ICAO

Doc 9303

Machine Readable Travel Documents

Seventh Edition, 2015

Part 11: Security Mechanisms for MRTDs

Approved by the Secretary General and published under his authority

INTERNATIONAL CIVIL AVIATION ORGANIZATION

ICAO

Doc 9303

Machine Readable Travel Documents

Seventh Edition, 2015

Part 11: Security Mechanisms for MRTDs

Approved by the Secretary General and published under his authority

INTERNATIONAL CIVIL AVIATION ORGANIZATION

Published in separate English, Arabic, Chinese, French, Russian
and Spanish editions by the
INTERNATIONAL CIVIL AVIATION ORGANIZATION
999 Robert-Bourassa Boulevard, Montréal, Quebec, Canada H3C 5H7

Downloads and additional information are available at www.icao.int/security/mrtd

Doc 9303, ***Machine Readable Travel Documents***
Part 11 — ***Security Mechanisms for MRTDs***
ISBN 978-92-9249-799-6

AMENDMENTS

Amendments are announced in the supplements to the *Products and Services Catalogue;* the Catalogue and its supplements are available on the ICAO website at www.icao.int. The space below is provided to keep a record of such amendments.

RECORD OF AMENDMENTS AND CORRIGENDA

AMENDMENTS		
No.	Date	Entered by

CORRIGENDA		
No.	Date	Entered by

TABLE OF CONTENTS

			Page
1.	**SCOPE**		**1**
2.	**ASSUMPTIONS AND NOTATIONS**		**2**
	2.1	Requirements for MRTD Chips and Terminals	2
	2.2	Notations	2
3.	**SECURING ELECTRONIC DATA**		**3**
4.	**ACCESS TO THE CONTACTLESS IC**		**4**
	4.1	Compliant Configurations	5
	4.2	Chip Access Procedure	6
	4.3	Basic Access Control	7
	4.4	Password Authenticated Connection Establishment	10
5.	**AUTHENTICATION OF DATA**		**21**
	5.1	Passive Authentication	21
6.	**AUTHENTICATION OF THE CONTACTLESS IC**		**22**
	6.1	Active Authentication	23
	6.2	Chip Authentication	26
7.	**ADDITIONAL ACCESS CONTROL MECHANISMS**		**31**
	7.1	Extended Access Control for Additional Biometrics	31
	7.2	Encryption of Additional Biometrics	32
8.	**INSPECTION SYSTEM**		**32**
	8.1	Basic Access Control	32
	8.2	Password Authenticated Connection Establishment	32
	8.3	Passive Authentication	32
	8.4	Active Authentication	33
	8.5	Chip Authentication	33
	8.6	Extended Access Control to Additional Biometrics	33
	8.7	Decryption of Additional Biometrics	33
9.	**COMMON SPECIFICATIONS**		**34**
	9.1	ASN.1 Structures	34
	9.2	Information on Supported Protocols	34
	9.3	APDUs	40
	9.4	Public Key Data Objects	41

Page

9.5 Domain Parameters 42
9.6 Key Agreement Algorithms 44
9.7 Key Derivation Mechanism 44
9.8 Secure Messaging 46

10. REFERENCES (NORMATIVE) 51

APPENDIX A TO PART 11.— ENTROPY OF MRZ-DERIVED ACCESS KEYS (INFORMATIVE) .... App A-1

APPENDIX B TO PART 11.— POINT ENCODING FOR THE ECDH-INTEGRATED MAPPING (INFORMATIVE) .... App B-1

B.1 High-level Description of the Point Encoding Method .... App B-1
B.2 Implementation for Affine Coordinates .... App B-1
B.3 Implementation for Jacobian Coordinates .... App B-2

APPENDIX C TO PART 11.— CHALLENGE SEMANTICS (INFORMATIVE) .... App C-1

APPENDIX D TO PART 11.— WORKED EXAMPLE: BASIC ACCESS CONTROL (INFORMATIVE) .... App D-1

D.1 Compute Keys from Key Seed (K_{seed}) .... App D-1
D.2 Derivation of Document Basic Access Keys (K_{Enc} and K_{MAC}) .... App D-2
D.3 Authentication and Establishment of Session Keys .... App D-3
D.4 Secure Messaging .... App D-5

APPENDIX E TO PART 11.— WORKED EXAMPLE: PASSIVE AUTHENTICATION (INFORMATIVE) .... App E-1

APPENDIX F TO PART 11.— WORKED EXAMPLE: ACTIVE AUTHENTICATION (INFORMATIVE) .... App F-1

APPENDIX G TO PART 11.— WORKED EXAMPLE: PACE – GENERIC MAPPING (INFORMATIVE) .... App G-1

G.1 ECDH based example .... App G-1
G.2 DH based example .... App G-10

APPENDIX H TO PART 11.— WORKED EXAMPLE: PACE – INTEGRATED MAPPING (INFORMATIVE) .... App H-1

H.1 ECDH based example .... App H-1
H.2 DH based example .... App H-4

1. SCOPE

The Seventh Edition of Doc 9303 represents a restructuring of the ICAO specifications for Machine Readable Travel Documents. Without incorporating substantial modifications to the specifications, in this new edition Doc 9303 has been reformatted into a set of specifications for Size 1 Machine Readable Official Travel Documents (TD1), Size 2 Machine Readable Official Travel Documents (TD2), and Size 3 Machine Readable Travel Documents (TD3), as well as visas. This set of specifications consists of various separate documents in which general (applicable to all MRTDs) as well as MRTD form factor specific specifications are grouped.

This Part 11 of Doc 9303 is based on the Sixth Edition of Doc 9303, Part 1, *Machine Readable Passports*, Volume 2, *Specifications for Electronically Enabled Passports with Biometric Identification Capability* (2006) and the Third Edition of Doc 9303, Part 3, *Machine Readable Official Travel Documents*, Volume 2, *Specifications for Electronically Enabled MRtds with Biometric Identification Capability* (2008).

This Part 11 provides specifications to enable States and suppliers to implement cryptographic security features for electronic machine readable travel documents ("eMRTDs") offering contactless integrated circuit (IC) read-only access. Cryptographic protocols are specified to:

- prevent skimming of data from the contactless IC;
- prevent eavesdropping on the communication between contactless IC and reader;
- provide authentication of the data stored on the contactless IC based on the Public Key Infrastructure (PKI) described in Part 12; and
- provide authentication of the contactless IC itself.

Additional access control to sensitive data (i.e. secondary biometrics) is not specified in this edition of Doc 9303, but national schemes to protect these data are allowed. An interoperable specification is foreseen for future editions of Doc 9303.

The authentication of the data stored on the contactless IC is the basic security feature to enable the use of the IC for manual and/or automated inspection. This feature is therefore REQUIRED.

Implementation of a protocol to prevent skimming of the data stored on the contactless IC and to prevent eavesdropping on the communication between IC and terminal is RECOMMENDED.

Implementation of the other protocols is OPTIONAL, allowing the issuing State or organization to decide on the necessary set of security features according to national regulations/demands.

This Part should be read in conjunction with the following Parts of Doc 9303:

- Part 1 — *Introduction*;
- Part 10 — *Logical Data Structure (LDS) for Storage of Biometrics and Other Data in the Contactless Integrated Circuit (IC)*; and
- Part 12 — *Public Key Infrastructure for MRTDs*.

2. ASSUMPTIONS AND NOTATIONS

It is assumed that the reader of this document is familiar with the concepts and mechanisms offered by public key cryptography and public key infrastructures.

Whilst the use of public key cryptography techniques adds some complexity to the implementation of eMRTDs, such techniques add value in that they will provide front-line border control points with an additional measure to determine the authenticity of the eMRTD. It is assumed that the use of such a technique is not the sole measure for determining authenticity and it SHOULD NOT be relied upon as a single determining factor.

In the event that the data from the contactless IC cannot be used, for instance as a result of a certificate revocation or an invalid signature verification, or if the contactless IC was left intentionally blank (see Doc 9303-10), the eMRTD is not necessarily invalidated. In such cases a receiving State MAY rely on other document security features for validation purposes.

2.1 Requirements for MRTD Chips and Terminals

This Part of Doc 9303 specifies requirements for implementations of MRTD chips and terminals. While MRTD chips must comply with those requirements according to the terminology described in Doc 9303-1, requirements for terminals are to be interpreted as guidance, i.e. interoperability of MRTD chip and terminal are only guaranteed if the terminal complies with those requirements, otherwise the interaction with the MRTD chip will either fail or the behaviour of the MRTD chip is undefined. In general, the MRTD chip need not enforce requirements related to terminals unless the security of the MRTD chip is directly affected.

2.2 Notations

The following notations are used to denote cryptographic primitives in an algorithm independent way:

- Encryption of clear text S with symmetric key K: **E**(K, S);
- Decryption of cipher text C with symmetric key K: **D**(K, C);
- The operation for computing a hash over a message *m* is denoted by **H***(m)*.
- Computing a Message Authentication Code with symmetric key K over message M: **MAC**(K,M);
- Key agreement based on asymmetric key pairs (SK, PK) and (SK’, PK’) and domain parameters D: **KA**(SK,PK’,D) / **KA**(SK’,PK,D);
- Key derivation from a shared secret S: **KDF**(S).

3. SECURING ELECTRONIC DATA

Besides Passive Authentication by digital signatures, States MAY choose additional security, using more complex ways of securing the contactless IC and its data.

Accessing an eMRTD comprises the following steps:

1. Gain access to the contactless IC of the eMRTD (Section 4)
2. Authentication of data (Section 5)
3. Authentication of the chip (Section 6)
4. Additional access control mechanisms (Section 7)
5. Reading data (see Doc 9303-10).

Different protocols are available for the different steps. The exact configuration of an eMRTD is chosen by the issuing State or organization. The options given in Table 1 can be suitably combined to achieve additional security according to the requirements of issuers.

Table 1. Securing Electronic Data (Summary)

<table>
<tr><th>Method</th><th>Contactless IC</th><th>Inspection System</th><th>Benefits</th><th>Note</th></tr>
<tr><td colspan="5">BASELINE SECURITY METHOD</td></tr>
<tr><td>Passive Authentication (Section 5.1)</td><td>M</td><td>m</td><td>Proves that the contents of the SO_D and the LDS are authentic and not changed.</td><td>Does not prevent an exact copy or IC substitution.
Does not prevent unauthorized access.
Does not prevent skimming.</td></tr>
<tr><td colspan="5">ADVANCED SECURITY METHODS</td></tr>
<tr><td>Comparison of conventional MRZ(OCR-B) and IC-based MRZ(LDS)</td><td>n/a</td><td>o</td><td>Proves that contactless IC's content and physical eMRTD belong together.</td><td>Adds (minor) complexity.
Does not prevent an exact copy of contactless IC and conventional document.</td></tr>
<tr><td>Active Authentication (Section 6.1)</td><td>O</td><td>o</td><td rowspan="2">Prevents copying the SO_D and proves that it has been read from the authentic contactless IC.
Proves that the contactless IC has not been substituted.</td><td rowspan="2">Does not prevent unauthorized access.
Adds complexity.</td></tr>
<tr><td>Chip Authentication (Section 6.2)</td><td>O</td><td>o</td></tr>
</table>

<table>
<tr><th>Method</th><th>Contactless IC</th><th>Inspection System</th><th>Benefits</th><th>Note</th></tr>
<tr><td>Basic Access Control (BAC)
(Section 4.3)</td><td>r/c
(see also 4.1)</td><td>m
(see also 4.1)</td><td rowspan="2">Prevents skimming and misuse.
Prevents eavesdropping on the communications between eMRTD and inspection system (when used to set up encrypted session channel).</td><td rowspan="2">Does not prevent an exact copy or IC substitution (requires also copying of the conventional document).
Adds complexity.</td></tr>
<tr><td>Password Authenticated Connection Establishment (PACE)
(Section 4.4)</td><td>R
(see also 4.1)</td><td>r
(see also 4.1)</td></tr>
<tr><td>Extended Access Control
(Section 7.1)</td><td>o</td><td>o</td><td>Prevents unauthorized access to additional biometrics.
Prevents skimming of additional biometrics.</td><td>Requires additional key management.
Does not prevent an exact copy or IC substitution (requires also copying of the conventional document).
Adds complexity.</td></tr>
<tr><td>Data Encryption
(Section 7.2)</td><td>o</td><td>o</td><td>Secures additional biometrics.
Does not require processor-ICs.</td><td>Requires complex decryption key management.
Does not prevent an exact copy or IC substitution.
Adds complexity.</td></tr>
<tr><td colspan="5">m = REQUIRED, r = RECOMMENDED, o = OPTIONAL, c = CONDITIONAL, n/a = not applicable.</td></tr>
</table>

Note.— See Section 4 for details on compliant configurations of contactless ICs with respect to the implementation of Basic Access Control and Password Authenticated Connection Establishment.

Implementation of advanced security methods as listed in Table 1 does not affect ICAO compliance.

4. ACCESS TO THE CONTACTLESS IC

Adding a contactless IC without access control to an MRTD introduces two new attack possibilities:

- the data stored in the contactless IC can be electronically read without authorizing this reading of the document (skimming); and
- the unencrypted communication between a contactless IC and a reader can be eavesdropped within a distance of several metres.

While there are physical measures possible against skimming (e.g. shielding using a metal mesh in the cover of a passport booklet), these do not address eavesdropping. Therefore, it is understood that issuing States or organizations SHOULD choose to implement a Chip Access Control mechanism, i.e. an access control mechanism that in effect requires the knowledge of the bearer of the eMRTD that the data stored in the contactless IC is being read in a secure way. This Chip Access Control mechanism prevents skimming as well as eavesdropping.

A contactless IC that is protected by a Chip Access Control mechanism denies access to its contents unless the inspection system can prove that it is authorized to access the contactless IC. This proof is given in a cryptographic protocol, where the inspection system proves knowledge of the information derived from the data page.

The inspection system MUST be provided with this information prior to being able to read the contactless IC. The information has to be retrieved optically/visually from the eMRTD (e.g. from the MRZ). It also MUST be possible for an inspector to enter this information manually in the inspection system in case machine-reading of the information is not possible.

Assuming that the information from the data page cannot be obtained from an unviewed document (e.g. since the information is derived from the optically read MRZ), it is accepted that the eMRTD was knowingly handed over for inspection. Due to the encryption of the channel, eavesdropping on the communication would require a considerable effort.

This section defines two mechanisms for Chip Access Control:

- Basic Access Control (BAC, Section 4.3), which is based purely on symmetric cryptography; and
- Password Authenticated Connection Establishment (PACE, Section 4.4), which employs asymmetric cryptography to provide higher entropy session keys.

See also Appendix A for additional information on the strength of session keys.

4.1 Compliant Configurations

The following configurations comply with this specification:

- eMRTD chips implementing no Chip Access Control (“plain eMRTDs”);
- eMRTD chips implementing BAC only;
- eMRTD chips implementing PACE *and* BAC;
- Starting 1 January 2018, eMRTD chips implementing PACE only.

Note.— For global interoperability, States MUST NOT implement PACE without implementing Basic Access Control until 31 December 2017. Inspection Systems SHOULD implement and use PACE if provided by the eMRTD chip.

BAC may become deprecated in the future. In this case PACE will become the default access control mechanism.

Compliant inspection systems MUST support all compliant eMRTD configurations. If an eMRTD supports both PACE and BAC, the inspection system SHALL use either BAC or PACE but not both in the same session.

4.2 Chip Access Procedure

The chip access procedure to authenticate the inspection system consists of the following steps. If PACE is not supported by the inspection system, Steps 1 and 2 are skipped.

1. **Read EF.CardAccess** **(REQUIRED)**

   If PACE is supported by the eMRTD, the eMRTD chip MUST provide the parameters to be used for PACE in the file EF.CardAccess.

   If EF.CardAccess is available, the inspection system SHALL read the file EF.CardAccess (cf. Section 9.2.8) to determine the parameters (i.e. symmetric ciphers, key agreement algorithms, domain parameters, and mappings) supported by the eMRTD chip. The inspection system may select any of those parameters.

   If the file EF.CardAccess is not available or does not contain parameters for PACE, the inspection system SHOULD try to read the eMRTD with Basic Access Control (skip to Step 4).

2. **PACE** **(CONDITIONAL)**

   This step is RECOMMENDED if PACE is supported by the eMRTD chip.

   - The inspection system SHOULD derive the key K_π from the MRZ. It MAY use the CAN instead of the MRZ if the CAN is known to the inspection system.
   - The eMRTD chip SHALL accept the MRZ as passwords for PACE. It MAY additionally accept the CAN.
   - The inspection system and the eMRTD chip mutually authenticate using K_π and derive session keys KS_{Enc} and KS_{MAC}. The PACE protocol as described in Section 4.4 SHALL be used.

   If successful, the eMRTD 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 Document Security Object — for the definition of “sensitive data” see Doc 9303-1).
   - It SHALL restrict access rights to require Secure Messaging.

3. **Select eMRTD Application** **(REQUIRED)**

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

   This step is REQUIRED if Chip Access Control is enforced by the eMRTD chip and PACE has not been used. If PACE was successfully performed or if the eMRTD does not enforce Chip Access Control, this step is skipped.

