTD chip SHALL return the Certificate Authority References of *appropriate* CVCA public keys stored on the MRTD chip:

- The references MUST by dynamically chosen to suit the terminal type indicated by PACE.
- It SHALL return at most two Certification Authority Reference data objects.
- The most recent Certification Authority Reference SHALL be contained in data object 0x87.

B.2. Chip Authentication

The following command MAY be used to implement Chip Authentication in version 1 with 3DES Secure Messaging:

1. MSE:Set KAT

Note: The usage of MSE:Set KAT is deprecated. It may only be used for `id-CA-DH-3DES-CBC-CBC` and `id-CA-ECDH-3DES-CBC-CBC`, i.e. Secure Messaging is restricted to 3DES.

The following sequence of commands SHALL be used to implement Chip Authentication in version 1 and 2:

1. MSE:Set AT
2. General Authenticate

The protocol specific data objects SHALL be exchanged with a General Authenticate command as shown below:

Step	Description	Protocol Command Data		Protocol Response Data	
1.	Chip Authentication	0x80	Ephemeral Public Key	0x81	Nonce
				0x82	Authentication Token

Note: Support of Protocol Response Data is CONDITIONAL: It MUST be provided for version 2 but MUST NOT be provided for version 1.

B.2.1. Ephemeral Public Key

The ephemeral public keys (cf. Appendix A.2.2 and Appendix D.3.4) SHALL be encoded as elliptic curve point (ECDH) or unsigned integer (DH).

B.2.2. Nonce

The nonce SHALL be encoded as octet string of size 8 octets.

B.2.3. Authentication Token

The authentication token (cf. Appendix A.2.4) SHALL be encoded as octet string.

B.3. Terminal Authentication

The following sequence of commands SHALL be used to implement Terminal Authentication:

1. MSE:Set DST
2. PSO:Verify Certificate
3. MSE:Set AT
4. Get Challenge
5. External Authenticate

Steps 1 and 2 are repeated for every CV certificate to be verified (CVCA Link Certificates, DV Certificate, Terminal Certificate).

For Terminal Authentication in version 2 the MRTD chip MUST in addition support the usage of Get Challenge before step 1, i.e. the MRTD chip MUST keep a generated challenge upon usage by External Authenticate.

B.4. Restricted Identification

The following sequence of commands SHALL be used to implement Restricted Identification:

1. MSE:Set AT
2. General Authenticate

The protocol specific data objects SHALL be exchanged with General Authenticate commands as shown below, command chaining MUST NOT be used. At least one of the steps MUST be executed:

Step	Description	Protocol Command Data		Protocol Response Data	
1.	Restricted Identification (CONDITIONAL)	0xA0	1st Sector Public Key	0x81	1st Sector-specific Identifier
2.	Restricted Identification (CONDITIONAL)	0xA2	2nd Sector Public Key	0x83	2nd Sector-specific Identifier

B.4.1. Public Key

The sector public key PK_{Sector} SHALL be encoded as public key data object (cf. Appendix D.3), the domain parameters MUST be included.

B.4.2. Sector-specific Identifier

The sector-specific identifier I_{ID}^{Sector} SHALL be encoded as octet string.

B.5. Auxiliary Data Verification

The following command SHALL be used to implement the verification function:

1. Verify

The following authenticated auxiliary data MUST have been sent to the MRTD chip as as part of Terminal Authentication:

- For Age Verification the terminal MUST have sent the required date of birth.
- For Document Validity Verification the terminal MUST have sent the current date.
- For Community ID Verification the terminal MUST have sent the (part of the) community ID.

B.6. PIN Management

B.6.1. Unblock or Change PIN

The following command SHALL be used to implement unblocking and/or changing of the PIN:

1. Reset Retry Counter
   - To set a new PIN and reset the retry counter the terminal SHALL use Reset Retry Counter with the new PIN as data.
   - To reset the retry counter the terminal SHALL use Reset Retry Counter with no data.

Usage of the command SHALL be restricted to authorized terminals: Before using this command the terminal must either authenticate as Authentication Terminal with effective access right for PIN Management or by using PACE with the PUK/PIN.

B.6.2. Activate or Deactivate PIN

The following command SHALL be used to activate the PIN:

1. Activate

The following command SHALL be used to deactivate the PIN:

2. Deactivate

Usage of the command SHALL be restricted to authorized terminals: Before using this command the terminal must authenticate as Authentication Terminal with effective access right for PIN Management.

B.7. eSign Application

Commands for installing, updating, and using the eSign application are out of scope of this specification.

B.8. Reading Data Groups

The APDUs for selecting and reading EAC-protected data groups already specified by ICAO [8], [9] SHALL be used (i.e. Select File and Read Binary). In accordance with ICAO specifications any unauthorized access to EAC-protected data groups SHALL be denied and the MRTD chip MUST respond with status bytes 0x6982 (“Security status not satisfied”).

B.9. Extended Length

Depending on the size of the cryptographic objects (e.g. public keys, signatures), APDUs with extended length fields MUST be used to send this data to the MRTD chip. For details on extended length see [13].

B.9.1. MRTD Chips

For MRTD chips support of extended length is CONDITIONAL. If the cryptographic algorithms and key sizes selected by the issuing state require the use of extended length, the MRTD chips SHALL support extended length. If the MRTD chip supports extended length this MUST be indicated in the ATR/ATS or in EF.ATR as specified in [13].

B.9.2. Terminals

For terminals support of extended length is REQUIRED. A terminal SHOULD examine whether or not support for extended length is indicated in the MRTD chip's ATR/ATS or in EF.ATR before using this option. The terminal MUST NOT use extended length for APDUs other than the following commands unless the exact input and output buffer sizes of the MRTD chip are explicitly stated in the ATR/ATS or in EF.ATR.

- PSO:Verify Certificate
- MSE:Set KAT
- General Authenticate
- External Authenticate

B.9.3. Errors

The MRTD chip SHALL indicate extended length errors with status bytes 0x6700.

B.10. Command Chaining

Command chaining is only used for the General Authenticate command. For details on command chaining see [13].

B.10.1. MRTD Chips

For MRTD chips support of command chaining is REQUIRED and support for command chaining MUST be indicated in the historical bytes of the ATR/ATS or in the EF.ATR as specified in [13].

B.10.2. Terminals

For terminals support of command chaining is REQUIRED. A terminal SHOULD test whether or not the MRTD chip supports command chaining before using this option.

B.10.3. Errors

If the MRTD chip expects the end of the chain, but receives a command that is not marked as the last command, the MRTD chip SHALL indicate that the last command in a chain was expected with status bytes 0x6883.

B.11. APDU Specification

B.11.1. MSE:Set AT

The command MSE:Set AT is used to select and initialize the following protocols: PACE, Chip Authentication, Terminal Authentication, and Restricted Identification.

Command			
CLA		Context specific	
INS	0x22	Manage Security Environment	
P1/P2	0xC1A4	*PACE:* Set Authentication Template for mutual authentication	
	0x41A4	*Chip Authentication / Restricted Identification:* Set Authentication Template for internal authentication	
	0x81A4	*Terminal Authentication:* Set Authentication Template for external authentication	
Data	0x80	*Cryptographic mechanism reference* Object Identifier of the protocol to select (value only, Tag 0x06 is omitted). This data object is REQUIRED for all protocols except Terminal Authentication in version 1.	CONDITIONAL
	0x83	*Reference of a public key / secret key* This data object is REQUIRED for the following protocols: • For PACE to indicate the password to be used: 0x01: MRZ 0x02: CAN 0x03: PIN 0x04: PUK • For Terminal Authentication to select the public key of the terminal by its ISO 8859-1 encoded name.	CONDITIONAL
	0x84	*Reference of a private key / Reference for computing a session key* This data object is REQUIRED for the following protocols (cf. Appendix A.2): • For PACE to indicate the identifier of the domain parameters to be used if the domain parameters are ambiguous, i.e. more than one set of domain parameters is available for PACE. • For Chip Authentication to indicate the identifier of the private key to be used if the private key is ambiguous, i.e. more than one private key is available for Chip Authentication. • For Restricted Identification to indicate the private	CONDITIONAL

		key to be used i.e. more than one private key is available for Restricted Identification.	
	0x67	*Auxiliary authenticated data* This data object is REQUIRED for Terminal Authentication (version 2) if age verification, document validity verification, or community ID verification shall be used.	CONDITIONAL
	0x91	*Ephemeral Public Key* This data object is REQUIRED for Terminal Authentication if the terminal's ephemeral public key $\widetilde{PK}_{PCD}$ is unknown or ambiguous to the MRTD chip when Terminal Authentication is performed (i.e. version 2). In this case the terminal's compressed ephemeral public key $\mathbf{Comp}(\widetilde{PK}_{PCD})$ MUST be sent to the MRTD chip.	CONDITIONAL
	0x7F4C	*Certificate Holder Authorization Template* This data object is REQUIRED for PACE if Terminal Authentication shall be used after PACE	CONDITIONAL

Response		
Data	–	Absent
Status Bytes	0x9000	*Normal operation* The protocol has been selected and initialized.
	0x6A80	*Incorrect parameters in the command data field* Algorithm not supported or initialization failed.
	0x6A88	*Referenced data not found* The referenced data (i.e. password, private key, public key, domain parameter) is not available.
PACE	0x63CX	*Warning* The password has been selected. X indicates the number of remaining verification tries, if not equal to the initial value: X=1: The password is suspended. The password MUST be resumed. X=0: The password is blocked. The password MUST be unblocked.
PACE	0x6283	*Warning* The password is deactivated.
	other	*Operating system dependent error* The initialization of the protocol failed.

Note:

- Some operating systems accept the selection of an unavailable public key and return an error only when the public key is used for the selected purpose.
- Resuming and unblocking a password requires explicitly setting the CAN or the PUK using MSE:Set AT.

B.11.2. General Authenticate

The command General Authenticate is used to perform the following protocols: PACE, Chip Authentication, and Restricted Identification.

Command			
CLA		Context specific.	
INS	0x86	General Authenticate	
P1/P2	0x0000	Keys and protocol implicitly known	
Data	0x7C	*Dynamic Authentication Data* Protocol specific data objects	REQUIRED

Response			
Data	0x7C	*Dynamic Authentication Data* Protocol specific data objects	REQUIRED
Status Bytes	0x9000	*Normal operation* The protocol (step) was successful.	
	0x6300	*Authentication failed* The protocol (step) failed.	
	0x63CX	*Authentication failed* The protocol (step) failed. X indicates the number of remaining verification tries: X=1: The password is suspended. The password MUST be resumed. X=0: The password is blocked. The password MUST be unblocked.	
	0x6982	*Security status not satisfied* The terminal is not authorized to perform the protocol (e.g. the password is blocked, deactivated, or suspended).	
	0x6983	*Authentication method blocked* The password is blocked.	
	0x6984	*Reference data not usable* The password is deactivated.	
	0x6985	*Conditions of use not satisfied* The password is suspended.	
	0x6A80	*Incorrect parameters in data field* Provided data is invalid.	
	other	*Operating system dependent error* The protocol (step) failed.	

Note: The MRTD chip MAY indicate a blocked, deactivated, or suspended password by responding with status bytes 0x6982 instead of using status bytes 0x6983, 0x6984, or 0x6985, respectively.

B.11.3. MSE:Set KAT (Deprecated)

The command MSE:Set KAT is used to perform Chip Authentication version 1 with 3DES. Usage of this command is deprecated.

Command			
CLA	0x0C	Secure Messaging with authenticated header, no chaining	
INS	0x22	Manage Security Environment	
P1/P2	0x41A6	Set for computation / Key Agreement Template	
Data	0x91	*Ephemeral Public Key* Ephemeral public key $\widetilde{PK}_{PCD}$ (cf. Appendix A.2) encoded as plain public key value.	REQUIRED
	0x84	*Reference of a private key* This data object is REQUIRED if the private key is ambiguous, i.e. more than one key pair is available for Chip Authentication (cf. Appendix A.1.1.2 and Appendix A.4.1)	CONDITIONAL

Response		
Data	–	Absent
Status Bytes	0x9000	*Normal operation* The key agreement operation was successfully performed. New session keys have been derived.
	0x6A80	*Incorrect Parameters in the command data field* The validation of the ephemeral public key failed. The previously established session keys remain valid.
	other	*Operating system dependent error* The previously established session keys remain valid.

B.11.4. MSE:Set DST

The command MSE:Set DST is used to setup certificate verification for Terminal Authentication.

Command			
CLA	0x0C	Secure Messaging with authenticated header, no chaining	
INS	0x22	Manage Security Environment	
P1/P2	0x81B6	Set for verification: Digital Signature Template	
Data	0x83	*Reference of a public key* ISO 8859-1 encoded name of the public key to be set	REQUIRED

Response		
Data	–	Absent
Status Bytes	0x9000	*Normal Operation* The key has been selected for the given purpose.

	0x6A88	*Referenced data not found* The selection failed as the public key is not available
	other	*Operating system dependent error* The key has not been selected.

Note: Some operating systems accept the selection of an unavailable public key and return an error only when the public key is used for the selected purpose.

B.11.5. PSO:Verify Certificate

The command PSO:Verify Certificate is used to verify and import certificates for Terminal Authentication.

Command			
CLA	0x0C	Secure Messaging with authenticated header, no chaining	
INS	0x2A	Perform Security Operation	
P1/P2	0x00BE	Verify self-descriptive certificate	
Data	0x7F4E	*Certificate body* The body of the certificate to be verified	REQUIRED
	0x5F37	*Signature* The signature of the certificate to be verified	REQUIRED

Response		
Data	–	Absent
Status Bytes	0x9000	*Normal operation* The certificate was successfully validated and the public key has been imported.
	other	*Operating system dependent error* The public key could not be imported (e.g. the certificate was not accepted).

B.11.6. Get Challenge

The command Get Challenge is used to perform Terminal Authentication.

Command			
CLA	0x0C	Secure Messaging with authenticated header, no chaining	
INS	0x84	Get Challenge	
P1/P2	0x0000		
Data	–	Absent	
Le	0x08		REQUIRED

Response		
Data	r_{PICC}	8 bytes of randomness.
Status Bytes	0x9000 other	*Normal operation* *Operating system dependent error*

B.11.7. External Authenticate

The command External Authenticate is used to perform Terminal Authentication.

Command			
CLA	0x0C	Secure Messaging with authenticated header, no chaining	
INS	0x82	External Authenticate	
P1/P2	0x0000	Keys and Algorithms implicitly known	
Data		Signature generated by the terminal.	REQUIRED

Response		
Data	–	Absent
Status Bytes	0x9000	*Normal operation* The authentication was successful. Access to data groups will be granted according to the effective authorization of the corresponding verified certificate.
	0x6982	*Security status not satisfied* The authentication failed as the current authentication level of the terminal does not allow to use Terminal Authentication (e.g. Terminal Authentication was already performed, etc.)
	other	*Operating system dependent error* The authentication failed.

B.11.8. Verify

The command Verify is used to verify authenticated auxiliary data, i.e. to perform age verification, document validity verification, or Community ID verification.

Command			
CLA	0x8C	Secure Messaging with authenticated header, no chaining, application specific	
INS	0x20	Verify	
P1/P2	0x8000	Verify authenticated auxiliary data.	
Data		Object Identifier of the auxiliary data to be verified.	REQUIRED

Response		
Data	–	Absent
Status Bytes	0x9000	*Normal operation* Verification successful.
	0x6300	*Verification failed* Verification failed.
	0x6A88	*Referenced data not found* The referenced data was not found.
	0x6982	*Security status not satisfied* The terminal is not authorized to perform verification
	other	*Operating system dependent error* Verification failed.

B.11.9. Reset Retry Counter

The command Reset Retry Counter is used to unblock or change the PIN.

Command			
CLA	0x0C	Secure Messaging with authenticated header, no chaining	
INS	0x2C	Reset Retry Counter	
P1	0x02- 0x03	see below	
P2		0x02: CAN 0x03: PIN	

Data		Context specific reset data depending on P1:
		P1=0x02: new PIN/CAN
		P1=0x03: Absent
			REQUIRED

Response		
Data	–	Absent
Status Bytes	0x9000	*Normal operation* Unblocking or changing of the PIN was successful.
	0x6982	*Security status not satisfied* The terminal is not authorized to unblock or change the PIN.
	other	*Operating system dependent error* Unblocking or changing of the PIN failed.

Note: As the CAN is a non-blocking password, unblocking the CAN is not necessary. The MRTD chip MAY however support changing the CAN.

B.11.10. Activate

The command Activate is used to set the PIN to the state activated.

Command		
CLA	0x0C	Secure Messaging with authenticated header, no chaining
INS	0x44	Activate
P1	0x10	Activate PIN referenced by parameter P2
P2		0x03: PIN
Data		Absent

Response		
Data	–	Absent
Status Bytes	0x9000	*Normal operation* PIN state has been set to activated.
	0x6982	*Security status not satisfied* The terminal is not authorized to change the PIN state.
	other	*Operating system dependent error* Changing of the PIN state failed.

B.11.11. Deactivate

The command Deactivate is used to set the PIN to the state deactivated.

Command		
CLA	0x0C	Secure Messaging with authenticated header, no chaining
INS	0x04	Deactivate
P1	0x10	Deactivate PIN referenced by parameter P2
P2		0x03: PIN
Data		Absent

Response		
Data	–	Absent
Status Bytes	0x9000	*Normal operation* PIN state has been set to deactivated.
	0x6982	*Security status not satisfied* The terminal is not authorized to change the PIN state.
	other	*Operating system dependent error* Changing of the PIN state failed.

C. CV Certificates

C.1. Certificate Profile

Self-descriptive card verifiable (CV) certificates according to ISO 7816 (cf. [13], [14], [15]) and the certificate profile specified in Table C.1 SHALL be used. Details on the encoding of the data objects used in the certificate profile can be found in Appendix D.2.

C.1.1. Certificate Profile Identifier

The version of the profile is indicated by the Certificate Profile Identifier. Version 1 as specified in Table C.1 is identified by a value of 0.