   - The inspection system SHOULD derive the Document Basic Access Keys (K_{Enc} and K_{MAC}) from the MRZ.
   - The inspection system and the eMRTD chip mutually authenticate using the Document Basic Access Keys and derive session keys KS_{Enc} and KS_{MAC}.

If successful, the eMRTD 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 Document Security Object).
- It SHALL restrict access rights to require Secure Messaging.

The inspection system MUST verify the authenticity of the contents of the file EF.CardAccess (see above) using DG14.

4.3 Basic Access Control

4.3.1 Protocol Specification

Authentication and Key Establishment is provided by a three-pass challenge-response protocol according to [ISO/IEC 11770-2] Key Establishment Mechanism 6 using 3DES [FIPS 46-3] as block cipher. A cryptographic checksum according to [ISO/IEC 9797-1] MAC Algorithm 3 is calculated over and appended to the ciphertexts. The modes of operation described in Section 4.3.3 MUST be used. Exchanged nonces MUST be of size 8 bytes, exchanged keying material MUST be of size 16 bytes. The IFD (inspection system) and the contactless IC MUST NOT use distinguishing identifiers as nonces.

In more detail, IFD and IC SHALL perform the following steps:

1) The IFD requests a challenge RND.IC by sending the GET CHALLENGE command. The IC generates and responds with a nonce RND.IC.
2) The IFD performs the following operations:
   a) generate a nonce RND.IFD and keying material K.IFD.
   b) generate the concatenation S = RND.IFD || RND.IC || K.IFD.
   c) compute the cryptogram E_{IFD} = $\mathbf{E}(K_{Enc,}\ S)$.
   d) compute the checksum M_{IFD} = $\mathbf{MAC}(K_{MAC,}\ E_{IFD})$.
   e) send an EXTERNAL AUTHENTICATE command with mutual authenticate function using the data E_{IFD} || M_{IFD}.
3) The IC performs the following operations:
   a) check the checksum M_{IFD} of the cryptogram E_{IFD}.
   b) decrypt the cryptogram E_{IFD}.
   c) extract RND.IC from S and check if IFD returned the correct value.
   d) generate keying material K.IC.

e) generate the concatenation R = RND.IC || RND.IFD || K.IC.

f) compute the cryptogram E_{IC} = $\mathbf{E}(K_{Enc}, R)$.

g) compute the checksum M_{IC} = $\mathbf{MAC}(K_{MAC}, E_{IC})$.

h) send the response using the data E_{IC} || M_{IC}.

4) The IFD performs the following operations:

a) check the checksum M_{IC} of the cryptogram E_{IC}.

b) decrypt the cryptogram E_{IC}.

c) extract RND.IFD from R and check if IC returned the correct value.

5) The IFD and the IC derive session keys KS_{Enc} and KS_{MAC} using the key derivation mechanism described in Section 9.7.4 with (K.IC xor K.IFD) as key seed.

4.3.2 Inspection Process

When an eMRTD with Basic Access Control is offered to the inspection system, optically or visually read information is used to derive the Document Basic Access Keys (K_{Enc} and K_{MAC}) to gain access to the contactless IC and to set up a secure channel for communications between the eMRTD's contactless IC and the inspection system.

An eMRTD's contactless IC that supports Basic Access Control MUST respond to unauthenticated read attempts, i.e. read attempts sent without Secure Messaging (including selection of (protected) files in the LDS), with "Security status not satisfied" (0x6982) once the Secure Channel is established. If the IC receives a plain SELECT, i.e. without Secure Messaging applied, in the Secure Channel, the IC SHALL abort the Secure Channel. When a plain SELECT is sent before the Secure Channel is established, or when the Secure Channel has been aborted, both 0x6982 and 0x9000 MAY be returned by the IC, i.e., are ICAO-compliant responses.

To authenticate the inspection system the following steps MUST be performed:

1) The inspection system reads the "MRZ_information". The "MRZ_information" consists of the concatenation of Document Number, Date of Birth and Date of Expiry, including their respective check digits, as described in Doc 9303-4, Doc 9303-5 or Doc 9303-6 for document form factors TD3, TD1 and TD2, respectively, from the MRZ using an OCR-B reader. Alternatively, the required information can be typed in; in this case it SHALL be typed in as it appears in the MRZ. The most significant 16 bytes of the SHA-1 hash of this "MRZ_information" are used as key seed to derive the Document Basic Access Keys using the key derivation mechanism described in Section 9.7.2.

2) The inspection system and the eMRTD's contactless IC mutually authenticate and derive session keys. The authentication and key establishment protocol described above MUST be used.

3) After a successful execution of the authentication protocol both the IFD and the IC compute session keys KS_{Enc} and KS_{MAC} using the key derivation mechanism described in Section 9.7.40 with (K.IC xor K.IFD) as key seed. All subsequent communication MUST be protected by Secure Messaging as described in Section 9.8.

4.3.3 Cryptographic Specifications

4.3.3.1 Encryption of Challenge and Response

Two key 3DES in CBC mode with zero IV (i.e. 0x00 00 00 00 00 00 00 00) according to [ISO/IEC 11568-2] SHALL be used for computation of E_{IFD} and E_{IC}. Padding for the input data MUST NOT be used when performing the EXTERNAL AUTHENTICATE command.

4.3.3.2 Authentication of Challenge and Response

The cryptographic checksums M_{IFD} and M_{IC} SHALL be calculated using [ISO/IEC 9797-1] MAC algorithm 3 with block cipher DES, zero IV (8 bytes), and [ISO/IEC 9797-1] padding method 2. The MAC length MUST be 8 bytes.

4.3.4 Application Protocol Data Units

Basic Access Control is performed using the commands GET CHALLENGE and EXTERNAL AUTHENTICATE with mutual authenticate function. The commands SHALL be encoded as specified in [ISO/IEC 7816-4].

4.3.4.1 GET CHALLENGE

Command		
CLA		Context specific
INS	0x84	GET CHALLENGE
P1/P2	0x0000	--
Data		*Absent*
Response		
Data	Random Nonce	
Status Bytes	0x9000	*Normal processing* Random Nonce successfully generated and transmitted.
	Other	*Operating system dependent error* Nonce could not be returned.

4.3.4.2 EXTERNAL AUTHENTICATE

Command			
CLA		Context specific	
INS	0x82	EXTERNAL AUTHENTICATE	
P1/P2	0x0000	--	
Data		Command data E_{IFD} || M_{IFD}	REQUIRED
Response			
Data		Response data E_{IC} || M_{IC}	REQUIRED
Status Bytes	0x9000	*Normal processing* The protocol has been performed successfully.	
	Other	*Operating system dependent error* The protocol failed.	

4.4 Password Authenticated Connection Establishment

PACE is a password authenticated Diffie-Hellman key agreement protocol that provides secure communication and password-based authentication of the eMRTD chip and the inspection system (i.e. eMRTD chip and inspection system share the same password π).

PACE establishes Secure Messaging between an eMRTD chip and an inspection system based on weak (short) passwords. The security context is established in the Master File. The protocol enables the eMRTD chip to verify that the inspection system is authorized to access stored data and has the following features:

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

PACE uses keys K_π derived from passwords with a key derivation function $\mathbf{KDF_\pi}$ (cf. Section 9.7.3). For globally interoperable machine readable travel documents the following two passwords and corresponding keys are available:

- MRZ: The key K_π defined by $K_\pi = \mathbf{KDF_\pi}(MRZ)$ is REQUIRED. It is derived from the Machine Readable Zone (MRZ) similar to Basic Access Control, i.e. the key is derived from the Document Number, the Date of Birth and the Date of Expiry.
- CAN: The key K_π defined by $K_\pi = \mathbf{KDF_\pi}(CAN)$ is OPTIONAL. It is derived from the Card Access Number (CAN). The CAN is a number printed on the *front side* of the datapage and MUST be chosen randomly or pseudo-randomly (e.g. using a cryptographically strong PRF).

Note.— In contrast to the MRZ (Document Number, Date of Birth, Data of Expiry) the CAN has the advantage that it can easily be typed in manually.

PACE supports different Mappings as part of the protocol execution:

- *Generic Mapping* based on a Diffie-Hellman Key Agreement;
- *Integrated Mapping* based on a direct mapping of a field element to the cryptographic group;
- *Chip Authentication Mapping* extends the Generic Mapping and integrates Chip Authentication into the PACE protocol.

If the chip supports Chip Authentication Mapping, at least one of Generic Mapping or Integrated Mapping and Chip Authentication MUST also be supported by the chip. This implies that for inspection systems supporting PACE, only support for Generic Mapping and Integrated Mapping is REQUIRED. Support for Chip Authentication Mapping is OPTIONAL.

4.4.1 Protocol Specification

The inspection system reads the parameters for PACE supported by the eMRTD chip from the file EF.CardAccess (cf. Section 9.2.8) and selects the parameters to be used, followed by the protocol execution.

The following commands SHALL be used:

- READ BINARY as specified in Doc 9303-10;
- MSE:SET AT (MANAGE SECURITY ENVIRONMENT command with SET Authentication Template function) as specified in Section 4.4.4.1;
- The following steps SHALL be performed by the inspection system and the eMRTD chip using a chain of General Authenticate commands as specified in Section 4.4.4.2:

  1) The eMRTD 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 to the inspection system.

  2) The inspection system recovers the plaintext $s = \mathbf{D}(K_\pi, z)$ with the help of the shared password π.

  3) Both the eMRTD chip and the inspection system perform the following steps:

     a) They exchange additional data required for the mapping of the nonce:

        i) for the generic mapping, the eMRTD chip and the inspection system exchange ephemeral key public keys.

        ii) for the integrated mapping, the inspection system sends an additional nonce to the eMRTD chip.

     b) They compute the ephemeral domain parameters $D = \mathbf{Map}(D_{IC}, s, \ldots)$ as described in Section 4.4.3.3.

     c) They perform an anonymous Diffie-Hellman key agreement (cf. Section 9.6) based on the ephemeral domain parameters and generate the shared secret $K = \mathbf{KA}(SK_{DH,IC}, PK_{DH,PCD}, D) = \mathbf{KA}(SK_{DH,PCD}, PK_{DH,IC}, D)$.

d) During Diffie-Hellman key agreement, the IC and the inspection system SHOULD check that the two public keys $PK_{DH,IC}$ and $PK_{DH,PCD}$ differ.

e) They derive session keys $KS_{MAC} = \mathbf{KDF}_{\mathrm{MAC}}(K)$ and $KS_{Enc} = \mathbf{KDF}_{\mathrm{Enc}}(K)$ as described in Section 9.7.1.

f) They exchange and verify the authentication token $T_{PCD} = \mathbf{MAC}(KS_{MAC},PK_{DH,IC})$ and $T_{IC} = \mathbf{MAC}(KS_{MAC},PK_{DH,PCD})$ as described in Section 4.4.3.4.

4) Conditionally, the MRTD chip computes Chip Authentication Data CA_{IC}, encrypts them $A_{IC} = \mathbf{E}(KS_{Enc}, CA_{IC})$ and sends them to the terminal (cf. Section 4.4.3.5.1). The terminal decrypts A_{IC} and verifies the authenticity of the chip using the recovered Chip Authentication Data CA_{IC} (cf. Section 4.4.3.5.2).

A simplified version of the protocol is also shown in Figure 1.

IC (chip)		***PCD (Inspection system)***
Static domain parameters D_{IC} Choose random nonce s Compute $z = \mathbf{E}(K_\pi,s)$		
	— z →	
		Compute $s = \mathbf{D}(K_\pi,z)$
	← additional data for **Map** →	
$D = \mathbf{Map}(D_{IC},s,\ldots)$ Choose random ephemeral key pair $(SK_{DH,IC}, PK_{DH,IC},D)$		$D = \mathbf{Map}(D_{IC},s,\ldots)$ Choose random ephemeral key pair $(SK_{DH,PCD},PK_{DH,PCD},D)$
	$\leftarrow PK_{DH,IC}, PK_{DH,PCD} \rightarrow$	
Check $PK_{DH,IC} \neq PK_{DH,PCD}$ $K = \mathbf{KA}(SK_{DH,IC}, PK_{DH,PCD},D)$ Compute $T_{IC} =$ $\mathbf{MAC}(KS_{MAC},PK_{DH,PCD})$ [compute CA_{IC} and encrypt as $A_{IC} = \mathbf{E}(KS_{Enc}, CA_{IC})$.]		Check $PK_{DH,IC} \neq PK_{DH,PCD}$ $K = \mathbf{KA}(SK_{DH,PCD},PK_{DH,IC},D)$ Compute $T_{PCD} =$ $\mathbf{MAC}(KS_{MAC},PK_{DH,IC})$
	$\leftarrow T_{IC}, T_{PCD} \rightarrow$ [— A_{IC} →]	
Verify T_{PCD}		Verify T_{IC} [decrypt A_{IC} and authenticate chip]

Figure 1. Password Authenticated Connection Establishment

4.4.2 Security Status

An eMRTD chip that supports PACE SHALL respond to unauthenticated read attempts (including selection of (protected) files in the LDS) with “Security status not satisfied” (0x6982).

Note.— This specification is more restrictive than the corresponding specification for BAC-only eMRTDs.

If PACE was successfully performed then the eMRTD chip has verified the used password. Secure Messaging is started using the derived session keys KS_{MAC} and KS_{Enc}.

4.4.3 Cryptographic Specifications

This section contains the cryptographic details of the specification.

Particular algorithms are selected by the issuer of the eMRTD. The inspection system MUST support all combinations described in the following subsections. The eMRTD chip MAY support more than one combination of algorithms.

Note.— Some algorithms are not available for the Chip Authentication Mapping: For security reasons, the use of 3DES is no longer recommended. DH-variants are not available to reduce the number of variants to be implemented by Terminals.

4.4.3.1 DH

For PACE with DH the respective algorithms and formats from Section 9.6 and Table 2 MUST be used.

Table 2. Algorithms and Formats for DH

OID	***Mapping***	***Sym. Cipher***	***Key-length***	***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

4.4.3.2 ECDH

For PACE with ECDH the respective algorithms and formats from Section 9.6 and Table 3 MUST be used.

Only prime curves with uncompressed points SHALL be used. The standardized domain parameters described in Section 09.5.1 SHOULD be used.

Table 3. Algorithms and Formats for ECDH

OID	***Mapping***	***Sym. Cipher***	***Key-length***	***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
id-PACE-ECDH-CAM-AES-CBC-CMAC-128	Chip Authentication	AES	128	CBC / CMAC	CMAC
id-PACE-ECDH-CAM-AES-CBC-CMAC-192		AES	192	CBC / CMAC	CMAC
id-PACE-ECDH-CAM-AES-CBC-CMAC-256		AES	256	CBC / CMAC	CMAC

4.4.3.3 Encrypting and Mapping Nonces

The eMRTD chip SHALL randomly and uniformly select the nonce *s* as a binary bit string of length *l*, where *l* is a multiple of the block size in bits of the respective block cipher **E**() chosen by the eMRTD chip.

- The nonce s SHALL be encrypted in CBC mode according to [ISO/IEC 10116] using the key $K_\pi = KDF_\pi(\pi)$ derived from the password π and IV set to the all-0 string.
- The nonce s SHALL be converted to a random generator using an algorithm-specific mapping function **Map**.
- For the Integrated Mapping the additional nonce *t* SHALL be selected randomly and uniformly as a binary bit string of length *k* and sent in clear. In this case *k* is the key size in bits of the respective block cipher **E**() and *l* SHALL be the smallest multiple of the block size of **E**() such that $l \geq k$.

To map the nonce *s* or the nonces *s*, *t* into the cryptographic group one of the following mappings SHALL be used:

- *Generic Mapping* (Section 4.4.3.3.1);
- *Integrated Mapping* (Section 4.4.3.3.2);
- *Chip Authentication Mapping* (Section 4.4.3.3.3).

4.4.3.3.1 Generic Mapping

ECDH

The function **Map**:G → Ĝ is defined as Ĝ = s×G+H, where H in <G> is chosen such that $\log_G H$ is unknown. The point H SHALL be calculated by an anonymous Diffie-Hellman Key Agreement [TR-03111] as H = **KA**$(SK_{Map,IC}, PK_{Map,PCD}, D_{IC})$ = **KA**$(SK_{Map,PCD}, PK_{Map,IC}, D_{IC})$.

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

DH

The function **Map**:g → ĝ is defined as $\hat{g} = g^s \times h$, where h in <g> is chosen such that $\log_g h$ is unknown. The group element h SHALL be calculated by an anonymous Diffie-Hellman Key Agreement as h = **KA**$(SK_{Map,IC}, PK_{Map,PCD}, D_{IC})$ = **KA**$(SK_{Map,PCD}, PK_{Map,IC}, D_{IC})$.