C.1.2. Certification Authority Reference

The Certification Authority Reference is used to identify the public key to be used to verify the signature of the certification authority (CVCA or DV). The Certification Authority Reference MUST be equal to the Certificate Holder Reference in the corresponding certificate of the certification authority (CVCA Link Certificate or DV Certificate). Details on the Certification Authority Reference can be found in Appendix A.6.1.

C.1.3. Public Key

Details on the encoding of public keys can be found in Appendix D.3.

Data Object	**Cert**
CV Certificate	m
Certificate Body	m
Certificate Profile Identifier	m
Certification Authority Reference	m
Public Key	m
Certificate Holder Reference	m
Certificate Holder Authorization Template	m
Certificate Effective Date	m
Certificate Expiration Date	m
Certificate Extensions	o
Signature	m

Table C.1: CV Certificate Profile

C.1.4. Certificate Holder Reference

The Certificate Holder Reference is used to identify the public key contained in the certificate. Details on the Certificate Holder Reference can be found in Appendix A.6.1.

C.1.5. Certificate Holder Authorization Template

The role and authorization of the certificate holder SHALL be encoded in the Certificate Holder Authorization Template. This template is a sequence that consists of the following data objects:

1. An object identifier that specifies the terminal type and the format of the template.
2. A discretionary data object that encodes the relative authorization, i.e. the role and authorization of the certificate holder relative to the certification authority.

The content and evaluation of the Certificate Holder Authorization Template is described in Appendix C.4.

C.1.6. Certificate Effective/Expiration Date

Indicates the validity period of the certificate. The Certificate Effective Date MUST be the date of the certificate generation.

C.1.7. Certificate Extensions

The certificate MAY contain extensions as defined in Appendix C.3.

C.1.8. Signature

The signature on the certificate SHALL be created over the encoded certificate body (i.e. including tag and length). The Certification Authority Reference SHALL identify the public key to be used to verify the signature.

C.2. Certificate Requests

Certificate requests are reduced CV certificates that may carry an additional signature. The certificate request profile specified in Table C.2 SHALL be used. Details on the encoding of the data objects used in the certificate request profile can be found in Appendix D.2.

C.2.1. Certificate Profile Identifier

The version of the profile is identified by the Certificate Profile Identifier. Version 1 as specified in Table C.2 is identified by a value of 0.

C.2.2. Certification Authority Reference

The Certification Authority Reference SHOULD be used to inform the certification authority about the private key that is **expected** by the applicant to be used to sign the certificate. If the Certification Authority Reference contained in the request deviates from the Certification Authority Reference contained in the issued certificate (i.e. the issued certificate is signed by a private key that is **not expected** by the applicant), the corresponding certificate of the certification authority SHOULD also be provided to the applicant in response.

Details on the Certification Authority Reference can be found in Appendix A.6.1.

C.2.3. Public Key

Details on the encoding of public keys can be found in Appendix D.3.

Data Object	**Req**
Authentication	c
CV Certificate	m
Certificate Body	m
Certificate Profile Identifier	m
Certification Authority Reference	r
Public Key	m
Certificate Holder Reference	m
Certificate Extensions	o
Signature	m
Certification Authority Reference	c
Signature	c

Table C.2: CV Certificate Request Profile

C.2.4. Certificate Holder Reference

The Certificate Holder Reference is used to identify the public key contained in the request and the resulting certificate. Details on the Certificate Holder Reference can be found in Appendix A.6.1.

C.2.5. Certificate Extensions

The certificate request MAY contain extensions as defined in Appendix C.3.

C.2.6. Signature(s)

A certificate request may have two signatures, an *inner signature* and an *outer signature*:

- **Inner Signature** **(REQUIRED)**

  The certificate body is self-signed, i.e. the inner signature SHALL be verifiable with the public key contained in the certificate request. The signature SHALL be created over the encoded certificate body (i.e. including tag and length).

- **Outer Signature** **(CONDITIONAL)**
  - The signature is OPTIONAL if an entity applies for the initial certificate. In this case the request MAY be additionally signed by another entity trusted by the receiving certification authority (e.g. the national CVCA may authenticate the request of a DV sent to a foreign CVCA).
  - The signature is REQUIRED if an entity applies for a successive certificate. In this case the request MUST be additionally signed by the applicant using a recent key pair previously registered with the receiving certification authority.

If the outer signature is used, an authentication data object SHALL be used to nest the CV Certificate (Request), the Certification Authority Reference and the additional signature. The Certification Authority Reference SHALL identify the public key to be used to verify the additional signature. The signature SHALL be created over the concatenation of the encoded CV Certificate *and* the encoded Certification Authority Reference (i.e. both including tag and length).

C.3. Certificate Extensions

The certificate extension is a sequence of discretionary data templates, where every discretionary data template SHALL contain the following data objects:

1. An object identifier that specifies the content and the format of the extension.
2. One or more context specific data objects that contain the encoded extension.

The following base object identifier is used to identify the certificate extensions defined below:

```
id-extensions OBJECT IDENTIFIER ::= {
  bsi-de applictions(3) mrtd(1) 3
}
```

C.3.1. Certificate Description

The following object identifier SHALL be used for this extension:

```
id-description OBJECT IDENTIFIER ::= {id-extensions 1}
```

The following context specific data object is used to encode the certificate description:

- 0x80: Hash of `CertificateDescription`

```
CertificateDescription ::= SEQUENCE {
  descriptionType  OBJECT IDENTIFIER,
  issuerName    [1] UTF8String,
  issuerURL     [2] PrintableString OPTIONAL,
  subjectName   [3] UTF8String,
  subjectURL    [4] PrintableString OPTIONAL,
  termsOfUsage  [5] ANY DEFINED BY descriptionType
}
```

The hash function to be used SHALL be defined by the hash function used to sign the certificate.

Note: The certificate description is used by a local terminal as part of the user interaction for online authentication of a remote terminal (cf. Section 3.4.1) and may be ignored by the MRTD chip.

C.3.1.1. Plain Text Format

The following object identifier SHALL be used to identify terms of usage in plain text format:

```
id-plainFormat OBJECT IDENTIFIER ::= {id-description 1}
PlainTermsOfUsage ::= UTF8String
```

C.3.1.2. HTML Format

The following object identifier SHALL be used to identify terms of usage in HTML format:

```
id-htmlFormat OBJECT IDENTIFIER ::= {id-description 2}
HtmlTermsOfUsage ::= IA5String
```

C.3.1.3. PDF Format

The following object identifier SHALL be used to identify terms of usage in PDF format [11]:

```
id-pdfFormat OBJECT IDENTIFIER ::= {id-description 3}
PdfTermsOfUsage ::= OCTET STRING
```

C.3.2. Terminal Sector

The following object identifier SHALL be used for this extension:

```
id-sector OBJECT IDENTIFIER ::= {id-extensions 2}
```

The following context specific data objects are used to encode the terminal sector:

- 0x80: Hash of 1st sector public key data object (cf. Appendix D.3).

- 0x81: Hash of 2nd sector public key data object (cf. Appendix D.3).

The hash function to be used SHALL be defined by the hash function used to sign the certificate. The public key itself is not contained in the certificate and MUST be provided by the terminal as part of Restricted Identification. The MRTD chip SHALL compute the hash over the received public key and compare it to the received hash.

Note: Context-specific tagging is used for the sector public key in Restricted Identification (cf. Appendix B.4.1). The MRTD chip MUST replace the context-specific tag 0xA0 by the application-specific tag 0x7F49 before computing the hash value.

C.4. Roles and Authorization Levels

The following object identifier SHALL be used to identify roles and authorization levels for EAC-protected MRTD chips:

```
id-roles OBJECT IDENTIFIER ::= {
  bsi-de applications(3) mrtd(1) 2
}
```

C.4.1. Relative Authorization

C.4.1.1. Inspection Systems

The following Object Identifier SHALL be used for inspection systems:

```
id-IS OBJECT IDENTIFIER ::= {id-roles 1}
```

The *relative authorization* of the certificate holder is encoded in one byte which is to be interpreted as binary bit map as shown in Table C.3. In more detail, this bit map contains a role and access rights. Both are relative to the authorization of all previous certificates in the chain.

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Description**
x	x	-	-	-	-	-	-	**Role**
1	1	-	-	-	-	-	-	CVCA
1	0	-	-	-	-	-	-	DV (official domestic)
0	1	-	-	-	-	-	-	DV (official foreign)
0	0	-	-	-	-	-	-	Inspection System
-	-	x	x	x	x	x	x	**Access Rights**
-	-	1	-	-	-	-	-	Read access to eID application
-	-	-	0	0	0	-	-	RFU
-	-	-	-	-	-	1	-	Read access to ePassport application: DG 4 (Iris)
-	-	-	-	-	-	-	1	Read access to ePassport application: DG 3 (Fingerprint)

Table C.3: Authorization of Inspection Systems

An authenticated inspection system (cf. Section 3.2.1) SHALL always have access to less-sensitive datagroups (e.g. DG1, DG2, DG14).

C.4.1.2. Authentication Terminals

The following Object Identifier SHALL be used for authentication terminals:

```
id-AT OBJECT IDENTIFIER ::= {id-roles 2}
```

The *relative authorization* of the certificate holder is encoded in five bytes which are to be interpreted as binary bit map as shown in Table C.4. In more detail, this bit map contains a role and access rights. Both are relative to the authorization of all previous certificates in the chain.