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

4.4.3.3.2 Integrated Mapping

ECDH

The function **Map**:G → Ĝ is defined as $\hat{G} = f_G(\mathbf{R}_p(s,t))$, where $\mathbf{R}_p()$ is a pseudo-random function that maps octet strings to elements of GF(p) and $f_G()$ is a function that maps elements of GF(p) to <G>. The random nonce t SHALL be chosen randomly by the inspection system and sent to the eMRTD chip. The pseudo-random function $\mathbf{R}_p()$is described below. The function $f_G()$ is defined in [BCIMRT2010]. An informative description is given in Appendix B.

DH

The function **Map**:g → ĝ is defined as $\hat{g} = f_g(\mathbf{R}_p(s,t))$, where $\mathbf{R}_p()$ is a pseudo-random function that maps octet strings to elements of GF(p) and $f_g()$ is a function that maps elements of GF(p) to <g>. The random nonce t SHALL be chosen randomly by the inspection system and sent to the eMRTD chip. The pseudo-random function $\mathbf{R}_p()$ is described below. The function $f_g()$ is defined as $f_g(x)=x^a \bmod p$, and a = (p-1)/q is the cofactor. Implementations MUST check that $\hat{g} \neq 1$.

Pseudo-random Number Mapping

The function $\mathbf{R}_p(s,t)$ is a function that maps octet strings s (of bit length l) and t (of bit length k) to an element $\mathrm{int}(x_1||x_2||...||x_n) \bmod p$ of GF(p). The function $\mathbf{R}_p(s,t)$ is specified below in Figure 2.

The construction is based on the respective block cipher **E**() in CBC mode according to [ISO/IEC 10116] with IV=0, where k is the key size (in bits) of **E**(). Where required, the output k_i MUST be truncated to key size k. The value n SHALL be selected as smallest number, such that $n*l \geq \log_2 p + 64$.

Note.— The truncation is only necessary for AES-192: Use octets 1 to 24 of k_i; additional octets are not used. In case of DES, k is considered to be equal to 128 bits, and the output of R(s,t) shall be 128 bits.

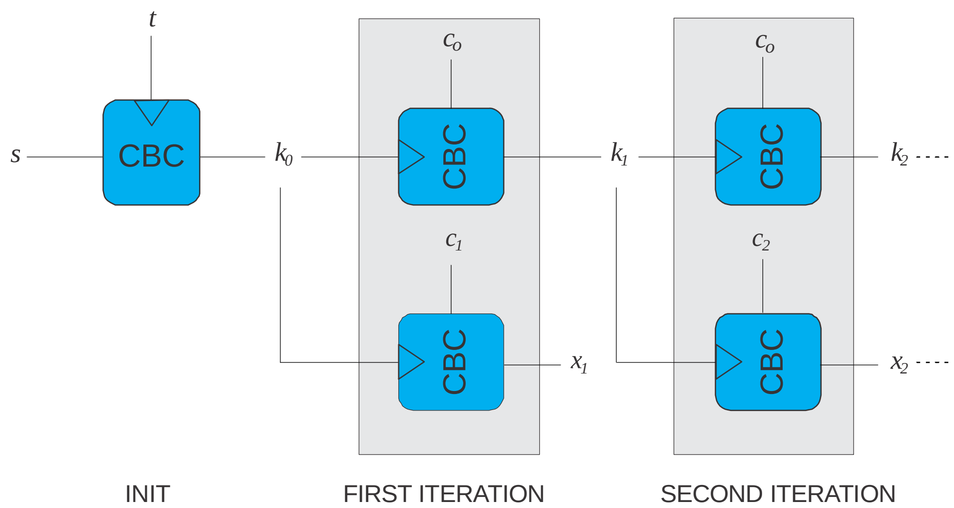

Figure 2. Pseudo-random number mapping

The constants c_0 and c_1 are defined as follows:

- For 3DES and AES-128 (l=128):
    - `c0=0xa668892a7c41e3ca739f40b057d85904`
    - `c1=0xa4e136ac725f738b01c1f60217c188ad`
- For AES-192 and AES-256 (l=192):
    - `c0=0xd463d65234124ef7897054986dca0a174e28df758cbaa03f240616414d5a1676`
    - `c1=0x54bd7255f0aaf831bec3423fcf39d69b6cbf066677d0faae5aadd99df8e53517`

4.4.3.3.3 Chip Authentication Mapping

The mapping phase of the PACE-CAM is identical to the mapping phase of PACE-GM (cf. Section 4.4.3.3.1).

4.4.3.4 Authentication Token

The authentication token SHALL be computed over a public key data object (cf. Section 9.4) containing the object identifier as indicated in MSE:Set AT (cf. Section 4.4.4.1), and the received ephemeral public key (i.e. excluding the domain parameters, cf. Section 9.4.4) using an authentication code and the key KS_{MAC} derived from the key agreement.

Note.— Padding is performed internally by the message authentication code, i.e. no application specific padding is performed.

3DES

3DES [FIPS 46-3] SHALL be used in Retail-mode according to [ISO/IEC 9797-1] MAC algorithm 3 / padding method 2 with block cipher DES and IV=0.

AES

AES [FIPS 197] SHALL be used in CMAC-mode [SP 800-38B] with a MAC length of 8 bytes.

4.4.3.5 Encrypted Chip Authentication Data

The MRTD chip MUST provide static key pair(s) SK_{IC}, PK_{IC} as described in Section 6.2. Encrypted Chip Authentication Data is REQUIRED for PACE with Chip Authentication Mapping.

4.4.3.5.1 Generation by the MRTD chip

The Chip Authentication Data SHALL be computed as $CA_{IC} = (SK_{IC})^{-1} * SK_{Map,IC} \bmod p$, where SK_{IC} is the static private key of the chip, $SK_{Map,IC}$ is the ephemeral private key used by the chip to computer H in the mapping phase of PACE (cf. Section 4.4.3.3.1) and p is the order of the used cryptographic group. The Chip Authentication Data SHALL be encrypted using the key KS_{Enc} derived from the key agreement as A_{IC} = $\mathbf{E}(KS_{Enc}, CA_{IC})$ to yield the Encrypted Chip Authentication Data.

Note.— $(SK_{IC})^{-1}$*can be precomputed during personalization of the MRTD chip and securely stored in the chip, avoiding the modular inversion during run-time.*

4.4.3.5.2 Verification by the terminal

The terminal SHALL decrypt A_{IC} to recover CA_{IC} and verify $PK_{Map,IC} = \mathbf{KA}(CA_{IC}, PK_{IC}, D_{IC})$, where PK_{IC} is the static public key of the MRTD chip.

Note.— Passive Authentication MUST be performed in combination with the Chip Authentication Mapping. Only after a successful validation of the respective Security Object may the MRTD chip be considered genuine.

4.4.3.5.3 Padding

The data to be encrypted SHALL be padded according to [ISO/IEC 9797-1] “Padding Method 2”.

4.4.3.5.4 AES

AES [19] SHALL be used in CBC-mode according to [ISO/IEC 10116] with $IV=E(KS_{Enc},-1)$, where -1 is the bit string of length 128 with all bits set to 1.

4.4.4 Application Protocol Data Units

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

1. MSE:Set AT
2. General Authenticate

4.4.4.1 MSE:Set AT

The command MSE:Set AT is used to select and initialize the PACE protocol.

Command			
CLA		Context specific	
INS	0x22	Manage Security Environment	
P1/P2	0xC1A4	Set Authentication Template for mutual authentication	
Data	0x80	*Cryptographic mechanism reference* Object Identifier of the protocol to select (value only, Tag 0x06 is omitted).	REQUIRED
	0x83	*Reference of a public key / secret key* The password to be used is indicated as follows: 0x01: MRZ_information 0x02: CAN	REQUIRED
	0x84	*Reference of a private key / Reference for computing a session key* This data object is REQUIRED 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.	CONDITIONAL
Response			
Data	–	Absent	
Status Bytes	0x9000	*Normal processing* 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 or domain parameter) is not available.	
	other	*Operating system dependent error* The initialization of the protocol failed.	

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

4.4.4.2 General Authenticate

A chain of General Authenticate commands is used to perform the PACE protocol.

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 as described in Section 4.4.5.	REQUIRED
Status Bytes	0x9000	*Normal processing* The protocol (step) was successful.	
	0x6300	*Authentication failed* The protocol (step) failed.	
	0x6A80	*Incorrect parameters in command data field* Provided data is invalid.	
	other	*Operating system dependent error* The protocol (step) failed.	

4.4.4.3 Command Chaining

Command chaining MUST be used for the General Authenticate command to link the sequence of commands to the execution of the protocol. Command chaining MUST NOT be used for other purposes unless clearly indicated by the chip. For details on command chaining see [ISO/IEC 7816-4].

4.4.5 Exchanged Data

The protocol specific data objects SHALL be exchanged in a chain of General Authenticate commands, with protocol specific command and response data encapsulated in a Dynamic Authentication data object (see Section 04.4.4.2) with context specific tags as shown in Table 4:

Table 4. Exchanged data for PACE

Step	***Description***	***Protocol Command Data***		***Protocol Response Data***	
1.	Encrypted Nonce	-	Absent[1]	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
				0x8A	Encrypted Chip Authentication Data (CONDITIONAL)

Encrypted Chip Authentication Data (cf. Section 4.4.3.5) MUST be present if Chip Authentication Mapping is used and MUST NOT be present otherwise.

4.4.5.1 Encrypted Nonce

The encrypted nonce (cf. Section 4.4.3.3) SHALL be encoded as octet string.

4.4.5.2 Mapping Data

The exchanged data is specific to the used mapping:

4.4.5.2.1 Generic Mapping

The ephemeral public keys (cf. Section 4.4.3.3 and Section 9.4.4) SHALL be encoded as elliptic curve point (ECDH) or unsigned integer (DH).

4.4.5.2.2 Integrated Mapping

The nonce *t* SHALL be encoded as octet string.

Note.— The context specific data object 0x82 SHALL be empty for the Integrated Mapping.

4.4.5.2.3 Chip Authentication Mapping

The encoding of the mapping data is identical to the Generic Mapping (cf. Section 4.4.5.2.1).

1. This implies an empty Dynamic Authentication Data Object

4.4.5.3 Public Keys

The public keys SHALL be encoded as described in Section 9.4.4.

4.4.5.4 Authentication Token

The authentication token (cf. Section 4.4.3.4) SHALL be encoded as octet string.

4.4.5.5 Encrypted Chip Authentication Data

The Chip Authentication Data SHALL be encoded as octet string using the function FE2OS() specified in [TR-03111] before encryption. Note that FE2OS() requires the encoding with the same number of octets as the prime order of the group, i.e. possibly including leading 0x00's. The Encrypted Chip Authentication Data SHALL be encoded as octet string.

5. AUTHENTICATION OF DATA

In addition to the LDS Data Groups, the contactless IC also contains a Document Security Object (SO_D). This object is digitally signed by the issuing State or organization and contains hash representations of the LDS contents (see Doc 9303-10).

An inspection system, containing the Document Signer Public Key (KPu_{DS}) of each State, or having read the Document Signer Certificate (C_{DS}) from the eMRTD, will be able to verify the Document Security Object (SO_D). In this way, through the contents of the Document Security Object (SO_D), the contents of the LDS are authenticated.

This verification mechanism does not require processing capabilities of the contactless IC in the eMRTD. Therefore it is called “Passive Authentication” of the contactless IC’s contents.

Passive Authentication proves that the contents of the Document Security Object (SO_D) and LDS are authentic and not changed. It does not prevent exact copying of the contactless IC’s content or chip substitution.

Therefore Passive Authentication SHOULD be supported by an additional physical inspection of the eMRTD.

5.1 Passive Authentication

5.1.1 Inspection Process

The inspection system performs the following steps:

1. The inspection system SHALL read the Document Security Object (SO_D) (which MUST contain the Document Signer Certificate (C_{DS}), see also Doc 9303-10) from the contactless IC.

2. The inspection system SHALL build and validate a certification path from a Trust Anchor to the Document Signer Certificate used to sign the Document Security Object (SO_D) according to Doc 9303-12.

3. The inspection system SHALL use the verified Document Signer Public Key (KPu_{DS}) to verify the signature of the Document Security Object (SO_D).

4. The inspection system MAY read relevant Data Groups from the contactless IC.

5. The inspection system SHALL ensure that the contents of the Data Group are authentic and unchanged by hashing the contents and comparing the result with the corresponding hash value in the Document Security Object (SO_D).

The following additional checks are considered Best Practice:

1. The inspection system or the inspection officer SHOULD check the presence of a DocumentTypeExtension in the Document Signer Certificate.

   - If yes, the inspection system SHOULD check the consistency of the DocumentTypeExtension, the Document Type from Data Group 1 and the Document Type from the visual MRZ (see Docs 9303-12, 9303-10 and 9303-3, respectively).

   - If no, the inspection system SHOULD check that the KeyUsage of the Document Signer Certificate is set to digitalSignature and that the Document Signer Certificate contains no ExtendedKeyUsage-Extension (see Doc 9303-12).

2. The inspection system or the inspection officer SHOULD check the consistency of the country codes from:

   - the Subject-field and, if present, the SubjectAltName of the Document Signer Certificate;

   - the Subject-field and, if present, the SubjectAltName of the Trust Anchor (CSCA certificate);

   - the Data Group 1 as read from the contactless IC; and

   - the visual MRZ.

   Additionally, the inspection system or the inspection officer MAY compare the contents of Data Group 1 to the visual MRZ (see Docs 9303-12, 9303-10 and 9303-3, respectively).

3. The inspection system SHOULD verify that the Issuing Date of the eMRTD is included in the Private Key Usage Period contained in the Document Signer Certificate (see Doc 9303-12).

The biometric information can now be used to perform the biometrics verification with the person who offers the eMRTD.

6. AUTHENTICATION OF THE CONTACTLESS IC

An issuing State or organization MAY choose to protect its eMRTDs against chip substitution.

The following mechanisms to verify the authenticity of the chip are available.

1. *Active Authentication*, as defined in Section 6.1. Support of Active Authentication is indicated by the presence of DG15. If available, the terminal MAY read and verify DG15 and perform Active Authentication.

2. *Chip Authentication*, as defined in Section 6.2. Support of Chip Authentication is indicated by the presence of corresponding `SecurityInfos` in DG14. If available, the terminal MAY read and verify DG14 and perform Chip Authentication.

3. *PACE with Chip Authentication Mapping* (PACE-CAM) as defined in Section 4.40. Support is indicated by the presence of a corresponding `PACEInfo` structure in CardAccess. If PACE-CAM was performed successfully in the chip access procedure, the terminal MAY perform the following to authenticate the chip:

   - read and verify CardSecurity
   - use the Public Key from CardSecurity together with the Mapping Data and the Chip Authentication Data received as part of PACE-CAM to authenticate the chip (Section 4.4.3.5.2).

6.1 Active Authentication

Active Authentication authenticates the contactless IC by signing a challenge sent by the IFD (inspection system) with a private key known only to the IC.

For this purpose the contactless IC contains its own Active Authentication Key pair (KPr_{AA} and KPu_{AA}). A hash representation of Data Group 15 (Public Key (KPu_{AA}) info) is stored in the Document Security Object (SO_D) and therefore authenticated by the issuer's digital signature. The corresponding Private Key (KPr_{AA}) is stored in the contactless IC's secure memory.

By authenticating the visual MRZ (through the hashed MRZ in the Document Security Object (SO_D)) in combination with the challenge response, using the eMRTD's Active Authentication Key Pair (KPr_{AA} and KPu_{AA}), the inspection system verifies that the Document Security Object (SO_D) has been read from the genuine contactless IC, stored in the genuine eMRTD.

Active Authentication requires processing capabilities of the eMRTD's contactless IC.

6.1.1 Protocol Specification

Active Authentication is performed using the [ISO/IEC 7816-4] INTERNAL AUTHENTICATE command.

If Active Authentication is performed after Secure Messaging was established, all commands and responses MUST be transmitted as Secure Messaging APDUs according to Section 9.8.

In more detail, IFD (inspection system) and IC (eMRTD's contactless IC) perform the following steps:

1. The IFD generates a nonce RND.IFD and sends it to the IC using the INTERNAL AUTHENTICATE command.

2. The IC performs the following operations:

   a) generate the message M;

   b) calculate h(M);

   c) compute the signature σ and send the response to the IFD.

3. The IFD verifies the response on the sent INTERNAL AUTHENTICATE command and checks if the IC returned the correct value.

6.1.2 Cryptographic Specifications

6.1.2.1 Nonce

The input is a nonce (RND.IFD) that MUST be 8 bytes.

Note.— Nonces MUST NOT be reused, e.g. the nonce used for BAC/PACE MUST NOT be reused for Active Authentication.

6.1.2.2 RSA

The IC SHALL compute a signature, when an integer factorization based mechanism is used, according to [ISO/IEC 9796-2] Digital Signature scheme 1.

In the following, k denotes the length of key for signature generation and L_h the length of the output of the hash function H used during signature generation. The trailer field option 1 MUST be used (and t set to 1) if SHA-1 is used during signature generation, trailer field option 2 MUST be used otherwise (and t set to 2).

The message M to be signed SHALL be the concatenation of M1 and M2, where M1 MUST be a nonce of length c - 4 bits (RND.IC) generated by the eMRTD, where c (the *capacity of the signature*) is given by $c = k - L_h - (8 \times t) - 4$, and M2 is RND.IFD generated by the Inspection System.

The result of the signature computation MUST be a signature σ without the non-recoverable message part M2.

eMRTDs SHOULD implement the signature generation scheme specified in [ISO/IEC 9796-2] paragraph B.6 and SHOULD NOT make use of the signature generation scheme specified in [ISO/IEC 9796-2] paragraph B.4. eMRTDs SHALL NOT implement other signature generation schemes.

Inspection systems SHALL implement the signature generation scheme specified in [ISO/IEC 9796-2] paragraph B.6 and SHOULD implement the signature generation scheme specified in [ISO/IEC 9796-2] paragraph B.4.

6.1.2.3 ECDSA

For ECDSA, the plain signature format according to [TR-03111] SHALL be used. Only prime curves with uncompressed points SHALL be used. A hash algorithm, whose output length is of the same length or shorter than the length of the ECDSA key in use, SHALL be used.

The message M to be signed is the nonce RND.IFD provided by the Inspection System.

6.1.3 Application Protocol Data Units

Active Authentication is performed by a single invocation of the INTERNAL AUTHENTICATE command as specified in [ISO/IEC 7816-4].

Command			
CLA		Context specific	
INS	0x88	INTERNAL AUTHENTICATE	
P1/P2	0x0000	--	
Data		*RND.IFD*	REQUIRED
Response			
Data		Signature σ generated by the IC	REQUIRED
Status Bytes	0x9000	*Normal processing* The protocol has been performed successfully.	
	Other	*Operating system dependent error* The protocol failed.	

6.1.4 Active Authentication Keys

The Active Authentication Key Pairs (KPr_{AA} and KPu_{AA}) SHALL be generated in a secure way.

Both the Active Authentication Public Key (KPu_{AA}) and the Active Authentication Private Key (KPr_{AA}) are stored in the eMRTD's contactless IC. After that, no Key Management is applicable for these keys.

Note.— It should be noted that when using key lengths exceeding 1 848 bits (if Secure Messaging with 3DES is used) / 1 792 bits (if Secure Messaging with AES is used) in Active Authentication with Secure Messaging, Extended Length APDUs MUST be supported by the eMRTD chip and the Inspection System.

Issuing States or organizations SHALL choose appropriate key lengths offering protection against attacks for the life time of the eMRTD. Suitable cryptographic catalogues SHOULD be taken into account.

6.1.5 Active Authentication Public Key Info

The Active Authentication Public Key is stored in the LDS Data Group 15. The format of the structure (`SubjectPublicKeyInfo`) is specified in [RFC 5280], see Section 9.1. All security objects MUST be produced in Distinguished Encoding Rule (DER) format to preserve the integrity of the signatures within them.

```
ActiveAuthenticationPublicKeyInfo ::= SubjectPublicKeyInfo
```

6.1.6 Inspection Process

When an eMRTD with Data Group 15 is offered to the inspection system, the Active Authentication mechanism MAY be performed to ensure that the data are read from the genuine contactless IC and that the contactless IC and data page belong to each other.

The inspection system and the contactless IC perform the following steps:

1. The entire MRZ is read visually from the eMRTD (if not already read as part of the Basic Access Control procedure) and compared with the MRZ value in Data Group 1. Since the authenticity and integrity of Data Group 1 have been checked through Passive Authentication, similarity ensures that the visual MRZ is authentic and unchanged.

2. Passive Authentication has also proven the authenticity and integrity of Data Group 15. This ensures that the Active Authentication Public Key (KPu_{AA}) is authentic and unchanged.

3. To ensure that the Document Security Object (SO_D) is not a copy, the inspection system uses the eMRTD's Active Authentication Key pair (KPr_{AA} and KPu_{AA}) in a challenge-response protocol with the eMRTD's contactless IC as described above.

After a successful challenge-response protocol, it is proven that the Document Security Object (SO_D) belongs to the data page, the contactless IC is genuine and contactless IC and data page belong to each other.

6.2 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.

The main differences to Active Authentication are:

- 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 C.

The static 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 (see Section 9.2.6).

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

As the eMRTD Application is selected as a result of the chip access procedure, Chip Authentication is performed in the eMRTD Application.

6.2.1 Protocol Specification

The following steps are performed by the terminal and the MRTD chip.

1. The MRTD chip sends its static Diffie-Hellman public key PK_{IC}, and the domain parameters D_{IC} to the terminal.

2. The terminal generates an ephemeral Diffie-Hellman key pair $(SK_{DH,PCD}, PK_{DH,PCD}, D_{IC})$ and sends the ephemeral public key $PK_{DH,PCD}$ to the MRTD chip.

3. Both the MRTD chip and the terminal compute the following:

   1. The shared secret K = $\mathbf{KA}(SK_{IC}, PK_{DH,PCD}, D_{IC}) = \mathbf{KA}(SK_{DH,PCD}, PK_{IC}, D_{IC})$

   2. The session keys $KS_{MAC} = \mathbf{KDF}_{MAC}(K)$ and $KS_{Enc} = \mathbf{KDF}_{Enc}(K)$ derived from K for Secure Messaging.

A simplified version is shown in Figure 3:

IC (chip)		***PCD (Inspection system)***
Static key pair $(SK_{IC}, PK_{IC}, D_{IC})$		
	— PK_{IC}, D_{IC} →	
		Choose random ephemeral key pair $(SK_{DH,PCD}, PK_{DH,PCD}, D_{IC})$
	← $PK_{DH,PCD}$ —	
$K = \mathbf{KA}(SK_{IC}, PK_{DH,PCD}, D_{IC})$		$K = \mathbf{KA}(SK_{DH,PCD}, PK_{IC}, D_{IC})$

Figure 3. Chip Authentication

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

6.2.2 Security Status

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

Note.— Passive Authentication MUST be performed in combination with Chip Authentication. Only after a successful validation of the respective Security Object may the MRTD chip be considered genuine.

6.2.3 Cryptographic Specifications

Particular algorithms are selected by the issuer of the MRTD. The inspection system MUST support all combinations described in the following subsections. The MRTD chip MAY support more than one combination of algorithms.

6.2.3.1 Chip Authentication with DH

For Chip Authentication with DH the respective algorithms and formats from Section 9.6 and Table 5 MUST be used. For Public Keys, PKCS#3 [PKCS#3] MUST be used instead of X9.42 [X9.42].

Table 5. Object Identifiers for Chip Authentication with DH

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

6.2.3.2 Chip Authentication with ECDH

For Chip Authentication with ECDH the respective algorithms and formats from Section 9.6 and Table 6 MUST be used.

Table 6. Object Identifiers for Chip Authentication with ECDH

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

6.2.4 Applications Protocol Data Units

Depending on the symmetric algorithm to be used, two implementations of Chip Authentication are available.

- The following command SHALL be used to implement Chip Authentication with 3DES Secure Messaging:

  1. MSE:Set KAT

- The following sequence of commands SHALL be used to implement Chip Authentication with AES Secure Messaging and MAY be used to implement Chip Authentication with 3DES Secure Messaging:

  1. MSE:Set AT
  2. General Authenticate

6.2.4.1 Implementation using MSE:Set KAT

Note.— MSE:Set KAT 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.

Command			
CLA		Context specific	
INS	0x22	Manage Security Environment	
P1/P2	0x41A6	Set Key Agreement Template for computation.	
Data	0x91	*Ephemeral Public Key* Ephemeral public key $PK_{DH,PCD}$ (cf. Section 9.4.4) 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. Section 6.2 and 9.2.6).	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.	
	other	*Operating system dependent error* The previously established session keys remain valid.	

6.2.4.2 Implementation using MSE:Set AT and General Authenticate

1. MSE:Set AT: The command MSE:Set AT is used to select and initialize the protocol.

Command			
CLA		Context specific	
INS	0x22	Manage Security Environment	
P1/P2	0x41A4	*Chip Authentication*: Set Authentication Template for internal authentication.	
Data	0x80	*Cryptographic mechanism reference* Object Identifier of the protocol to select (value only, Tag 0x06 is omitted).	REQUIRED

	0x84	*Reference of a private key* This data object is REQUIRED 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.	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. private key) is not available.	
	other	*Operating system dependent error* The initialization of the protocol failed.	

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

2. General Authenticate: The command General Authenticate is used to perform the Chip Authentication.

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
		0x80	Ephemeral Public Key	
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.		

	0x6A80	*Incorrect parameters in data field* Provided data is invalid.
	0x6A88	*Referenced data not found* The referenced data (i.e. private key) is not available.
	other	*Operating system dependent error* The protocol (step) failed.

Note.— The public keys for Chip Authentication supported by the chip are made available in the Security Object (see Section 9.2.8). If more than one public key is supported, the terminal MUST select the corresponding private key of the chip to be used within MSE:Set AT.

6.2.4.3 Ephemeral Public Key

The ephemeral public keys (cf. Section 9.4.4) SHALL be encoded as elliptic curve point (ECDH) or unsigned integer (DH).

7. ADDITIONAL ACCESS CONTROL MECHANISMS

The personal data stored in the contactless IC as defined to be the mandatory minimum for global interoperability are the MRZ and the digitally stored image of the bearer's face. Both items can also be seen (read) visually after the eMRTD has been opened and offered for inspection.

Besides the digitally stored image of the face as the primary biometric for global interoperability, ICAO has endorsed the use of digitally stored images of fingers and/or irises in addition to the face. For national or bilateral use, States MAY choose to store templates and/or MAY choose to limit access or encrypt this data, as to be decided by States themselves.

Access to this more sensitive personal data SHOULD be more restricted. This can be accomplished in two ways: extended access control or data encryption. Although these options are mentioned in this section, ICAO is not proposing or specifying any specifications or practices in these areas at this time.

7.1 Extended Access Control for Additional Biometrics

The Extended Access Control mechanism is OPTIONAL. For Extended Access Control a Document Extended Access Key set is used instead of the Document Basic Access Keys (K_{Enc} and K_{MAC}).

Defining the (IC-individual) Document Extended Access Key set is up to the implementing State. The Document Extended Access Key set MAY consist of either symmetric keys, e.g. derived from the MRZ and a National Master key, or an asymmetric key pair with a corresponding card verifiable certificate.

Extended Access Control requires processing capabilities of the eMRTD's contactless IC.

The implementation of the protection of the additional biometrics depends on the State's internal specifications or the bilaterally agreed specifications between States sharing this information.

7.2 Encryption of Additional Biometrics

Restricting access to the additional biometrics MAY also be done by encrypting them. To be able to decrypt the encrypted data, the inspection system MUST be provided with a decryption key. Defining the encryption/decryption algorithm and the keys to be used is up to the implementing State and is outside the scope of this document.

The implementation of the protection of the additional biometrics depends on the State's internal specifications or the bilaterally agreed specifications between States sharing this information.

8. INSPECTION SYSTEM

In order to support the required functionality and the defined options that can be implemented on eMRTDs that will be offered, the inspection system will have to meet certain pre-conditions.

8.1 Basic Access Control

Inspection systems supporting Basic Access Control MUST meet the following pre-conditions:

1. The inspection system is equipped with means to acquire the MRZ from the data page to derive the Document Basic Access Keys (K_{Enc} and K_{MAC}) from the eMRTD.

2. The inspection system's software supports the protocol described in Section 4.3, in the case that an eMRTD with Basic Access Control is offered to the system, including the encryption of the communication channel with Secure Messaging.

8.2 Password Authenticated Connection Establishment

Inspection systems supporting PACE MUST meet the following pre-conditions:

1. The inspection system is equipped with means to acquire the MRZ and/or the CAN from the data page.

2. The inspection system's software supports the protocol described in Section 4.4, in the case that an eMRTD with PACE is offered to the system, including the encryption of the communication channel with Secure Messaging.

8.3 Passive Authentication

To be able to perform a passive authentication of the data stored in the eMRTDs contactless IC, the inspection system needs to have knowledge of key information of the issuing States or organizations:

1. For each issuing State or organization, the Country Signing CA Certificate (C_{CSCA}) or the relevant information extracted from the certificate SHALL be securely stored in the inspection system.

2. Alternatively, for each issuing State or organization, the Document Signer Certificates (C_{DS}) or the relevant information extracted from the certificates SHALL be securely stored in the inspection system.

Before using a Country Signing CA Public Key of an issuing State or organization, trust in this key MUST be established by the receiving State or organization.

Before using a Document Signer Certificate (C_{DS}) for verification of a SO_D, the inspection system SHALL verify its digital signature, using the Country Signing CA Public Key (KPu_{CSCA}).

Additionally, inspection systems SHALL have access to verified revocation information.

8.4 Active Authentication

Support of Active Authentication by inspection systems is OPTIONAL.

If the inspection system supports Active Authentication, it is REQUIRED that the inspection system have the ability to read the visual MRZ.

If the inspection system supports Active Authentication, the inspection system's software SHALL support the Active Authentication protocol described in Section 6.1.

8.5 Chip Authentication

Support of Chip Authentication by inspection systems is OPTIONAL.

If the inspection system supports Chip Authentication, it is REQUIRED that the inspection system have the ability to read the visual MRZ.

If the inspection system supports Chip Authentication, the inspection system's software SHALL support the Chip Authentication protocol described in Section 6.2.

8.6 Extended Access Control to Additional Biometrics

The implementation of the protection of the optional additional biometrics depends on the State's internal specifications or the bilaterally agreed specifications between States sharing this information.

8.7 Decryption of Additional Biometrics

The implementation of the protection of the optional additional biometrics depends on the State's internal specifications or the bilaterally agreed specifications between States sharing this information.

9. COMMON SPECIFICATIONS

9.1 ASN.1 Structures

The data structures `SubjectPublicKeyInfo` and `AlgorithmIdentifier` are defined as follows:

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

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

Details on the `parameters` can be found in [X9.42] and [TR-03111].

9.2 Information on Supported Protocols

The ASN.1 data structure `SecurityInfos` SHALL be provided by the eMRTD 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.

Security Infos for PACE

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

- At least one `PACEInfo` using a standardized domain parameter MUST be present.
- For each supported set of explicit domain parameters a `PACEDomainParameterInfo` MUST be present.

Security Infos for Active Authentication

If ECDSA based signature algorithm is used for Active Authentication by the eMRTD chip, the `SecurityInfos` MUST contain the following `SecurityInfo` entry:

- `ActiveAuthenticationInfo`

Security Infos for Chip Authentication

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

- At least one `ChipAuthenticationInfo` and the corresponding `ChipAuthenticationPublicKeyInfo` using explicit domain parameters MUST be present.

Security Infos for Other Protocols

`SecurityInfos` may contain additional entries indicating support for other protocols. The inspection system may discard any unknown entry.

9.2.1 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. Only version 2 is supported by this specification.

- The integer `parameterId` is used to indicate the domain parameter identifier. It MUST be used if the eMRTD chip uses standardized domain parameters (cf. Section 9.5.1), provides multiple explicit domain parameters for PACE or `protocol` is one of the `*-CAM-*` OIDs. In case of PACE with Chip Authentication Mapping, the `parameterID` also denotes the ID of the Chip Authentication key used, i.e. the chip MUST provide a `ChipAuthenticationPublicKeyInfo` with `keyID` equal to `parameterID` from this data structure.

```
PACEInfo ::= SEQUENCE {
    Protocol    OBJECT IDENTIFIER(
                id-PACE-DH-GM-3DES-CBC-CBC |
                id-PACE-DH-GM-AES-CBC-CMAC-128 |
                id-PACE-DH-GM-AES-CBC-CMAC-192 |
                id-PACE-DH-GM-AES-CBC-CMAC-256 |
                id-PACE-ECDH-GM-3DES-CBC-CBC |
                id-PACE-ECDH-GM-AES-CBC-CMAC-128 |
                id-PACE-ECDH-GM-AES-CBC-CMAC-192 |
                id-PACE-ECDH-GM-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
                id-PACE-ECDH-CAM-AES-CBC-CMAC-128 |
                id-PACE-ECDH-CAM-AES-CBC-CMAC-192 |
                id-PACE-ECDH-CAM-AES-CBC-CMAC-256),
    version     INTEGER, -- MUST be 2
    parameterId INTEGER OPTIONAL
}
```

9.2.2 PACEDomainParameterInfo

This data structure is REQUIRED if the eMRTD chip provides explicit domain parameters for PACE and MUST be omitted otherwise.

- 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 eMRTD chip provides multiple explicit domain parameters for PACE.

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

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

Domain parameters contained in `PACEDomainParameterInfo` are unprotected and may be insecure. Using insecure domain parameters for PACE will leak the used password. eMRTD chips MUST support at least one set of standardized domain parameters as specified in Section 9.5.1. Inspection systems MUST NOT use explicit domain parameters provided by the eMRTD chip unless those domain parameters are explicitly known to be secure by the inspection systems.

Ephemeral public keys MUST be exchanged as plain public key values. More information on the encoding can be found in Section 09.4.4.