39	38	37	…	33	32	…	29	28	…	8	7	6	5	4	3	2	1	0	Description
x	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**Role**
1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	CVCA
1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	DV (official domestic)
0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	DV (non-official / foreign)
0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Authentication Terminal
-	-	x	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**Write Access (eID)**
-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	DG 17
-	-	-	…	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	…
-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	DG 21
-	-	-	-	-	x	x	x	-	-	-	-	-	-	-	-	-	-	-	RFU: R/W Access
-	-	-	-	-	-	-	-	x	x	x	-	-	-	-	-	-	-	-	**Read Access (eID)**
-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	DG 21
-	-	-	-	-	-	-	-	-	…	-	-	-	-	-	-	-	-	-	…
-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	DG 1
-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	**Special Functions**
-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	Install Qualified Certificate
-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	Install Certificate
-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	PIN Management
-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	CAN allowed
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	RFU
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	Restricted Identification
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	Community ID Verification
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	Age Verification

Table C.4: Authorization of Authentication Terminals

An authenticated authentication terminal (cf. Section 3.2.2) SHALL always have access to the following functions:

- Restricted Identification unless the MRTD chip requires the terminal to be authorized to use the function (`authorizedOnly` is set, cf. Appendix A.1.1.4)
- Document Validity Verification

C.4.1.3. Signature Terminals

The following Object Identifier SHALL be used for signature terminals:

```
id-ST OBJECT IDENTIFIER ::= {id-roles 3}
```

The *relative authorization* of the certificate holder is encoded in one byte which is to be interpreted as binary bit map as shown in Table C.5. In more detail, this bit map contains a role and access rights. Both are relative to the authorization of all previous certificates in the chain.

7	6	5	4	3	2	1	0	Description
x	x	-	-	-	-	-	-	**Role**
1	1	-	-	-	-	-	-	CVCA
1	0	-	-	-	-	-	-	DV (Accreditation Body)
0	1	-	-	-	-	-	-	DV (Certification Service Provider)
0	0	-	-	-	-	-	-	Signature Terminal
-	-	x	x	x	x	x	x	**Access Rights (eSign)**
-	-	0	0	0	0	-	-	RFU
-	-	-	-	-	-	1	-	Generate qualified electronic signature
-	-	-	-	-	-	-	1	Generate electronic signature

Table C.5: Authorization of Signature Terminals

C.4.2. Effective Authorization

To determine the *effective authorization* of a certificate holder, the MRTD chip MUST calculate a bitwise Boolean 'and' of the *relative authorization* contained in the Terminal Certificate, the referenced Document Verifier Certificate, the referenced CVCA Certificate and – if PACE was used –. the required authorization indicated with PACE.

C.4.2.1. Access Rights

The effective authorization is to be interpreted by the MRTD chip as follows:

- The effective role is a CVCA:
  - This link certificate was issued by the national CVCA.
  - The MRTD chip MUST update its internal trust-point, i.e. the public key and the effective authorization.
  - The certificate issuer is a trusted source of time and the MRTD chip MUST update its current date using the Certificate Effective Date.
  - The MRTD chip MUST NOT grant the CVCA extended access to sensitive data (i.e. the effective access rights SHOULD be ignored).
- The effective role is a DV:
  - The certificate was issued by the national CVCA for an authorized DV.
  - The certificate issuer is a trusted source of time and the MRTD chip MUST update its current date using the Certificate Effective Date.
  - The MRTD chip MUST NOT grant a DV extended access to sensitive data (i.e. the effective access rights SHOULD be ignored).
- The effective role is a Terminal:
  - The certificate was issued by either an official domestic, a foreign, or a non-official DV.
  - If the certificate is an accurate terminal certificate (cf. Section 2.2.5), the issuer is a trusted source of time and the MRTD chip MUST update its current date using the Certificate Effective Date.
  - The MRTD chip MUST grant the authenticated Terminal extended access to sensitive data according to the effective access rights.

A valid terminal certificate MUST be accepted as accurate by the MRTD chip if it was issued by an official domestic DV and SHOULD NOT be accepted as accurate otherwise.

C.5. Certificate Policy

It is RECOMMENDED that each CVCA and every DV publishes a certificate policy and/or a certification practice statement.

C.5.1. Procedures

The certificate policy SHOULD specify the following procedures:

- Entity identification, authentication, and registration;
- Certificate application, issuance, and distribution;
- Compromise and disaster recovery;
- Auditing.

C.5.2. Usage Restrictions

The certificate policy SHOULD imply restrictions on the devices used to store/process corresponding private keys and other sensitive (personal) data:

- Physical and operational security;
- Access control mechanisms;
- Evaluation and certification (e.g. Common Criteria Protection Profiles);
- Data Protection.

D. DER Encoding (Normative)

The Distinguished Encoding Rules (DER) according to X.690 [17] SHALL be used to encode both ASN.1 data structures and (application specific) data objects. The encoding results in a Tag-Length-Value (TLV) structure as follows:

Tag: The tag is encoded in one or two octets and indicates the content.

Length: The length is encoded as unsigned integer in one, two, or three octets resulting in a maximum length of 65535 octets. The minimum number of octets SHALL be used.

Value: The value is encoded in zero or more octets.

D.1. ASN.1

The encoding of data structures defined in ASN.1 syntax is described in X.690 [17].

D.2. Data Objects

Table D.1 gives an overview on the tags, lengths, and values of the data objects used in this specification.

Name	Tag	Len	Value	Comment
Object Identifier	0x06	V	Object Identifier	–
Discretionary Data	0x53	V	Octet String	Contains arbitrary data.
Certificate Extensions	0x65	V	Sequence	Nests certificate extensions.
Discretionary Data Template	0x73	V	Sequence	Nests arbitrary data objects.
Public Key	0x7F49	V	Sequence	Nests the public key value and the domain parameters.
CV Certificate	0x7F21	V	Sequence	Nests certificate body and signature.
Certificate Body	0x7F4E	V	Sequence	Nests data objects of the certificate body.
Certificate Profile Identifier	0x5F29	1F	Unsigned Integer	Version of the certificate and certificate request format.
Certification Authority Reference	0x42	16V	Character String	Identifies the public key of the issuing certification authority in a certificate.
Certificate Holder Reference	0x5F20	16V	Character String	Associates the public key contained in a certificate with an identifier.
Certificate Holder Authorization Template	0x7F4C	V	Sequence	Encodes the role of the certificate holder (i.e. CVCA, DV, Terminal) and assigns read/write access rights.
Certificate Effective Date	0x5F25	6F	Date	The date of the certificate generation.
Certificate Expiration Date	0x5F24	6F	Date	The date after which the certificate expires.
Signature	0x5F37	V	Octet String	Digital signature produced by an asymmetric cryptographic algorithm.
Authentication	0x67	V	Sequence	Contains authentication related data objects.

F: fixed length (exact number of octets), V: variable length (up to number of octets)

Table D.1: Overview on Data Objects

Note: The tag 0x7F4C is not yet defined by ISO/IEC 7816. The allocation is requested.

D.2.1. Encoding of Values

The basic value types used in this specification are the following: (unsigned) integers, elliptic curve points, dates, character strings, octet strings, object identifiers, and sequences.

D.2.1.1. Unsigned Integers

All integers used in this specification are unsigned integers. An unsigned integer SHALL be converted to an octet string using the binary representation of the integer in big-endian format. The minimum number of octets SHALL be used, i.e. leading octets of value 0x00 MUST NOT be used.

Note: In contrast the ASN.1 type `INTEGER` is always a signed integer.

Code	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2	SP	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8																
9																
A	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	SHY	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

SP: Space, NBSP: Non-breaking Space, SHY: Soft Hyphen

Table D.2: ISO/IEC 8859-1 Character Set

D.2.1.2. Elliptic Curve Points

The conversion of Elliptic Curve Points to octet strings is specified in [4]. The uncompressed format SHALL be used.

D.2.1.3. Dates

A date is encoded in 6 digits $d_1 \cdots d_6$ in the format YYMMDD using timezone GMT. It is converted to an octet string $o_1 \cdots o_6$ by encoding each digit d_j to an octet o_j as unpacked BCDs $(1 \leq j \leq 6)$.

The year YY is encoded in two digits and to be interpreted as 20YY, i.e. the year is in the range of 2000 to 2099.

D.2.1.4. Character Strings

A character string $c_1 \cdots c_n$ is a concatenation of n characters c_j with $1 \leq j \leq n$. It SHALL be converted to an octet string $o_1 \cdots o_n$ by converting each character c_j to an octet o_j using the ISO/IEC 8859-1 character set. For informational purposes the character set can be found in Table D.2.

The character codes 0x00-0x1F and 0x7F-0x9F are unassigned and MUST NOT be used. The conversion of an octet to an unassigned character SHALL result in an error.

D.2.1.5. Octet Strings

An octet string $o_1 \cdots o_n$ is a concatenation of n octets o_j with $1 \leq j \leq n$. Every octet o_j consists of 8 bits.

D.2.1.6. Object Identifiers

An object identifier $i_1.i_2.\cdots.i_n$ is encoded as an ordered list of n unsigned integers i_j with $1 \leq j \leq n$. It SHALL be converted to an octet string $o_1 \cdots o_{n-1}$ using the following procedure:

1. The first two integers i_1 and i_2 are packed into a single integer i that is then converted to the octet string o_1. The value i is calculated as follows:

$$i = i_1 \cdot 40 + i_2$$

2. The remaining integers i_j are directly converted to octet strings o_{j-1} with $3 \leq j \leq n$.

More details on the encoding can be found in [17].

Note: The unsigned integers are encoded as octet strings using the big-endian format as described in Appendix D.2.1.1, however only bits 1-7 of each octet are used. Bit 8 (the leftmost bit) set to one is used to indicate that this octet is *not* the last octet in the string.

D.2.1.7. Sequences

A sequence $D_1 \cdots D_n$ is an ordered list of n data objects D_j with $1 \leq j \leq n$. The sequence SHALL be converted to a concatenated list of octet strings $O_1 \cdots O_n$ by DER encoding each data object D_j to an octet string O_j.

D.3. Public Key Data Objects

A public key data object is a sequence consisting of an object identifier and several context specific data objects:

- The object identifier is application specific and refers not only to the public key format (i.e. the context specific data objects) but also to its usage.
- The context specific data objects are defined by the object identifier and contain the public key value and the domain parameters.

The format of public keys data objects used in this specification is described below.

D.3.1. RSA Public Keys

The data objects contained in an RSA public key are shown in Table D.3. The order of the data objects is fixed.

Data Object	**Abbrev.**	**Tag**	**Type**	**CV Certificate**
Object Identifier		0x06	Object Identifier	m
Composite modulus	n	0x81	Unsigned Integer	m
Public exponent	e	0x82	Unsigned Integer	m

Table D.3: RSA Public Key

D.3.2. Diffie Hellman Public Keys

The data objects contained in a DH public key are shown in Table D.4. The order of the data objects is fixed.

Data Object	**Abbrev.**	**Tag**	**Type**
Object Identifier		0x06	Object Identifier
Prime modulus	p	0x81	Unsigned Integer
Order of the subgroup	q	0x82	Unsigned Integer
Generator	g	0x83	Unsigned Integer
Public value	y	0x84	Unsigned Integer

Table D.4: DH Public Key

D.3.3. Elliptic Curve Public Keys

The data objects contained in an EC public key are shown in Table D.5. The order of the data objects is fixed, CONDITIONAL domain parameters MUST be either all present, except the cofactor, or all absent as follows:

Data Object	**Abbrev.**	**Tag**	**Type**	**CV Certificate**
Object Identifier		0x06	Object Identifier	m
Prime modulus	p	0x81	Unsigned Integer	c
First coefficient	a	0x82	Unsigned Integer	c
Second coefficient	b	0x83	Unsigned Integer	c
Base point	G	0x84	Elliptic Curve Point	c
Order of the base point	r	0x85	Unsigned Integer	c
Public point	Y	0x86	Elliptic Curve Point	m
Cofactor	f	0x87	Unsigned Integer	c

Table D.5: EC Public Keys

- CVCA Link Certificates MAY contain domain parameters.

- DV and Terminal Certificates MUST NOT contain domain parameters. The domain parameters of DV and terminal public keys SHALL be inherited from the respective CVCA public key.
- Certificate Requests MUST always contain domain parameters.

D.3.4. Ephemeral Public Keys

For ephemeral public keys the format and the domain parameters are already known. Therefore, only the plain public key value, i.e. the public value y for Diffie-Hellman public keys and the public point Y for Elliptic Curve Public Keys, is used to convey the ephemeral public key in a context specific data object.

E. eID Application (Normative)

E.1. eID Application

The eID application consists of 21 data groups (DG1 - DG21) containing personal data. An overview on the data groups is given in Table E.1.

E.1.1. Application Identifier

The eID application SHALL be identified by the standard application identifier 0xE80704007F00070302 that is based on the following object identifier:

```
id-eID OBJECT IDENTIFIER ::= {
  bsi-de applications(3) 2
}
```

E.2. ASN.1 Definition

Each elementary file contains an ASN.1-structure as defined below. The data is encoded according to the Distinguished Encoding Rules (DER) as specified in [17].

```
DocumentType     ::= [APPLICATION 1]  ICAOString (SIZE (2))
IssuingState     ::= [APPLICATION 2]  ICAOCountry
DateOfExpiry     ::= [APPLICATION 3]  Date
GivenNames       ::= [APPLICATION 4]  UTF8String
FamilyNames      ::= [APPLICATION 5]  UTF8String
ArtisticName     ::= [APPLICATION 6]  UTF8String
AcademicTitle    ::= [APPLICATION 7]  UTF8String
DateOfBirth      ::= [APPLICATION 8]  Date
PlaceOfBirth     ::= [APPLICATION 9]  Place (WITH COMPONENTS{
                                        city,state,country})
Nationality      ::= [APPLICATION 10] ICAOCountry
Sex              ::= [APPLICATION 11] ICAOSex
OptionalDataR    ::= [APPLICATION 12] SET OF OptionalData

PlaceOfResidence ::= [APPLICATION 17] Place
CommunityID      ::= [APPLICATION 18] OCTET STRING
ResidencePermitI ::= [APPLICATION 19] Text
ResidencePermitII::= [APPLICATION 20] Text
OptionalDataRW   ::= [APPLICATION 21] SET OF OptionalData

ICAOString ::= PrintableString (FROM ("A".. "Z" | " "))

ICAOCountry ::= ICAOString (SIZE (1|3))  -- ICAO country code

ICAOSex ::= PrintableString (FROM ("M"|"F"|" "))

Date ::= NumericString (SIZE (8)) -- YYYYMMDD
```

```
Place ::= SEQUENCE {
  street     [10] UTF8String OPTIONAL,
  city       [11] UTF8String,
  state      [12] UTF8String OPTIONAL,
  country    [13] ICAOCountry
}

Text ::= CHOICE {
  uncompressed [1] UTF8String
  compressed   [2] OCTET STRING
      -- contains a DEFLATE-compressed UTF8String (cf. [5] for details on
      -- the compression algorithm)
}

OptionalData ::= SEQUENCE {
  type OBJECT IDENTIFIER,
  data ANY DEFINED BY type OPTIONAL
}
```

DG	Content	FID	SFID	ASN.1 type	R/W	Access
DG1	Document Type	0x0101	0x01	DocumentType	R	PACE + TA + CA
DG2	Issuing State	0x0102	0x02	IssuingState	R	PACE + TA + CA
DG3	Date of Expiry	0x0103	0x03	DateOfExpiry	R	PACE + TA + CA
DG4	Given Names	0x0104	0x04	GivenNames	R	PACE + TA + CA
DG5	Family Names	0x0105	0x05	FamilyNames	R	PACE + TA + CA
DG6	Religous / Artistic Name	0x0106	0x06	ArtisticName	R	PACE + TA + CA
DG7	Academic Title	0x0107	0x07	AcademicTitle	R	PACE + TA + CA
DG8	Date of Birth	0x0108	0x08	DateOfBirth	R	PACE + TA + CA
DG9	Place of Birth	0x0109	0x09	PlaceOfBirth	R	PACE + TA + CA
DG10	Nationality	0x010A	0x0A	Nationality	R	PACE + TA + CA
DG11	Sex	0x010B	0x0B	Sex	R	PACE + TA + CA
DG12	Optional Data	0x010C	0x0C	OptionalDataR	R	PACE + TA + CA
DG13	--	0x010D	0x0D	RFU	R	PACE + TA + CA
DG14	--	0x010E	0x0E	RFU	R	PACE + TA + CA
DG15	--	0x010F	0x0F	RFU	R	PACE + TA + CA
DG16	--	0x0110	0x10	RFU	R	PACE + TA + CA
DG17	Normal Place of Residence	0x0111	0x11	PlaceOfResidence	R/W	PACE + TA + CA
DG18	Community ID	0x0112	0x12	CommunityID	R/W	PACE + TA + CA
DG19	Residence Permit I	0x0113	0x13	Residence PermitI	R/W	PACE + TA + CA
DG20	Residence Permit II	0x0114	0x14	ResidencePermitII	R/W	PACE + TA + CA
DG21	Optional Data	0x0115	0x15	OptionalDataRW	R/W	PACE + TA + CA

Table E.1: Data Groups of the eID Application

F. Secure Messaging (Normative)

As this guideline only considers command APDUs with even instruction byte, Appendix F solely takes into account Secure Messaging for command/response pairs where the command APDU has an even INS byte.

F.1. Message Structure of Secure Messaging APDUs

Secure Messaging Data Objects SHALL be used according to Table F.1 in the following order:

- Command APDU: [DO‘87’] [DO‘97’] DO‘8E’
- Response APDU: [DO‘87’] DO‘99’ DO‘8E’

All secure messaging data objects SHALL be encoded in DER (cf. Appendix D.2). The actual value of Lc will be modified to Lc’ after application of secure messaging. If required, an appropriate data object may optionally be included into the APDU data part in order to convey the original value of Le. In the protected command APDU the *new Le* byte SHALL be set to ‘00’.

Note: Secure messaging MUST be indicated by using class byte CLA = ‘XC’, with a bit mask X, where bit 8 (set to 0) indicates the interindustry class and bit 5 (set to 1) indicates command chaining.