9.2.3 PACE Object Identifier

The object identifiers used for PACE 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
}
```

The following Object Identifier SHALL be used:

```
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}

id-PACE-ECDH-CAM                  OBJECT IDENTIFIER ::= {id-PACE 6}
id-PACE-ECDH-CAM-AES-CBC-CMAC-128 OBJECT IDENTIFIER ::= {id-PACE-ECDH-CAM 2}
id-PACE-ECDH-CAM-AES-CBC-CMAC-192 OBJECT IDENTIFIER ::= {id-PACE-ECDH-CAM 3}
id-PACE-ECDH-CAM-AES-CBC-CMAC-256 OBJECT IDENTIFIER ::= {id-PACE-ECDH-CAM 4}
```

9.2.4 ActiveAuthenticationInfo

If ECDSA based signature algorithm is used for Active Authentication by the eMRTD chip, the `SecurityInfos` in LDS Data Group 14 of the eMRTD chip MUST contain following `SecurityInfo` entry:

```
ActiveAuthenticationInfo ::= SEQUENCE {
    protocol id-icao-mrtd-security-aaProtocolObject
    version INTEGER -- MUST be 1
    signatureAlgorithm OBJECT IDENTIFIER
}
```

```
id-icao-mrtd-security-aaProtocolObject OBJECT IDENTIFIER ::=
                                  { id-icao-mrtd-security 5 }
```

For `signatureAlgorithm`, the object identifiers defined in [TR-03111] SHALL be used.

Note.— The Object Identifier `id-icao-mrtd-security` is defined in Doc 9303-10.

9.2.5 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, only version 1 is supported by this specification.
- 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.

```
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
  keyId      INTEGER OPTIONAL
}
```

9.2.6 ChipAuthenticationPublicKeyInfo

This data structure provides a public key for Chip Authentication or PACE with Chip Authentication Mapping 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 or if this key is used for PACE with Chip Authentication Mapping.

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

Note.— 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` and `ChipAuthenticationPublicKeyInfo`.

9.2.7 Chip Authentication Object Identifier

The following Object Identifier SHALL be used:

```
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}

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}
```

9.2.8 Storage on the Chip

To indicate support for the protocols and supported parameters, the MRTD chip SHALL provide `SecurityInfos` in transparent elementary files (see Doc 9303-10):

- The file EF.CardAccess contained in the master file is REQUIRED if PACE is supported by the eMRTD chip and SHALL contain the relevant `SecurityInfos` that are required for PACE:
  - o `PACEInfo`
  - o `PACEDomainParameterInfo`

- The file *CardSecurity* contained in the master file is REQUIRED if PACE with Chip Authentication Mapping is supported by the MRTD chip and SHALL contain the following `SecurityInfos`:
    - o `ChipAuthenticationPublicKeyInfo` as required for PACE-CAM
    - o The `SecurityInfos` contained in CardAccess.
- The file *DG14* contained in the eMRTD Application is REQUIRED if Chip Authentication or PACE-GM/-IM is supported by the MRTD chip and SHALL contain the following `SecurityInfos`:
    - o `ChipAuthenticationInfo` as required for Chip Authentication
    - o `ChipAuthenticationPublicKeyInfo` as required for Chip Authentication
    - o The `SecurityInfos` contained in CardAccess.
- The full set of `SecurityInfos` (including `SecurityInfos` contained in EF.CardAccess not specified in Doc 9303) SHALL additionally be stored in DG14 of the eMRTD Application (see Doc 9303-10).

The files MAY contain additional `SecurityInfos` out of scope of this specification.

Note.— While the authenticity of `SecurityInfos` *stored in DG14 and CardSecurity is protected by Passive Authentication, the file EF.CardAccess is unprotected.*

9.3 APDUs

9.3.1 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 [ISO/IEC 7816-4].

9.3.1.1 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/INFO as specified in [ISO/IEC 7816-4].

9.3.1.2 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/INFO 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/INFO.

- MSE:Set KAT
- General Authenticate

9.3.2 Command Chaining

Command chaining MUST be used for the General Authenticate command to link the sequence of commands to the execution of the protocol. Command chaining MUST NOT be used for other purposes unless clearly indicated by the chip. For details on command chaining see [ISO/IEC 7816-4].

9.4 Public Key Data Objects

A public key data object is a constructed BER TLV structure containing an object identifier and several context specific data objects nested within the cardholder public key template 0x7F49.

- 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.

9.4.1 Data Object Encoding

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.

To encode elliptic curve points, uncompressed encoding according to [TR-03111] SHALL be used.

9.4.2 Diffie Hellman Public Keys

The data objects contained in a DH public key are shown in Table 7. The order of the data objects is fixed.

Table 7. Data objects for DH public keys

Data Object	**Notation**	**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

Note.— The encoding of key components as unsigned integer implies that each of them is encoded over the least number of bytes possible, i.e. without preceding bytes set to 0x00. In particular, DH public key may be encoded over a number of bytes smaller than the number of bytes of the prime.

9.4.3 Elliptic Curve Public Keys

The data objects contained in an EC public key are shown in Table 8. The order of the data objects is fixed, CONDITIONAL domain parameters MUST be either all present, except the cofactor, or all absent as follows:

Table 8. Data objects for ECDH public keys

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

9.4.4 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.

Note.— 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.

9.5 Domain Parameters

With the exception of domain parameters contained in `PACEInfo`, all domain parameters SHALL be provided as `AlgorithmIdentifier` (cf. Section 9.1).

Within `PACEInfo`, the ID of standardized domain parameters described in Table 9 SHALL be referenced directly. Explicit domain parameters provided by `PACEDomainParameterInfo` MUST NOT use those IDs reserved for standardized domain parameters.

9.5.1 Standardized Domain Parameters

The standardized domain parameters IDs described in the table below SHOULD be used. Explicit domain parameters MUST NOT use those IDs reserved for standardized domain parameters.

The following object identifier SHOULD be used to reference standardized domain parameters in an `AlgorithmIdentifier` (cf. Section 9.1):

```
standardizedDomainParameters OBJECT IDENTIFIER ::= {
  bsi-de algorithms(1) 2
}
```

Within an `AlgorithmIdentifier` this object identifier SHALL reference the ID of the standardized domain parameter as contained in Table 9 as `INTEGER`, contained as `parameters` in the `AlgorithmIdentifier`.

Table 9. Standardized domain parameters

ID	***Name***	***Size (bit)***	***Type***	***Reference***
0	1024-bit MODP Group with 160-bit Prime Order Subgroup	1024/160	GFP	[RFC 5114]
1	2048-bit MODP Group with 224-bit Prime Order Subgroup	2048/224	GFP	[RFC 5114]
2	2048-bit MODP Group with 256-bit Prime Order Subgroup	2048/256	GFP	[RFC 5114]
3-7	RFU			
8	NIST P-192 (secp192r1)	192	ECP	[RFC 5114], [FIPS 186-4]
9	BrainpoolP192r1	192	ECP	[RFC 5639]
10	NIST P-224 (secp224r1) *	224	ECP	[RFC 5114], [FIPS 186-4]
11	BrainpoolP224r1	224	ECP	[RFC 5639]
12	NIST P-256 (secp256r1)	256	ECP	[RFC 5114], [FIPS 186-4]
13	BrainpoolP256r1	256	ECP	[RFC 5639]
14	BrainpoolP320r1	320	ECP	[RFC 5639]
15	NIST P-384 (secp384r1)	384	ECP	[RFC 5114], [FIPS 186-4]
16	BrainpoolP384r1	384	ECP	[RFC 5639]
17	BrainpoolP512r1	512	ECP	[RFC 5639]
18	NIST P-521 (secp521r1)	521	ECP	[RFC 5114], [FIPS 186-4]
19-31	RFU			

* This curve cannot be used with the Integrated Mapping.

9.5.2 Explicit Domain Parameters

The object identifier `dhpublicnumber` or `ecPublicKey` for DH or ECDH, respectively, SHALL be used to reference explicit domain parameters in an `AlgorithmIdentifier` (cf. Section 9.1):

```
dhpublicnumber OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) ansi-x942(10046) number-type(2) 1
}