F.1.1. Command APDU

The command with applied Secure Messaging therefore SHALL have the following structure, depending on the case of the respective unsecured command:

Case 1: CH || Lc’ || DO’8E’ || new Le

Case 2: CH || Lc’ || DO’97’ || DO’8E’ || new Le

Case 3: CH || Lc’ || DO’87’ || DO’8E’ || new Le

Case 4: CH || Lc’ || DO’87’ || DO’97’ || DO’8E’ || new Le

with CH: Command Header (CLA INS P1 P2)

Name	Tag	Len	Command	Response
Padding-content indicator byte followed by cryptogram	0x87	V	c	c
Protected Le	0x97	2V	c	x
Processing Status	0x99	2F	x	m
Cryptographic Checksum	0x8E	8F	m	m

F: fixed length (exact number of octets), V: variable length (up to number of octets)

Table F.1: Usage of Secure Messaging Data Objects

F.1.2. Response APDU

The response with applied Secure Messaging SHALL have the following structure, depending on the case of the respective unsecured command:

Case 1: DO’99’ || DO’8E’ || SW1SW2

Case 2: DO’87’ || DO’99’ || DO’8E’ || SW1SW2

Case 3: DO’99’ || DO’8E’ || SW1SW2

Case 4: DO’87’ || DO’99’ || DO’8E’ || SW1SW2

F.1.3. Padding

The data to be encrypted SHALL be padded according to ISO 7816-4 [13] using padding-content indicator 0x01. For the calculation of the cryptographic checksum the APDU SHALL be padded according to ISO 7816-4 [13].

Note: Padding is always performed by the secure messaging layer not by the underlying cryptographic algorithm.

F.1.4. Examples

Three Examples are provided at the end of this section:

- Figure F.1 shows the transformation of an unprotected command APDU to a protected command APDU in the case Data and/or Le are available. If no Data is available, leave building DO ’87’ out. If Le is not available, leave building DO ’97’ out.
- Figure F.2 shows the transformation of an unprotected command APDU to a protected command APDU in the case Data and Le are not available.
- Figure F.3 shows the transformation of an unprotected response APDU to a protected response APDU in the case Data are available. If no Data is available, leave building DO ’87’ out.

F.2. Cryptographic Algorithms

Secure Messaging is based on either 3DES [19] or AES [20] in encrypt-then-authenticate mode, i.e. data is encrypted first and afterwards the formatted encrypted data is input to the authentication calculation. The session keys SHALL be derived from PACE or Chip Authentication using the key derivation function described in Appendix A.2.3.

Note: If a command does not contain command data, no encryption applies for the command. If a response does not contain response data, no encryption applies for the response.

F.2.1. 3DES

3DES is specified in [19].

F.2.1.1. 3DES Encryption

For message encryption two key 3DES SHALL be used in CBC-mode according to ISO 10116 [12] with key K_{Enc} and $IV = 0$.

F.2.1.2. 3DES Authentication

For message authentication 3DES SHALL be used in Retail-mode according to ISO/IEC 9797-1 [16] MAC algorithm 3 with block cipher DES, key K_{MAC} and $IV = 0$. The datagram to be authenticated SHALL be prepended by the Send Sequence Counter.

F.2.2. AES

AES is specified in [20].

F.2.2.1. AES Encryption

For message encryption AES SHALL be used in CBC-mode according to ISO 10116 [12] with key K_{Enc} and $IV = \mathbf{E}(K_{Enc}, SSC)$.

F.2.2.2. AES Authentication

For message authentication AES SHALL be used in CMAC-mode [22] with K_{MAC} with a MAC length of 8 bytes. The datagram to be authenticated SHALL be prepended by the Send Sequence Counter.

F.3. Send Sequence Counter

An unsigned integer SHALL be used as Send Sequence Counter (SSC). The bitsize of the SSC SHALL be equal to the blocksize of the block cipher used for Secure Messaging, i.e. 64 bit for 3DES and 128 bit for AES.

The SSC SHALL be increased every time before a command or response APDU is generated, i.e. if the starting value is x, in the next command the value of the SSC is $x+1$. The value of SSC for the first response is $x+2$.

If Secure Messaging is restarted, the SSC is used as follows:

- The commands used for key agreement are protected with the old session keys and old SSC. This applies in particular for the response of the last command used for session key agreement.
- The Send Sequence Counter is set to its new start value, i.e. within this specification the SSC is set to 0.
- The new session keys and the new SSC are used to protect subsequent commands/responses.

F.4. Secure Messaging Errors

The MRTD chip MUST abort Secure Messaging if and only if a Secure Messaging error occurs:

- If expected Secure Messaging data objects are missing, the MRTD chip SHALL respond with status bytes 0x6987
- If Secure Messaging data objects are incorrect, the MRTD chip SHALL respond with status bytes 0x6988

If Secure Messaging is aborted, the MRTD chip SHALL delete the stored session keys and reset the terminal's access rights.

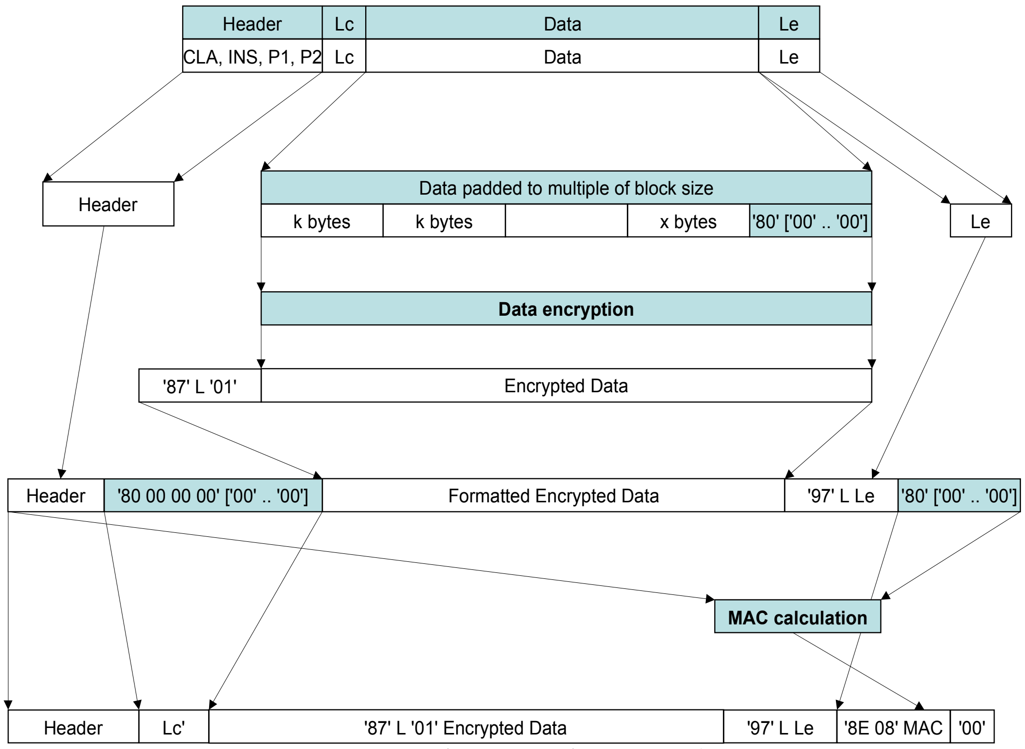

Figure F.1: Transformation of a command APDU

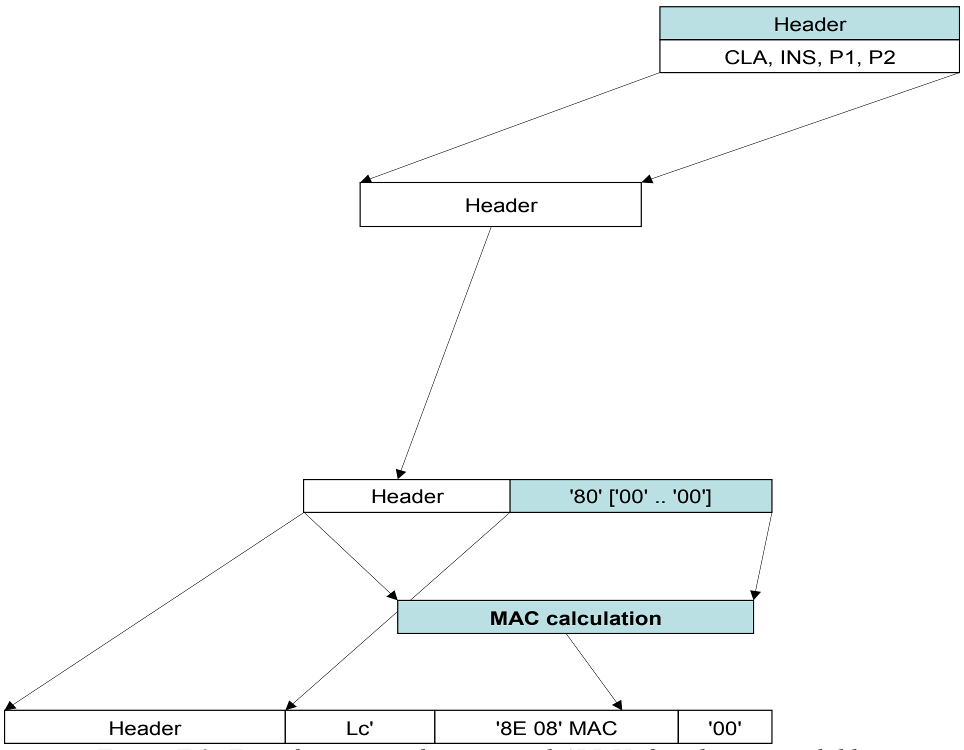

Figure F.2: Transformation of a command APDU if no data is available

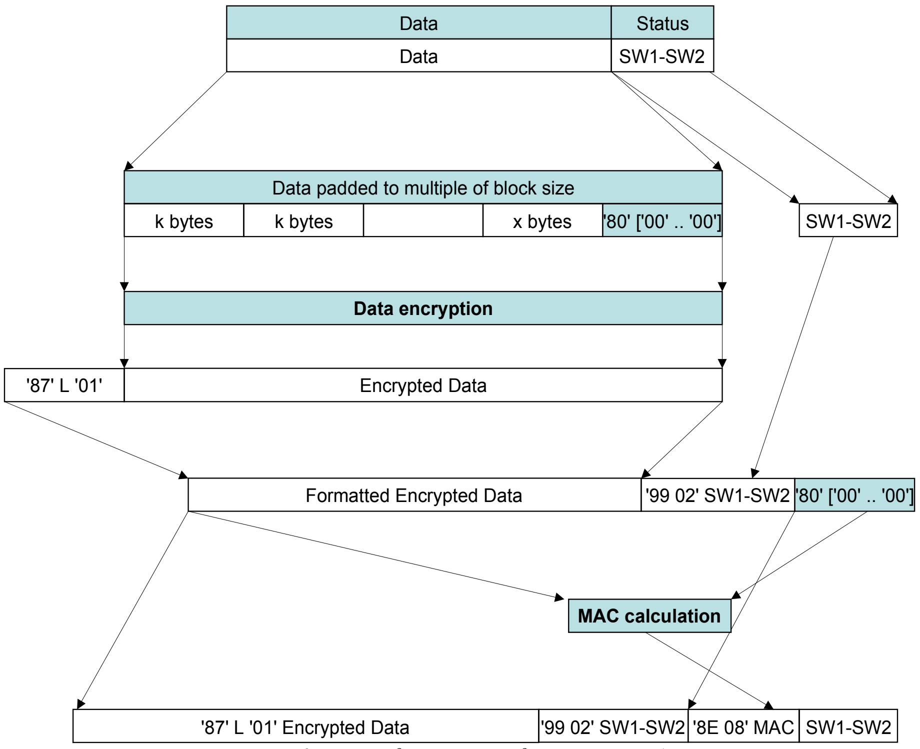

Figure F.3: Transformation of a response APDU

G. ICAO ePassport Procedures (Normative)

This appendix specifies the inspection procedures for an ICAO compliant ePassport application.

Depending on whether or not a device (i.e. an MRTD chip or a terminal) is compliant to this specification the device is called *compliant* or *non-compliant*, respectively. Depending on the combination of a terminal and an MRTD chip, either the *standard inspection procedure* or the *advanced inspection procedure* is used:

- A non-compliant inspection system uses the standard inspection procedure. Less-sensitive data stored on a compliant MRTD chip MUST be readable by every non-compliant inspection system.
- A compliant inspection system SHALL use the advanced inspection procedure if the MRTD chip is compliant. Otherwise the standard inspection procedure SHALL be used.

Table G.1 gives an overview on the inspection procedures to be used.

A terminal may use either the standard inspection procedure to access less-sensitive data contained in the ePassport application or the advanced inspection procedure to access less-sensitive and sensitive data in the ePassport application.

For the standard inspection procedure the MRZ SHOULD be known to the terminal (as Basic Access Control is RECOMMENDED for MRTD chips). Either the MRZ or the CAN MUST be known to the terminal for the advanced inspection procedure.

Note: As described in Section 1.1 Passive Authentication is a continuous process that requires the computation of a hash value of each data group read from the chip and its comparison to the corresponding hash value contained in the Security Object. While this continuous process is assumed to be applied in the following procedures, it is not explicitly described.

G.1. Standard ePassport Inspection Procedure

The standard inspection procedure can be used for all ICAO-compliant ePassport applications and consists of the following steps:

1. **Select ePassport application** **(REQUIRED)**

   The MRTD chip performs the following:

   - *BAC required:* It SHALL NOT grant access to any data (except general system data).
   - *BAC not required:* It SHALL grant access to less-sensitive data (e.g. DG1, DG2, DG14, DG15, etc. and the Security Object).

Inspection System	**MRTD chip**	
	compliant	**non-compliant**
compliant	Advanced	Standard
non-compliant	Standard	Standard

Table G.1: Inspection Procedures

2. **Basic Access Control** **(CONDITIONAL)**

   This step is REQUIRED if Basic Access Control is enforced by the MRTD chip.

   If successful, the MRTD chip performs the following:

   - It SHALL start Secure Messaging.
   - It SHALL grant access to less-sensitive data (e.g. DG1, DG2, DG14, DG15, etc. and the Security Object).
   - It SHALL restrict access rights to require Secure Messaging.

3. **Passive Authentication (started)** **(REQUIRED)**

   The terminal MUST read and verify the Security Object.

4. **Active Authentication** **(OPTIONAL)**

   If available, the terminal MAY read and verify DG15 and perform Active Authentication.

5. **Read and authenticate data**

   The terminal MAY read and verify read data groups containing less-sensitive data.

G.2. Advanced ePassport Inspection Procedure

The advanced inspection procedure can only be used for EAC-compliant ePassport applications and consists of the following steps:

1. **Select ePassport application** **(REQUIRED)**

   The MRTD chip SHALL NOT grant access to any data (except general system data).

2. **Basic Access Control or PACE** **(REQUIRED)**

   The MRTD chip SHALL accept the keys derived from the MRZ for BAC and following PINs for PACE: MRZ, CAN

   If successful, the MRTD chip performs the following:

   - It SHALL start Secure Messaging.
   - It SHALL grant access to DG14 (containing the Chip Authentication Public Key).
   - It MAY grant access to less-sensitive data (e.g. DG1, DG2, DG15, etc. and the Security Object).[5]
   - It SHALL restrict access rights to require Secure Messaging.

3. **Chip Authentication** **(REQUIRED)**

   The terminal SHALL read DG14 and perform Chip Authentication.

   The MRTD chip performs the following:

   - It SHALL restart Secure Messaging.

5 For an ICAO-compliant ePassport application the MRTD chip MUST grant access to all less-sensitive data. If compliance to ICAO standards is not required, the MRTD chip MAY deny access to certain data groups until Chip Authentication is performed.

- It SHALL grant access to less-sensitive data (e.g. DG1, DG2, DG15, etc. and the Security Object).
- It SHALL restrict access rights to require Secure Messaging established by Chip Authentication.

4. **Passive Authentication (started) (REQUIRED)**

   The terminal performs the following:

   - It SHALL read and verify the Security Object.
   - It SHALL verify DG14.

5. **Active Authentication (OPTIONAL)**

   If available, the terminal MAY read and verify DG15 and perform Active Authentication.

6. **Terminal Authentication (CONDITIONAL)**

   This step is REQUIRED to access sensitive ePassport data.

   If successful the MRTD chip performs the following:

   - It SHALL additionally grant access to data groups according the terminal's access rights.
   - It SHALL grant access to the eID application according to the terminal's access rights.
   - It SHALL restrict all access rights to require Secure Messaging established by Chip Authentication using the ephemeral public key authenticated by Terminal Authentication.

7. **Read and authenticate data**

   The terminal MAY read and verify read data groups according to the terminal's access rights.

H. Basic Access Control (Informative)

The protocol for Basic Access Control is specified by ICAO [8], [9]. Basic Access Control checks that the terminal has *physical* access to the MRTD's data page. This is enforced by requiring the terminal to derive an authentication key from the *optically* read MRZ of the MRTD. The protocol for Basic Access Control is based on ISO/IEC 11770-2 [10] key establishment mechanism 6. This protocol is also used to generate session keys that are used to protect the confidentiality (and integrity) of the transmitted data.

H.1. Document Basic Access Keys

The Document Basic Access Keys KB_{Enc} and KB_{MAC} stored on the RF-chip in secure memory, have to be derived by the terminal from the MRZ of the MRTD prior to accessing the RF-chip. Therefore, the terminal optically reads the MRZ and generates the Document Basic Access Keys by applying the ICAO KDF [8], [9] to the most significant 16 bytes of the SHA-1 [21] hash of some fields of the MRZ. As reading the MRZ optically is error-prone, only the fields protected by a check-digit are used to generate the Basic Access Key(s): Document Number, Date of Birth, and Date of Expiry. As a consequence the resulting authentication key has a relatively low entropy. The actual entropy mainly depends on the type of the Document Number. For 10 year valid travel document the **maximum** strength of the authentication key is approximately:

- 56 Bit for a numeric Document Number ($365^2 \cdot 10^{12}$ possibilities)
- 73 Bit for an alphanumeric Document Number ($365^2 \cdot 36^9 \cdot 10^3$ possibilities)

Especially in the second case this estimation requires the Document Number to be randomly and uniformly chosen. Depending on the knowledge of the attacker, the actual entropy of the Document Basic Access Key may be lower, e.g. if the attacker knows all Document Numbers in use or is able to correlate Document Numbers and Dates of Expiry.