ecPublicKey OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) ansi-x962(10045) keyType(2) 1
}
```

In the case of elliptic curves, domain parameters MUST be described explicitly in the `ECParameters` structure, contained as `parameters` in the `AlgorithmIdentifier`, i.e. named curves and implicit domain parameters MUST NOT be used.

9.6 Key Agreement Algorithms

This specification supports Diffie-Hellman and Elliptic Curve Diffie-Hellman key agreement as summarized in the following table:

Table 10. Key agreement algorithms

Algorithm / Format	***DH***	***ECDH***
Key Agreement Algorithm	[PKCS#3]	ECKA [TR-03111]
X.509 Public Key Format	[X9.42]	[TR-03111]
TLV Public Key Format	TLV, cf. Section 9.4.2	TLV, cf. Section 9.4.3
Ephemeral Public Key Validation	[RFC 2631]	[TR-03111]

9.7 Key Derivation Mechanism

9.7.1 Key Derivation Function

The key derivation function **KDF**(K,c), is defined as follows:

Input: The following inputs are required:

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

Output: An octet string keydata.

Actions: The following actions are performed:

- keydata = **H**(K || c)
- Output octet string keydata

The key derivation function **KDF**(K,c) requires a suitable hash function denoted by **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 [TR-03111], the x-coordinate of the generated point SHALL be used.

9.7.1.1 3DES

To derive 128-bit (112-bit excluding parity bits) 3DES [FIPS 46-3] keys the hash function SHA-1 [FIPS 180-2] 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).

9.7.1.2 AES

To derive 128-bit AES [FIPS 197] keys the hash function SHA-1 [FIPS 180-2] 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 [FIPS 197] keys SHA-256 [FIPS 180-2] 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.

9.7.2 Document Basic Access Keys

The computation of two key 3DES keys from a key seed (K) is used in the establishment of the Document Basic Access Keys K_{Enc} = **KDF**(K,1) and K_{MAC} = **KDF**(K,2).

9.7.3 PACE

Let $\mathbf{KDF}_{\pi}(\pi)$ = **KDF**(f(π),3) be a key derivation function to derive encryption keys from a password π. The encoding of passwords, i.e. K= f(π) is specified in Table 11:

Table 11. Password encodings

Password	Encoding
MRZ	SHA-1(Serial Number || Date of Birth || Date of Expiry)
CAN	[ISO/IEC 8859-1] encoded character string

9.7.4 Secure Messaging Keys

Keys for encryption and authentication are derived with $\mathbf{KDF}_{Enc}(K) = \mathbf{KDF}(K,1)$ and $\mathbf{KDF}_{MAC}(K) = \mathbf{KDF}(K,2)$ respectively, from a shared secret K.

9.8 Secure Messaging

9.8.1 Session Initiation

A session is started when secure messaging is established. Within a session the secure messaging keys (i.e. established by Basic Access Control, PACE or Chip Authentication) may be changed.

Secure Messaging is based on either 3DES [FIPS 46-3] or AES [FIPS 197] in encrypt-then-authenticate mode, i.e. data is padded, encrypted and afterwards the formatted encrypted data is input to the authentication calculation. The session keys SHALL be derived using the key derivation function described in Section 9.7.1.

Note.— Padding is always performed by the secure messaging layer, therefore the underlying message authentication code need not perform any internal padding.

9.8.2 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 first 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, see Section 9.8.6.3 for 3DES/ Section 9.8.7.3 for AES.
- The new session keys and the new SSC are used to protect subsequent commands/responses.

9.8.3 Session Termination

The MRTD chip MUST abort Secure Messaging if and only if a Secure Messaging error occurs or a plain APDU is received.

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

Note.— The MRTD chip MAY implicitly select the Master File when a session is terminated.

9.8.4 Message Structure of SM APDUs

The SM Data Objects (see [ISO/IEC 7816-4]) MUST be used in the following order:

- Command APDU: [DO'85' or DO'87'] [DO'97'] DO'8E'.
- Response APDU: [DO'85' or DO'87'] [DO'99'] DO'8E'.

All SM Data Objects MUST be encoded in BER TLV as specified in [ISO/IEC 7816-4]. The command header MUST be included in the MAC calculation, therefore the class byte CLA = 0x0C MUST be used.

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 Lc.

Figure 4 shows the transformation of an unprotected command APDU to a protected command APDU in the case *Data* and *Le* are available. If no *Data* is available, leave building DO '87' out. If *Le* is not available, leave building DO '97' out. To avoid ambiguity it is RECOMMENDED not to use an empty value field of Le Data Object (see also Section 10.4 of [ISO/IEC 7816-4]).

Figure 5 shows the transformation of an unprotected response APDU to a protected response APDU in case *Data* is available. If no *Data* is available, leave building DO '87' out.

9.8.5 SM Errors

Abortion of the Secure Channel for the eMRTD application occurs when:

- the contactless IC is de-powered; or
- the contactless IC recognizes an SM error while interpreting a command. In this case the status bytes must be returned without SM.

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

Note.— There MAY be other circumstances in which the contactless IC aborts the session. It is not feasible to provide a complete list of such circumstances.

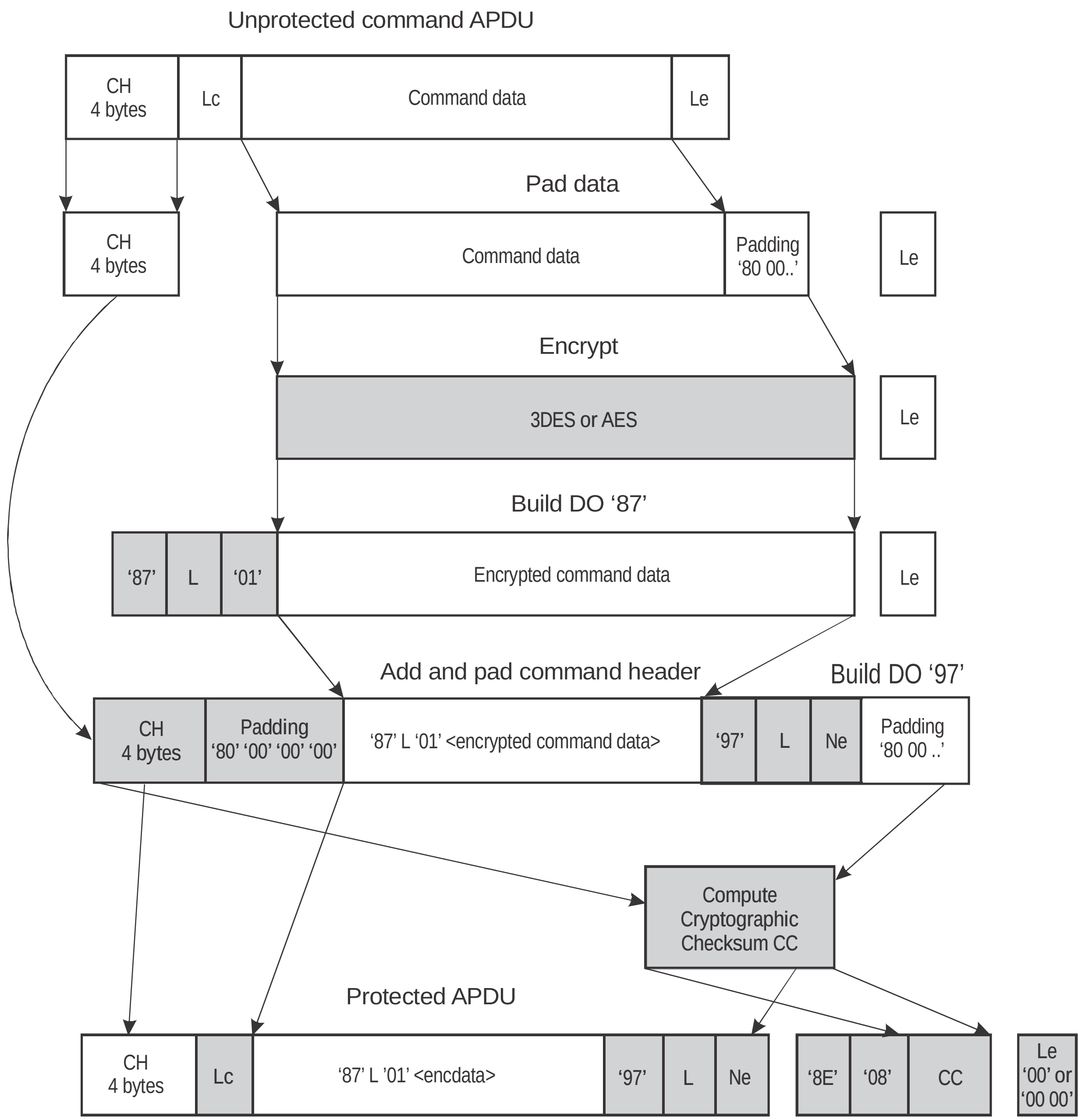

Figure 4. Computation of an SM command APDU for even INS Byte

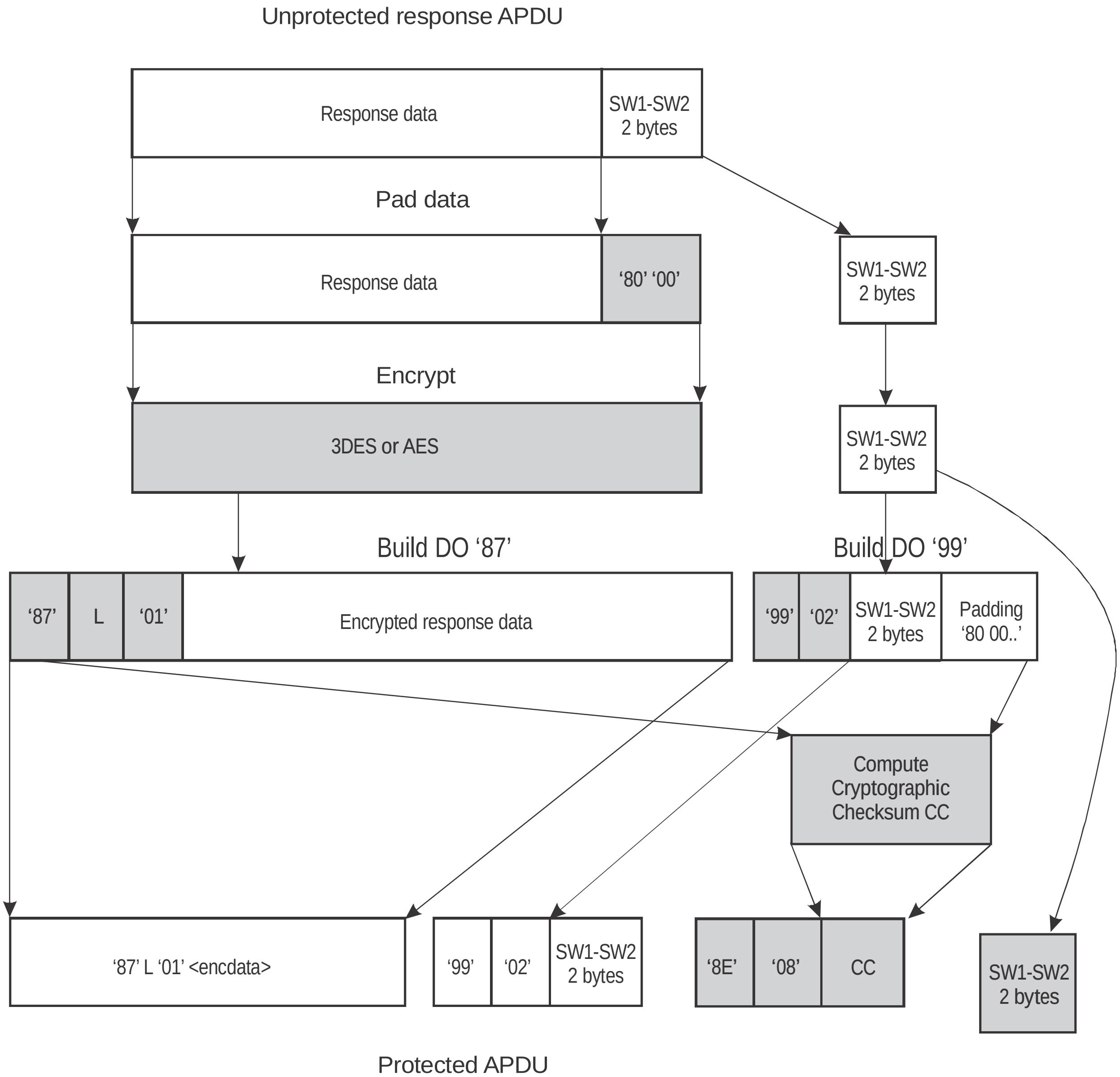

Figure 5. Computation of an SM response APDU for even INS Byte

9.8.6 3DES Modes of Operation

9.8.6.1 Encryption

Two key 3DES in CBC mode with zero IV (i.e. 0x00 00 00 00 00 00 00 00) according to [ISO/IEC 11568-2] is used. Padding according to [ISO/IEC 9797-1] padding method 2 is used.

9.8.6.2 Message Authentication

Cryptographic checksums are calculated using [ISO/IEC 9797-1] MAC algorithm 3 with block cipher DES, zero IV (8 bytes), and [ISO/IEC 9797-1] padding method 2. The MAC length MUST be 8 bytes.

After a successful authentication the datagram to be MACed MUST be prepended by the Send Sequence Counter.

9.8.6.3 Send Sequence Counter

For Secure Messaging following BAC, the Send Sequence Counter SHALL be initialized by concatenating the four least significant bytes of RND.IC and RND.IFD, respectively:

SSC = RND.IC (4 least significant bytes) || RND.IFD (4 least significant bytes).

In all other cases, the SSC SHALL be initialized to zero (i.e. 0x00 00 00 00 00 00 00 00).

9.8.7 AES Modes of Operation

9.8.7.1 Encryption

For message encryption AES [FIPS 197] SHALL be used in CBC-mode according to [ISO/IEC 10116] with key KS_{Enc} and IV = $\mathbf{E}(KS_{Enc}, SSC)$.

9.8.7.2 Message Authentication

For message authentication AES SHALL be used in CMAC-mode [SP 800-38B] with KS_{MAC} with a MAC length of 8 bytes. The datagram to be authenticated SHALL be prepended by the Send Sequence Counter.

9.8.7.3 Send Sequence Counter

The Send Sequence Counter SHALL be initialized to zero (i.e. 0x00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00).

10. REFERENCES (NORMATIVE)

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

[ISO/IEC 7816-4] ISO/IEC 7816-4:2013 Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange

[ISO/IEC 8859-1] ISO/IEC 8859-1:1998 Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1

[ISO/IEC 9796-2] ISO/IEC 9796-2:2010 Information technology — Security techniques — Digital signature schemes giving message recovery — Part 2: Integer factorization based mechanisms

[ISO/IEC 9797-1] ISO/IEC 9797-1:2011 Information technology — Security techniques — Message Authentication Codes (MACs) — Part 1: Mechanisms using a block cipher

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

[ISO/IEC 11568-2] ISO/IEC 11568-2:2012 Financial services — Key management (retail) — Part 2: Symmetric ciphers, their key management and life cycle

[ISO/IEC 11770-2] ISO/IEC 11770-2:2008 Information technology — Security techniques — Key management — Part 2: Mechanisms using symmetric techniques

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

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

[FIPS 186-4] NIST FIPS PUB 186-4, Digital Signature Standard (DSS), 2013

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

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

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

[RFC 5114] Lepinski, Matt; Kent, Stephen: RFC 5114 Additional Diffie-Hellman Groups for Use with IETF Standards, 2008

[RFC 5280] D. Cooper, S. Santesson, S. Farrell, S. Boyen, R. Housley, W. Polk, RFC 5280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2008

[RFC 5639] Lochter, Manfred; Merkle, Johannes: RFC 5639 Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation, 2010

[TR-03111] BSI: Technical Guideline TR-03111: Elliptic Curve Cryptography, Version 2.0, 2012

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

[Keesing2009] J. Bender, D. Kügler: Introducing the PACE solution, in: Keesing Journal of Documents & Identity, Issue 30, Keesing, 2009.

[BFK2009] J. Bender, M. Fischlin, D. Kügler: Security Analysis of the PACE Key-Agreement Protocol, in: Proceedings ISC 2009, LNCS volume 5735, Springer, 2009.

[BCIMRT2010] Brier, Eric; Coron, Jean-Sébastien; Icart, Thomas; Madore, David; Randriam, Hugues; and Tibouch, Mehdi, Efficient Indifferentiable Hashing into Ordinary Elliptic Curves, Advances in Cryptology – CRYPTO 2010, Springer-Verlag, 2010

— — — — — — — —

Appendix A to Part 11

ENTROPY OF MRZ-DERIVED ACCESS KEYS (INFORMATIVE)

Due to its simplicity Basic Access Control turned out to be a very successful protocol and it is implemented in almost every eMRP.

The security provided by Basic Access Control is limited by the design of the protocol. The Document Basic Access Keys (K_{Enc} and K_{MAC}) are generated from printed data with very limited randomness. The data that is used for the generation of the keys are Document Number, Date of Birth, and Date of Expiry. As a consequence the resulting keys have a relatively low entropy and are cryptographically weak. The actual entropy mainly depends on the type of the Document Number. For a 10-year valid travel document the maximum strength of the keys is approximately:

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

Especially in the second case this estimation requires the Document Number to be randomly and uniformly chosen which is usually not the case. 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.

There is no straightforward way to strengthen Basic Access Control as its limitations are inherent to the design of the protocol based on symmetric (“secret key”) cryptography. A cryptographically strong access control mechanism must (additionally) use asymmetric (“public key”) cryptography.

Password Authenticated Connection Establishment (PACE) was designed to overcome this problem. It employs asymmetric cryptography to establish session keys, whose strength is independent of the entropy of the used password. If PACE is implemented with elliptic curve cryptography with 256 Bit curves and AES-128 (a common choice), the session keys have 128 Bit entropy.

Two types of attacks must be distinguished:

- Skimming: this is an online attack, i.e. the attacker tries to access the contactless IC in real time, e.g. by guessing the password. If the protocol used to protect the contactless IC has no cryptographic weakness, the success probability of the attacker is given by the time the attacker has access to the IC, the duration of a single attempt to guess the password, and the entropy of the passport.
- Eavesdropping: this is an offline attack, i.e. the attacker tries to decrypt intercepted communication without access to the contactless IC. If the protocol used to establish the session keys has no cryptographic weakness, the success probability is given by the strength of the session keys and the computing power available to the attacker.

For further information see [Keesing2009] for a general discussion on entropy of session keys and a comparison of BAC and PACE, and [BFK2009] for a cryptographic analysis of PACE.

— — — — — — — —

Appendix B to Part 11

POINT ENCODING FOR THE ECDH-INTEGRATED MAPPING (INFORMATIVE)

B.1 HIGH-LEVEL DESCRIPTION OF THE POINT ENCODING METHOD

The algorithm takes as inputs the curve parameters (a, b, p, f) where (a, b) are the curve coefficients, p is the characteristic of the prime field over which the curve

$$E : y^2 \equiv x^3 + ax + b \ (mod\ p)$$

is defined. The order of E is always of the form fq for some prime q and f is called the co-factor. PACE v2 requires the generation of a point that belongs to the q-subgroup of E that we denote by $E[q]$. The point encoding also takes as input a number t such that

$$0 < t < p$$

and returns, in constant time, a point that belongs to $E[q]$. As described in [BCIMRT2010], point encoding comes in two flavours, depending on the coordinate system preferred by the implementation:

- A first implementation, described in Section B.2, outputs the elliptic curve point in affine coordinates (x, y);
- An alternate implementation, presented in Section B.3, outputs the same point in Jacobian coordinates (X, Y, Z).

Irrespective of the option taken, the generated point is identical in the sense that

$$x = XZ^{-2}\ mod\ p \text{ and } y = YZ^{-3}\ mod\ p$$

and the implementation of the subsequent phase of PACE v2 (the elliptic curve Diffie-Hellman key exchange phase) can therefore take advantage of using the option that best fits the interface of the cryptographic API that performs elliptic-curve operations.

As noted hereafter, point encoding for affine coordinates roughly requires two modular exponentiations modulo p whereas point encoding for Jacobian coordinates requires only a single one.

Note that for the two available implementations, point encoding explicitly requires that $p \equiv 3\ mod\ 4$.

B.2 IMPLEMENTATION FOR AFFINE COORDINATES

The algorithm is implemented as follows:

Inputs: curve parameters (a, b, p, f) and t such that $0 < t < p$
Output: a point (x, y) in the prime-order subgroup $E[q]$ of E

1. Compute $\alpha = -t^2 \bmod p$
2. Compute $X_2 = -ba^{-1}(1+(\alpha+\alpha^2)^{-1}) \bmod p$
3. Compute $X_3 = \alpha X_2 \bmod p$
4. Compute $h_2 = (X_2)^3 + a X_2 + b \bmod p$
5. Compute $h_3 = (X_3)^3 + a X_3 + b \bmod p$
6. Compute $U = t^3 h_2 \bmod p$
7. Compute $A = (h_2)^{p-1-(p+1)/4} \bmod p$
8. If $A^2 h_2 = 1 \bmod p$ define $(x, y) = (X_2, A\, h_2 \bmod p)$
9. Otherwise define $(x, y) = (X_3, A\, U \bmod p)$