Given that in the first case the maximum entropy (56 Bit) is relatively low, calculating the authentication key from an eavesdropped session is possible. On the other hand, this still requires more effort than to obtain the same (less-sensitive) data from another source.

H.2. Protocol Specification

Basic Access Control is shown in Figure H.1. For better readability encryption and message authentication are combined into a single authenticated encryption primitive

$$\mathbf{EM}(K,S)=\mathbf{E}(KB_{Enc},S)\|\mathbf{MAC}(K_{MAC},\mathbf{E}(KB_{Enc},S)),$$

where $K=\{KB_{Enc},KB_{MAC}\}$. The corresponding operation $\mathbf{DM}(K,C)$ is defined analogous, i.e. as verification and decryption.

1. The MRTD chip sends the nonce r_{PICC} to the terminal.
2. The terminal sends the encrypted challenge

$$e_{PCD}=\mathbf{EM}(K,r_{PCD}\|r_{PICC}\|K_{PCD})$$

to the MRTD chip, where r_{PICC} is the MRTD chip's nonce, r_{PCD} is the terminal's randomly chosen nonce, and K_{PCD} is keying material for the generation of the session keys.

3. The MRTD chip performs the following actions:
   a) It decrypts the received challenge to $r'_{PCD} \| r'_{PICC} \| K'_{PCD} = \mathbf{DM}(K, e_{PCD})$ and verifies that $r'_{PICC} = r_{PICC}$.
   b) It responds with the encrypted challenge $e_{PICC} = \mathbf{EM}(K, r_{PICC} \| r'_{PCD} \| K_{PICC})$, where r_{PICC} is the MRTD chip's randomly chosen nonce and K_{PICC} is keying material for the generation of the session keys.
4. The terminal decrypts the encrypted challenge to $r'_{PICC} \| r''_{PCD} \| K'_{PICC} = \mathbf{DM}(K, e_{PICC})$ and verifies that $r''_{PCD} = r_{PCD}$.
5. After a successful authentication all further communication MUST be protected by Secure Messaging in Encrypt-then-Authenticate mode using session keys K_{Enc} and K_{MAC} derived according to [8], [9] from the common master secret $K_{Master} = K_{PICC} \oplus K_{PCD}$ and a Send Sequence Counter SSC derived from r_{PICC} and r_{PCD}.

MRTD Chip (PICC)		**Terminal (PCD)**
		Read MRZ optically and derive K
Choose r_{PICC} and K_{PICC} randomly		Choose r_{PCD} and K_{PCD} randomly
	$\xrightarrow{r_{PICC}}$	
	$\xleftarrow{e_{PCD}}$	$e_{PCD} = \mathbf{EM}(K, r_{PCD} \| r_{PICC} \| K_{PCD})$
$r'_{PCD} \| r'_{PICC} \| K'_{PCD} = \mathbf{DM}(K, e_{PCD})$		
Check $r'_{PICC} = r_{PICC}$		
$e_{PICC} = \mathbf{EM}(K, r_{PICC} \| r'_{PCD} \| K_{PICC})$	$\xrightarrow{e_{PICC}}$	
		$r'_{PICC} \| r''_{PCD} \| K'_{PICC} = \mathbf{DM}(K, e_{PICC})$
		Check $r''_{PCD} = r_{PCD}$

Figure H.1: Basic Access Control

I. Challenge Semantics (Informative)

Consider a signature based challenge-response protocol between an MRTD chip (PICC) and a terminal (PCD), where the MRTD chip wants to prove knowledge of its private key SK_{PICC} :

1. The terminal sends a randomly chosen challenge c to the MRTD chip.
2. The MRTD chip responds with the signature $s = \mathbf{Sign}(SK_{PICC}, c)$.

While this is a very simple and efficient protocol, the MRTD chip in fact signs the message c without knowing the semantic of this message. As signatures provide a transferable proof of authenticity, any third party can – in principle – be convinced that the MRTD chip has indeed signed this message.

Although c should be a random bit string, the terminal can as well generate this bit string in an unpredictable but (publicly) verifiable way, e.g. let SK_{PCD} be the terminal's private key and

$$c = \mathbf{Sign}(SK_{PCD}, ID_{PICC} || \text{Date} || \text{Time} || \text{Location})$$

be the challenge generated by using a signature scheme with message recovery. The signature guarantees that the terminal has indeed generated this challenge. Due to the transferability of the terminal's signature, any third party having trust in the terminal and knowing the corresponding public key PK_{PCD} can check that the challenge was created correctly by verifying this signature. Furthermore, due to the transferability of MRTD chip's signature on the challenge, the third party can conclude that the assertion became true: The MRTD chip was indeed at a certain date and time at a certain location.

On the positive side, countries may use Challenge Semantics for their internal use, e.g. to prove that a certain person indeed has immigrated. On the negative side such proves can be misused to track persons. In particular since Active Authentication is not restricted to authorized terminals misuse is possible. The worst scenario would be MRTD chips that provide Active Authentication without Basic Access Control. In this case a very powerful tracking system may be set up by placing secure hardware modules at prominent places. The resulting logs cannot be faked due to the signatures. Basic Access Control diminishes this problem to a certain extent, as interaction with the bearer is required. Nevertheless, the problem remains, but is restricted to places where the travel document of the bearer is read anyway, e.g. by airlines, hotels etc.

One might object that especially in a contactless scenario, challenges may be eavesdropped and reused at a different date, time or location and thus render the proof at least unreliable. While eavesdropping challenges is technically possible, the argument is still invalid. By assumption a terminal is trusted to produce challenges correctly and it can be assumed that it has checked the MRTD chip's identity before starting Active Authentication. Thus, the eavesdropped challenge will contain an identity different from the prover's identity who signs the challenge.

Bibliography

[1] ANSI. Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography, ANSI X9.42-2000, 1999

[2] Bradner, Scott. Key words for use in RFCs to indicate requirement levels, RFC 2119, 1997

[3] BSI. eCard-API-Framework - ISO24727-3 Interface Version 1.1, TR-03112-4, 2008

[4] BSI. Elliptic Curve Cryptography (ECC) Version 1.11, TR-03111, 2009

[5] Deutsch, Peter. DEFLATE compressed data format specification version 1.3., RFC 1951, 1996

[6] Housley, Russel. Cryptographic message syntax (CMS), RFC 3852, 2004

[7] Housley, Russel; Polk,Tim; Ford, Warwick and Solo, David. Internet X.509 public key infrastructure - certificate and certificate revocation list (CRL) profile, RFC 3280, 2002

[8] ICAO, Machine Readable Travel Documents - Part 1: Machine Readable Passport, Specifications for electronically enabled passports with biometric identification capabilities, ICAO Doc 9303, 2006

[9] ICAO, Machine Readable Travel Documents - Part 3: Machine Readable Official Travel Documents, Specifications for electronically enabled official travel documents with biometric identification capabilities, ICAO Doc 9303, 2008

[10] ISO 11770-2. Information technology − Security techniques − Key management − Part 2: Mechanisms using symmetric techniques, 1996

[11] ISO 32000-1:2008. Document management − Portable document format − Part 1: PDF 1.7, 2008

[12] ISO/IEC 10116:2006. Information technology − Security techniques − Modes of operation for an n-bit block cipher, 2006

[13] ISO/IEC 7816-4:2005. Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange, 2005

[14] ISO/IEC 7816-6:2004. Identification cards – Integrated circuit cards – Part 6: Interindustry data elements for interchange, 2004

[15] ISO/IEC 7816-8:2004. Identification cards – Integrated circuit cards – Part 8: Commands for security operations, 2004

[16] ISO/IEC 9797-1:1999. Information technology − Security techniques − Message Authentication Codes (MACs) − Part 1: Mechanisms using a block cipher, 1999

[17] ITU-T. Information Technology – ASN.1 encoding rules: Specification of Basic Encoding Rules(BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), X.690, 2002

[18] Jonsson, Jakob and Kaliski, Burt. Public-key cryptography standards (PKCS)#1: RSA cryptography specifications version 2.1, RFC 3447, 2003

[19] NIST. Data Encryption Standard (DES), FIPS PUB 46-3, 1999

[20] NIST. Specification for the Advanced Encryption Standard (AES), FIPS PUB 197, 2001

[21] NIST. Secure hash standard (and Change Notice to include SHA-224), FIPS PUB 180-2, 2002

[22] NIST. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, Special Publication 800-38B, 2005

[23] Rescorla, Eric. Diffie-Hellman key agreement method, RFC 2631, 1999

[24] RSA Laboratories. PKCS#3: Diffie-Hellman key-agreement standard, RSA Laboratories Technical Note, 1993

[25] RSA Laboratories. PKCS#1 v2.1: RSA cryptography standard, RSA Laboratories Technical Note, 2002

[26] Thomas Icart. Hashing onto Elliptic Curves, 2009