10. Output $(x, y) = [f]\,(x, y)$.

Implementation Notes

Neglecting modular multiplications and additions, the execution time of the above implementation is dominated by two modular exponentiations:

- Step 2 can be rewritten

$$X_2 = -ba^{-1}(1+(\alpha+\alpha^2)^{-1}) = -b(1+\alpha+\alpha^2)\,(a(\alpha+\alpha^2))^{p-2} \bmod p$$

  which essentially amounts to a modular exponentiation with exponent $p-2$;

- Step 7 is a modular exponentiation with exponent $p-1-(p+1)/4$.

Note.— Step 10 requires a scalar multiplication by the co-factor f. *For many curves, the co-factor is equal to 1 so that this scalar multiplication can be avoided.*

B.3 IMPLEMENTATION FOR JACOBIAN COORDINATES

The algorithm is implemented as follows:

Inputs: curve parameters (a, b, p, f) and t such that $0 < t < p$

Output: a point (X, Y, Z) in the prime-order subgroup $E[q]$ of E

1. Compute $\alpha = -t^2 \bmod p$
2. Compute $Z = a(\alpha+\alpha^2) \bmod p$
3. Compute $X_2 = -bZ\,(1+\alpha+\alpha^2) \bmod p$
4. Compute $X_3 = \alpha X_2 \bmod p$
5. Compute $h_2 = (X_2)^3 + a X_2 Z^4 + b Z^6 \bmod p$
6. Compute $h_3 = (X_3)^3 + a X_3 Z^4 + b Z^6 \bmod p$
7. Compute $U = -\alpha\, t\, h_2 \bmod p$
8. Compute $A = (h_2)^{p-1-(p+1)/4} \bmod p$
9. If $A^2 h_2 = 1 \bmod p$ define $(X, Y, Z) = (X_2, A\, h_2 \bmod p, Z)$
10. Otherwise define $(X, Y, Z) = (X_3, A\, U \bmod p, Z)$
11. Output $(X, Y, Z) = [f](X, Y, Z)$.

Implementation Notes

Neglecting modular multiplications and additions, the execution time of the above implementation is dominated by the single modular exponentiation of Step 7. Therefore, it is expected to be roughly twice as fast as the implementation for affine coordinates.

Note.— The scalar multiplication in Step 10 can be completely avoided when the co-factor f *is equal to 1.*

— — — — — — — —

Appendix C to Part 11

CHALLENGE SEMANTICS (INFORMATIVE)

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

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

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

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

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, States may use Challenge Semantics for their internal use, e.g., to prove that a certain person indeed has immigrated. On the negative side such proofs 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 or hotels.

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 are 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 identity of the prover who signs the challenge.

— — — — — — — —

Appendix D to Part 11

WORKED EXAMPLE: BASIC ACCESS CONTROL (INFORMATIVE)

D.1 COMPUTE KEYS FROM KEY SEED (K_{SEED})

This Section provides an example for derivation of 3DES keys from a seed value K_{seed}. This procedure will be used as a "subroutine" in the examples for Basic Access Control.

Input:

K_{seed} = '239AB9CB282DAF66231DC5A4DF6BFBAE'

Compute encryption key (c = '00000001'):

1. Concatenate K_{seed} and c:
   D = '239AB9CB282DAF66231DC5A4DF6BFBAE00000001'

2. Calculate the SHA-1 hash of D:
   H_{SHA-1}(D) = 'AB94FCEDF2664EDFB9B291F85D7F77F27F2F4A9D'

3. Form DES keys K_a and K_b, intended to be used as first and second key for 3DES (i.e. the 3DES key is the concatenation of K_a and K_b):
   $K_{a'}$ = 'AB94FCEDF2664EDF'
   $K_{b'}$ = 'B9B291F85D7F77F2'

4. Adjust parity bits:
   K_a = 'AB94FDECF2674FDF'
   K_b = 'B9B391F85D7F76F2'

Compute MAC computation key (c = '00000002'):

1. Concatenate K_{seed} and c:
   D = '239AB9CB282DAF66231DC5A4DF6BFBAE00000002'

2. Calculate the SHA-1 hash of D:
   H_{SHA-1}(D) = '7862D9ECE03C1BCD4D77089DCF131442814EA70A'

3. Form keys $K_{a'}$ and $K_{b'}$:
   $K_{a'}$ = '7862D9ECE03C1BCD'
   $K_{b'}$ = '4D77089DCF131442'

4. Adjust parity bits:
   K_a = '7962D9ECE03D1ACD'
   K_b = '4C76089DCE131543'

D.2 DERIVATION OF DOCUMENT BASIC ACCESS KEYS (K_{ENC} AND K_{MAC})

This section provides examples how the Basic Access Keys are derived from the MRZ.

TD2 MRZ, document number exceeds 9 characters

1. Read the MRZ

   MRZ = `I<UTOSTEVENSON<<PETER<JOHN<<<<<<<<<<`
   `D23145890<UTO3407127M95071227349<<<8`

2. Construct the 'MRZ information' from the MRZ

Document number	= D23145890734	check digit = 9
Date of Birth	= 340712	check digit = 7
Date of Expiry	= 950712	check digit = 2
MRZ_information	= D23145890734934071279507122	

Continue with step 3.

TD2 MRZ, document number 9 characters

1. Read the MRZ:

   MRZ = `I<UTOERIKSSON<<ANNA<MARIA<<<<<<<<<<<`
   `L898902C<3UTO6908061F9406236<<<<<<<8`

2. Construct the 'MRZ_information' from the MRZ:

Document number	= L898902C<	check digit = 3
Date of birth	= 690806	check digit = 1
Date of expiry	= 940623	check digit = 6
MRZ_information	= L898902C<369080619406236	

Continue with step 3.

TD1 MRZ, document number exceeds 9 characters

1. Read the MRZ

   MRZ = `I<UTOD23145890<7349<<<<<<<<<<<`
   `3407127M9507122UTO<<<<<<<<<<<2`
   `STEVENSON<<PETER<JOHN<<<<<<<<<`

2. Construct the 'MRZ information' from the MRZ

Document number	= D23145890734	check digit = 9
Date of Birth	= 340712	check digit = 7
Date of Expiry	= 950712	check digit = 2
MRZ_information	= D23145890734934071279507122	

Continue with step 3.

TD1 MRZ, document number 9 characters

1. Read the MRZ

   MRZ = `I<UTOL898902C<3<<<<<<<<<<<<<<<`
   `6908061F9406236UTO<<<<<<<<<<<1`
   `ERIKSSON<<ANNA<MARIA<<<<<<<<<<`

2. Construct the 'MRZ information' from the MRZ
   Document number = L898902C< check digit = 3
   Date of Birth = 690806 check digit = 1
   Date of Expiry = 940623 check digit = 6
   MRZ_information = L898902C<369080619406236

3. Calculate the SHA-1 hash of 'MRZ_information':
   H_{SHA-1}(MRZ_information) = '239AB9CB282DAF66231DC5A4DF6BFBAEDF477565'

4. Take the most significant 16 bytes to form the K_{seed}:
   K_{seed} = '239AB9CB282DAF66231DC5A4DF6BFBAE'

5. Calculate the basic access keys (K_{Enc} and K_{MAC}) according to Section 9.7.1/Appendix D.1:
   K_{Enc} = 'AB94FDECF2674FDFB9B391F85D7F76F2'
   K_{MAC} = '7962D9ECE03D1ACD4C76089DCE131543'

D.3 AUTHENTICATION AND ESTABLISHMENT OF SESSION KEYS

This section provides an example for performing Basic Access Control.

Inspection system:

1. Request an 8 byte random number from the eMRTD's contactless IC:

Command APDU:				
CLA	**INS**	**P1**	**P2**	**Le**
00	84	00	00	08

Response APDU:	
Response data field	**SW1-SW2**
RND.IC	9000

RND.IC = '4608F91988702212'

2. Generate an 8 byte random and a 16 byte random:
   RND.IFD = '781723860C06C226'
   K_{IFD} = '0B795240CB7049B01C19B33E32804F0B'

3. Concatenate RND.IFD, RND.IC and K_{IFD}:
   S = '781723860C06C2264608F91988702212
   0B795240CB7049B01C19B33E32804F0B'

4. Encrypt S with 3DES key K_{Enc}:
   E_{IFD} = '72C29C2371CC9BDB65B779B8E8D37B29
   ECC154AA56A8799FAE2F498F76ED92F2'

5. Compute MAC over E_{IFD} with 3DES key K_{MAC}:
   M_{IFD} = `'5F1448EEA8AD90A7'`

6. Construct command data for EXTERNAL AUTHENTICATE and send command APDU to the eMRTD's contactless IC:
   cmd_data = `'72C29C2371CC9BDB65B779B8E8D37B29ECC154AA 56A8799FAE2F498F76ED92F25F1448EEA8AD90A7'`

Command APDU:						
CLA	**INS**	**P1**	**P2**	**Lc**	**Command data field**	**Le**
00	82	00	00	28	cmd_data	28

eMRTD's contactless IC:

1. Decrypt and verify received data and compare RND.IC with response on GET CHALLENGE.

2. Generate a 16 byte random:
   K_{IC} = `'0B4F80323EB3191CB04970CB4052790B'`

3. Calculate XOR of K_{IFD} and K_{IC}:
   K_{seed} = `'0036D272F5C350ACAC50C3F572D23600'`

4. Calculate session keys (KS_{Enc} and KS_{MAC}) according to Section 9.7.1/Appendix D.1:
   KS_{Enc} = `'979EC13B1CBFE9DCD01AB0FED307EAE5'`
   KS_{MAC} = `'F1CB1F1FB5ADF208806B89DC579DC1F8'`

5. Calculate send sequence counter:
   SSC = `'887022120C06C226'`

6. Concatenate RND.IC, RND.IFD and K_{IC}:
   R = `'4608F91988702212781723860C06C226 0B4F80323EB3191CB04970CB4052790B'`

7. Encrypt R with 3DES key K_{Enc}:
   E_{IC} = `'46B9342A41396CD7386BF5803104D7CE DC122B9132139BAF2EEDC94EE178534F'`

8. Compute MAC over E_{IC} with 3DES key K_{MAC}:
   M_{IC} = `'2F2D235D074D7449'`

9. Construct response data for EXTERNAL AUTHENTICATE and send response APDU to the inspection system:
   resp_data = `'46B9342A41396CD7386BF5803104D7CEDC122B91 32139BAF2EEDC94EE178534F2F2D235D074D7449'`

Response APDU:	
Response data field	**SW1-SW2**
resp_data	9000

Inspection system:

1. Decrypt and verify received data and compare received RND.IFD with generated RND.IFD.

2. Calculate XOR of K_{IFD} and K_{IC}:
   K_{seed} = ‘0036D272F5C350ACAC50C3F572D23600’

3. Calculate session keys (KS_{Enc} and KS_{MAC}) according to Section 9.7.1/Appendix D.1:
   KS_{Enc} = ‘979EC13B1CBFE9DCD01AB0FED307EAE5’
   KS_{MAC} = ‘F1CB1F1FB5ADF208806B89DC579DC1F8’

4. Calculate send sequence counter:
   SSC = ‘887022120C06C226’

D.4 SECURE MESSAGING

After authentication and establishment of the session keys, the inspection system selects the EF.COM (File ID = ‘011E’) and reads the data using secure messaging. The calculated KS_{Enc}, KS_{MAC} and SSC (previous steps 3 and 4 of the inspection system) will be used.

First the EF.COM will be selected, then the first four bytes of this file will be read so that the length of the structure in the file can be determined and after that the remaining bytes are read.

1. Select EF.COM

   Unprotected command APDU:

CLA	INS	P1	P2	Lc	Command data field
00	A4	02	0C	02	01 1E

   a) Mask class byte and pad command header:
      CmdHeader = ‘0CA4020C80000000’

   b) Pad data:
      Data = ‘011E800000000000’

   c) Encrypt data with KS_{Enc}:
      EncryptedData = ‘6375432908C044F6’

   d) Build DO‘87’:
      DO87 = ‘8709016375432908C044F6’

   e) Concatenate CmdHeader and DO‘87’:
      M = ‘0CA4020C800000008709016375432908C044F6’

f) Compute MAC of M:

   i) Increment SSC with 1:
      SSC = '887022120C06C227'

   ii) Concatenate SSC and M and add padding:
      N = '887022120C06C2270CA4020C80000000
      8709016375432908C044F68000000000'

   iii) Compute MAC over N with KS_{MAC}:
      CC = 'BF8B92D635FF24F8'

g) Build DO'8E':
   DO8E = '8E08BF8B92D635FF24F8'

h) Construct and send protected APDU:
   ProtectedAPDU = '0CA4020C158709016375432908C0
   44F68E08BF8B92D635FF24F800'

i) Receive response APDU of eMRTD's contactless IC:
   RAPDU = '990290008E08FA855A5D4C50A8ED9000'

j) Verify RAPDU CC by computing MAC of DO'99':

   i) Increment SSC with 1:
      SSC = '887022120C06C228'

   ii) Concatenate SSC and DO'99' and add padding:
      K = '887022120C06C2289902900080000000'

   iii) Compute MAC with KS_{MAC}:
      CC' = 'FA855A5D4C50A8ED'

   iv) Compare CC' with data of DO'8E' of RAPDU.
      'FA855A5D4C50A8ED' == 'FA855A5D4C50A8ED' ? YES.

2. Read Binary of first four bytes:

Unprotected command APDU:

CLA	INS	P1	P2	Le
00	B0	00	00	04

a) Mask class byte and pad command header:
   CmdHeader = '0CB0000080000000'

b) Build DO'97':
   DO97 = '970104'

c) Concatenate CmdHeader and DO'97':
   M = '0CB0000080000000970104'

d) Compute MAC of M:

i) Increment SSC with 1:
SSC = `‘887022120C06C229’`

ii) Concatenate SSC and M and add padding:
N = `‘887022120C06C2290CB00000`
`8000000097010480000000000’`

iii) Compute MAC over N with KSMAC:
CC = `‘ED6705417E96BA55’`

e) Build DO‘8E’:
DO8E = `‘8E08ED6705417E96BA55’`

f) Construct and send protected APDU:
ProtectedAPDU = `‘0CB000000D9701048E08ED6705417E96BA5500’`

g) Receive response APDU of eMRTD’s contactless IC:
RAPDU = `‘8709019FF0EC34F992265199029000`
`8E08AD55CC17140B2DED9000’`

h) Verify RAPDU CC by computing MAC of concatenation DO‘87’ and DO‘99’:

i) Increment SSC with 1:
SSC = `‘887022120C06C22A’`

ii) Concatenate SSC, DO‘87’ and DO‘99’ and add padding:
K = `‘887022120C06C22A8709019F`
`F0EC34F99226519902900080’`

iii) Compute MAC with KS_{MAC}:
CC’ = `‘AD55CC17140B2DED’`

iv) Compare CC’ with data of DO‘8E’ of RAPDU:
`‘AD55CC17140B2DED’ == ‘AD55CC17140B2DED’` ? YES.

i) Decrypt data of DO‘87’ with KS_{Enc}:
DecryptedData = `‘60145F01’`

j) Determine length of structure:
L = ‘14’ + 2 = 22 bytes

3. Read Binary of remaining 18 bytes from offset 4:

Unprotected command APDU:

CLA	INS	P1	P2	Le
00	B0	00	04	12

a) Mask class byte and pad command header:
   CmdHeader = `'0CB0000480000000'`

b) Build DO'97':
   DO97 = `'970112'`

c) Concatenate CmdHeader and DO'97':
   M = `'0CB0000480000000970112'`

d) Compute MAC of M:

   i) Increment SSC with 1:
      SSC = `'887022120C06C22B'`

   ii) Concatenate SSC and M and add padding:
      N = `'887022120C06C22B0CB00004`
      `800000009701128000000000'`

   iii) Compute MAC over N with KS_{MAC}:
      CC = `'2EA28A70F3C7B535'`

e) Build DO'8E':
   DO8E = `'8E082EA28A70F3C7B535'`

f) Construct and send protected APDU:
   ProtectedAPDU = `'0CB000040D9701128E082EA28A70F3C7B53500'`

g) Receive response APDU of eMRTD's contactless IC:
   RAPDU = `'871901FB9235F4E4037F2327DCC8964F1F9B8C30F42`
   `C8E2FFF224A990290008E08C8B2787EAEA07D749000'`

h) Verify RAPDU CC by computing MAC of concatenation DO'87' and DO'99':

   i) Increment SSC with 1:
      SSC = `'887022120C06C22C'`

   ii) Concatenate SSC, DO'87' and DO'99' and add padding:
      K = `'887022120C06C22C871901FB9235F4E4037F232`
      `7DCC8964F1F9B8C30F42C8E2FFF224A99029000'`

   iii) Compute MAC with KS_{MAC}:
      CC' = `'C8B2787EAEA07D74'`

   iv) Compare CC' with data of DO'8E' of RAPDU:
      `'C8B2787EAEA07D74' == 'C8B2787EAEA07D74'` ? YES.

i) Decrypt data of DO'87' with KS_{Enc}:
   DecryptedData = `'04303130365F36063034303030305C026175'`

RESULT:

EF.COM data = `'60145F0104303130365F36063034303030305C026175'`

— — — — — — — —

Appendix E to Part 11

WORKED EXAMPLE: PASSIVE AUTHENTICATION (INFORMATIVE)

Step 1. Read the Document Security Object (SO_D) (optionally containing the Document Signer Certificate (C_{DS})) from the contactless IC.

Step 2: Read the Document Signer (DS) from the Document Security Object (SO_D).

Step 3: The inspection system verifies SO_D by using Document Signer Public Key (KPu_{DS}).

Step 4: The inspection system verifies C_{DS} by using the Country Signing CA Public Key (KPu_{CSCA}).

If both verifications in step 3 and 4 are correct, then this ensures that the contents of SO_D can be trusted and can be used in the inspection process.

Step 5: Read the relevant Data Groups from the LDS.

Step 6: Calculate the hashes of the relevant Data Groups.

Step 7: Compare the calculated hashes with the corresponding hash values in the SO_D.

If the hash values in step 7 are identical, this ensures that the contents of the Data Group are authentic and unchanged.

— — — — — — — —

Appendix F to Part 11

WORKED EXAMPLE: ACTIVE AUTHENTICATION (INFORMATIVE)

This worked example uses the following settings:

1. Integer factorization-based mechanism: RSA
2. Modulus length (k): 1 024 bits (128 bytes)
3. Hash algorithm: SHA-1

Inspection system:

Step 1. Generate an 8 byte random:
RND.IFD = ‘F173589974BF40C6’

Step 2. Construct command for internal authenticate and send command APDU to the eMRTD’s contactless IC:

Command APDU

CLA	INS	P1	P2	Lc	Command data field	Le
00	88	00	00	08	RND.IFD	00

eMRTD’s contactless IC:

Step 3. Determine M_2 from incoming APDU:
M_2 = ‘F173589974BF40C6’

Step 4. Create the trailer:
T = ‘BC’ (i.e. SHA-1)
t (length of T in octets) = 1

Step 5. Determine lengths:
a. $c = k – L_h – 8t – 4 = 1024 – 160 – 8 – 4 = 852$ bits
b. $L_{M1} = c – 4 = 848$ bits

Step 6. Generate nonce M_1 of length L_{M1}:
M_1 = ‘9D2784A67F8E7C659973EA1AEA25D95B
6C8F91E5002F369F0FBDCE8A3CEC1991
B543F1696546C5524CF23A5303CD6C98
599F40B79F377B5F3A1406B3B4D8F967
84D23AA88DB7E1032A405E69325FA91A
6E86F5C71AEA978264C4A207446DAD4E
7292E2DCDA3024B47DA8’

Step 7. Create M:

M = M_1 | M_2 = ‘9D2784A67F8E7C659973EA1AEA25D95B
6C8F91E5002F369F0FBDCE8A3CEC1991
B543F1696546C5524CF23A5303CD6C98
599F40B79F377B5F3A1406B3B4D8F967
84D23AA88DB7E1032A405E69325FA91A
6E86F5C71AEA978264C4A207446DAD4E
7292E2DCDA3024B47DA8F173589974BF
40C6’

Step 8. Calculate SHA-1 digest of M:

H = SHA-1(M) = ‘C063AA1E6D22FBD976AB0FE73D94D2D9
C6D88127’

Step 9.[2] Construct the message representative:

F = ‘6A’ | M_1 | H | T =
‘6A9D2784A67F8E7C659973EA1AEA25D9
5B6C8F91E5002F369F0FBDCE8A3CEC19
91B543F1696546C5524CF23A5303CD6C
98599F40B79F377B5F3A1406B3B4D8F9
6784D23AA88DB7E1032A405E69325FA9
1A6E86F5C71AEA978264C4A207446DAD
4E7292E2DCDA3024B47DA8C063AA1E6D
22FBD976AB0FE73D94D2D9C6D88127BC’

Step 10. Encrypt F with the Active Authentication Private Key to form the signature:

S = ‘756B683B036A6368F4A2EB29EA700F96
E26100AFC0809F60A91733BA29CAB362
8CB1A017190A85DADE83F0B977BB513F
C9C672E5C93EFEBBE250FE1B722C7CEE
F35D26FC8F19219C92D362758FA8CB0F
F68CEF320A8753913ED25F69F7CEE772
6923B2C43437800BBC9BC028C49806CF
2E47D16AE2B2CC1678F2A4456EF98FC9’

Step 11. Construct response data for INTERNAL AUTHENTICATE and send response APDU to the inspection system:

Response APDU:

Response data field	**SW1-SW2**
S	9000

2 Since the known part (RND.IFD) is not returned, but must be appended by the IFD itself, Partial Recovery applies (‘6A’).

Inspection system:

Step 12. Decrypt the signature with the public key:

F = ‘6A9D2784A67F8E7C659973EA1AEA25D9
5B6C8F91E5002F369F0FBDCE8A3CEC19
91B543F1696546C5524CF23A5303CD6C
98599F40B79F377B5F3A1406B3B4D8F9
6784D23AA88DB7E1032A405E69325FA9
1A6E86F5C71AEA978264C4A207446DAD
4E7292E2DCDA3024B47DA8C063AA1E6D
22FBD976AB0FE73D94D2D9C6D88127BC’

Step 13. Determine hash algorithm by trailer T*:

T = ‘BC’ (i.e. SHA-1)

Step 14. Extract digest:

D = ‘C063AA1E6D22FBD976AB0FE73D94D2D9
C6D88127’

Step 15. Extract M_1:

M_1 =‘9D2784A67F8E7C659973EA1AEA25D95B
6C8F91E5002F369F0FBDCE8A3CEC1991
B543F1696546C5524CF23A5303CD6C98
599F40B79F377B5F3A1406B3B4D8F967
84D23AA88DB7E1032A405E69325FA91A
6E86F5C71AEA978264C4A207446DAD4E
7292E2DCDA3024B47DA8’

Step 16. Header indicates partial recovery but signature has modulus length so concatenate M_1 with known M_2 (i.e. RND.IFD):

M* = ‘9D2784A67F8E7C659973EA1AEA25D95B
6C8F91E5002F369F0FBDCE8A3CEC1991
B543F1696546C5524CF23A5303CD6C98
599F40B79F377B5F3A1406B3B4D8F967
84D23AA88DB7E1032A405E69325FA91A
6E86F5C71AEA978264C4A207446DAD4E
7292E2DCDA3024B47DA8F173589974BF
40C6’

Step 17. Calculate SHA-1 digest of M*:

D* =‘C063AA1E6D22FBD976AB0FE73D94D2D9
C6D88127’

Step 18. Compare D and D*:

D is equal to D* so verification successful.

— — — — — — — —

Appendix G to Part 11

WORKED EXAMPLE: PACE – GENERIC MAPPING (INFORMATIVE)

This Appendix provides two worked examples for the PACE protocol as defined in Section 4.4 using the generic mapping. The first example is based on ECDH while the second one uses DH. All numbers contained in the tables are noted hexadecimal.

In both examples, the MRZ is used as password. This also leads to the same symmetric key K_{π}. The relevant data fields of the MRZ including the check digits are:

- Serial Number: T220001293;
- Date of Birth: 6408125;
- Date of Expiry: 1010318.

Hence, the encoding K of the MRZ and the derived encryption key K_{π} are

K	`7E2D2A41 C74EA0B3 8CD36F86 3939BFA8 E9032AAD`
K_{π}	`89DED1B2 6624EC1E 634C1989 302849DD`

G.1 ECDH BASED EXAMPLE

This example is based on ECDH applying the standardized BrainpoolP256r1 domain parameters (see [RFC 5639]).

The first section introduces the corresponding `PACEInfo`. Subsequently, the exchanged APDUs including all generated nonces and ephemeral keys are listed and examined.

Elliptic Curve Parameters

Using standardized domain parameters, all information required to perform PACE is given by the data structure `PACEInfo`. In particular, no `PACEDomainParameterInfo` is needed.

PACEInfo	`3012060A 04007F00 07020204 02020201 0202010D`

The detailed structure of `PACEInfo` is itemized in the following table.

Tag	Length	Value	ASN.1 Type	Comment
30	12		SEQUENCE	PACEInfo
06	0A	04 00 7F 00 07 02 02 04 02 02	OBJECT IDENTIFIER	PACE with ECDH, generic mapping and AES 128 session keys
02	01	02	INTEGER	Version 2
02	01	0D	INTEGER	Brainpool P256r1 Standardized Domain Parameters

For convenience, an ASN.1 encoding of the BrainpoolP256r1domain parameters is given below.

Tag	Length	Value	ASN.1 Type	Comment
30	81 EC		SEQUENCE	Domain parameter
06	0A	2A 86 48 CE 3D 02 01	OBJECT IDENTIFIER	Algorithm id-ecPublicKey
30	81 E0		SEQUENCE	Domain Parameter
02	01	01	INTEGER	Version
30	2C		SEQUENCE	Underlying field
06	07	2A 86 48 CE 3D 01 01	OBJECT IDENTIFIER	Prime field
02	21	00 A9 FB 57 DB A1 EE A9 BC 3E 66 0A 90 9D 83 8D 72 6E 3B F6 23 D5 26 20 28 20 13 48 1D 1F 6E 53 77	INTEGER	Prime p
30	44		SEQUENCE	Curve equation
04	20	7D 5A 09 75 FC 2C 30 57 EE F6 75 30 41 7A FF E7 FB 80 55 C1 26 DC 5C 6C E9 4A 4B 44 F3 30 B5 D9	OCTET STRING	Parameter a
04	20	26 DC 5C 6C E9 4A 4B 44 F3 30 B5 D9 BB D7 7C BF 95 84 16 29 5C F7 E1 CE 6B CC DC 18 FF 8C 07 B6	OCTET STRING	Parameter b

Tag	***Length***	***Value***	***ASN.1 Type***	***Comment***
04	41		OCTET STRING	Group generator G
		04	-	Uncompressed point
		8B D2 AE B9 CB 7E 57 CB 2C 4B 48 2F FC 81 B7 AF B9 DE 27 E1 E3 BD 23 C2 3A 44 53 BD 9A CE 32 62	-	x-coordinate
		54 7E F8 35 C3 DA C4 FD 97 F8 46 1A 14 61 1D C9 C2 77 45 13 2D ED 8E 54 5C 1D 54 C7 2F 04 69 97	-	y-coordinate
02	21	00 A9 FB 57 DB A1 EE A9 BC 3E 66 0A 90 9D 83 8D 71 8C 39 7A A3 B5 61 A6 F7 90 1E 0E 82 97 48 56 A7	INTEGER	Group order n
02	01	01	INTEGER	Cofactor f

Application flow of the ECDH-based example

To initialize PACE, the terminal sends the command MSE:AT to the chip.

T>C :	`00 22 C1 A4 0F 80 0A 04 00 7F 00 07 02 02 04 02 02 83 01 01`
C>T :	`90 00`

Here, T>C is an abbreviation for an APDU sent from terminal to chip while C>T denotes the corresponding response sent by the chip to the terminal. The encoding of the command is explained in the next table.

Command				
CLA	00		Plain	
INS	22		Manage security environment	
P1/P2	C1 A4		Set Authentication Template for mutual authentication	
Lc	0F		Length of data field	
Data	Tag	Length	Value	Comment
	80	0A	04 00 7F 00 07 02 02 04 02 02	Cryptographic mechanism: PACE with ECDH, generic mapping and AES128 session keys
	83	01	01	Password: MRZ

Response		
Status Bytes	90 00	Normal operation

Encrypted Nonce

Next, the chip randomly generates the nonce s and encrypts it by means of K_π.

Decrypted Nonce s	3F00C4D3 9D153F2B 2A214A07 8D899B22
Encrypted Nonce z	95A3A016 522EE98D 01E76CB6 B98B42C3

The encrypted nonce is queried by the terminal.

T>C :	10 86 00 00 02 7C 00 00
C>T :	7C 12 80 10 95 A3 A0 16 52 2E E9 8D 01 E7 6C B6 B9 8B 42 C3 90 00

The encoding of the command APDU and the corresponding response can be found in the following table.

Command				
CLA	10		Command chaining	
INS	86		General Authenticate	
P1/P2	00 00		Keys and protocol implicitly known	
Lc	02		Length of data	
Data	Tag	Length	Value	Comment
	7C	00	-	Absent
Le	00		Expected maximal byte length of the response data field is 256	
Response				
Data	Tag	Length	Value	Comment
	7C	12		Dynamic Authentication Data
	80	10	95 A3 A0 16 52 2E E9 8D 01 E7 6C B6 B9 8B 42 C3	Encrypted Nonce
Status Bytes	90 00		Normal operation	

Map Nonce

The nonce is mapped to an ephemeral group generator via generic mapping. The required randomly chosen ephemeral keys are also collected in the next table.

Terminal's Private Key	7F4EF07B 9EA82FD7 8AD689B3 8D0BC78C F21F249D 953BC46F 4C6E1925 9C010F99
Terminal's Public Key	7ACF3EFC 982EC455 65A4B155 129EFBC7 4650DCBF A6362D89 6FC70262 E0C2CC5E, 544552DC B6725218 799115B5 5C9BAA6D 9F6BC3A9 618E70C2 5AF71777 A9C4922D
Chip's Private Key	498FF497 56F2DC15 87840041 839A8598 2BE7761D 14715FB0 91EFA7BC E9058560
Chip's Public Key	824FBA91 C9CBE26B EF53A0EB E7342A3B F178CEA9 F45DE0B7 0AA60165 1FBA3F57, 30D8C879 AAA9C9F7 3991E61B 58F4D52E B87A0A0C 709A49DC 63719363 CCD13C54
Shared secret H	60332EF2 450B5D24 7EF6D386 8397D398 852ED6E8 CAF6FFEE F6BF85CA 57057FD5, 0840CA74 15BAF3E4 3BD414D3 5AA4608B 93A2CAF3 A4E3EA4E 82C9C13D 03EB7181
Mapped generator $\hat{G}$	8CED63C9 1426D4F0 EB1435E7 CB1D74A4 6723A0AF 21C89634 F65A9AE8 7A9265E2, 8C879506 743F8611 AC33645C 5B985C80 B5F09A0B 83407C1B 6A4D857A E76FE522

The following APDUs are exchanged by terminal and chip to map the nonce.

T>C :	10 86 00 00 45 7C 43 81 41 04 7A CF 3E FC 98 2E C4 55 65 A4 B1 55 12 9E FB C7 46 50 DC BF A6 36 2D 89 6F C7 02 62 E0 C2 CC 5E 54 45 52 DC B6 72 52 18 79 91 15 B5 5C 9B AA 6D 9F 6B C3 A9 61 8E 70 C2 5A F7 17 77 A9 C4 92 2D 00
C>T :	7C 43 82 41 04 82 4F BA 91 C9 CB E2 6B EF 53 A0 EB E7 34 2A 3B F1 78 CE A9 F4 5D E0 B7 0A A6 01 65 1F BA 3F 57 30 D8 C8 79 AA A9 C9 F7 39 91 E6 1B 58 F4 D5 2E B8 7A 0A 0C 70 9A 49 DC 63 71 93 63 CC D1 3C 54 90 00

The structure of the APDUs can be described as follows:

Command				
CLA	10		Command chaining	
INS	86		General Authenticate	
P1/P2	00 00		Keys and protocol implicitly known	
Lc	45		Length of data	
Data	Tag	Length	Value	Comment
	7C	43	-	Dynamic Authentication Data
	81	41		Mapping Data
			04	Uncompressed Point
			7A CF 3E FC 98 2E ... C2 CC 5E	x-coordinate
			54 45 52 DC B6 72 ... C4 92 2D	y-coordinate
Le	00		Expected maximal byte length of the response data field is 256	
Response				
Data	Tag	Length	Value	Comment
	7C	43		Dynamic Authentication Data
	82	41		Mapping Data
			04	Uncompressed Point
			82 4F BA 91 C9 CB ... BA 3F 57	x-coordinate
			30 D8 C8 79 AA A9 ... D1 3C 54	y-coordinate
Status Bytes	90 00		Normal operation	

Perform Key Agreement

In the third step, chip and terminal perform an anonymous ECDH key agreement using the new domain parameters determined by the ephemeral group generator of the previous step. Only the x-coordinate is required as shared secret since the KDF uses only the first coordinate to derive the session keys.

Terminal's Private Key	A73FB703 AC1436A1 8E0CFA5A BB3F7BEC 7A070E7A 6788486B EE230C4A 22762595
Terminal's Public Key	2DB7A64C 0355044E C9DF1905 14C625CB A2CEA487 54887122 F3A5EF0D 5EDD301C, 3556F3B3 B186DF10 B857B58F 6A7EB80F 20BA5DC7 BE1D43D9 BF850149 FBB36462
Chip's Private Key	107CF586 96EF6155 053340FD 633392BA 81909DF7 B9706F22 6F32086C 7AFF974A
Chip's Public Key	9E880F84 2905B8B3 181F7AF7 CAA9F0EF B743847F 44A306D2 D28C1D9E C65DF6DB, 7764B222 77A2EDDC 3C265A9F 018F9CB8 52E111B7 68B32690 4B59A019 3776F094
Shared Secret	28768D20 701247DA E81804C9 E780EDE5 82A9996D B4A31502 0B273319 7DB84925

The key agreement is performed as follows:

T>C :	10 86 00 00 45 7C 43 83 41 04 2D B7 A6 4C 03 55 04 4E C9 DF 19 05 14 C6 25 CB A2 CE A4 87 54 88 71 22 F3 A5 EF 0D 5E DD 30 1C 35 56 F3 B3 B1 86 DF 10 B8 57 B5 8F 6A 7E B8 0F 20 BA 5D C7 BE 1D 43 D9 BF 85 01 49 FB B3 64 62 00
C>T :	7C 43 84 41 04 9E 88 0F 84 29 05 B8 B3 18 1F 7A F7 CA A9 F0 EF B7 43 84 7F 44 A3 06 D2 D2 8C 1D 9E C6 5D F6 DB 77 64 B2 22 77 A2 ED DC 3C 26 5A 9F 01 8F 9C B8 52 E1 11 B7 68 B3 26 90 4B 59 A0 19 37 76 F0 94 90 00

The encoding of the key agreement is examined in the following table:

Command					
CLA	10		Command chaining		
INS	86		General Authenticate		
P1/P2	00 00		Keys and protocol implicitly known		
Lc	45		Length of data		
Data	Tag	Length	Value	Comment	
	7C	43	-	Dynamic Authentication Data	
	83	41		Terminal's Ephemeral Public Key	
			04		Uncompressed Point

			2D B7 A6 4C 03 55 ... DD 30 1C		x-coordinate
			35 56 F3 B3 B1 86 ... B3 64 62		y-coordinate
Le	00		Expected maximal byte length of the response data field is 256		
Response					
Data	Tag	Length	Value	Comment	
	7C	43		Dynamic Authentication Data	
	84	41		Chip's Ephemeral Public Key	
			04		Uncompressed Point
			9E 88 0F 84 29 05 ... 5D F6 DB		x-coordinate
			77 64 B2 22 77 A2 ... 76 F0 94		y-coordinate
Status Bytes	90 00		Normal operation		

By means of the KDF, the AES 128 session keys KS_{Enc} and KS_{MAC} are derived from the shared secret. These are

KS_{Enc}	F5F0E35C 0D7161EE 6724EE51 3A0D9A7F
KS_{MAC}	FE251C78 58B356B2 4514B3BD 5F4297D1

Mutual Authentication

The authentication tokens are derived by means of KS_{MAC} using

Input Data for T_{PCD}	7F494F06 0A04007F 00070202 04020286 41049E88 0F842905 B8B3181F 7AF7CAA9 F0EFB743 847F44A3 06D2D28C 1D9EC65D F6DB7764 B22277A2 EDDC3C26 5A9F018F 9CB852E1 11B768B3 26904B59 A0193776 F094
Input Data for T_{IC}	7F494F06 0A04007F 00070202 04020286 41042DB7 A64C0355 044EC9DF 190514C6 25CBA2CE A4875488 7122F3A5 EF0D5EDD 301C3556 F3B3B186 DF10B857 B58F6A7E B80F20BA 5DC7BE1D 43D9BF85 0149FBB3 6462

as input. The encoding of the input data is shown below

Tag	***Length***	***Value***	***ASN.1 Type***	***Comment***
7F49	4F		PUBLIC KEY	Input data for T_{PCD}
06	0A	04 00 7F 00 07 02 02 04 02 02	OBJECT IDENTIFIER	PACE with ECDH, generic mapping and AES 128 session keys
86	41		ELLIPTIC CURVE POINT	Chip's Ephemeral Public Point
		04		Uncompressed Point
		9E 88 0F 84 29 ... 5D F6 DB		x-coordinate
		77 64 B2 22 77 ... 76 F0 94		y-coordinate

Tag	***Length***	***Value***	***ASN.1 Type***	***Comment***
7F49	4F		PUBLIC KEY	Input data for T_{IC}
06	0A	04 00 7F 00 07 02 02 04 02 02	OBJECT IDENTIFIER	PACE with ECDH, generic mapping and AES 128 session keys
86	41		ELLIPTIC CURVE POINT	Terminal's Ephemeral Public Point
		04		Uncompressed Point
		2D B7 A6 4C 03 ... DD 30 1C		x-coordinate
		35 56 F3 B3 B1 ... B3 64 62		y-coordinate

The computed authentication tokens are:

T_{PCD}	`C2B0BD78 D94BA866`
T_{IC}	`3ABB9674 BCE93C08`

Finally, these tokens are exchanged and verified.

T>C :	`00 86 00 00 0C 7C 0A 85 08 C2 B0 BD 78 D9 4B A8 66 00`
C>T :	`7C 0A 86 08 3A BB 96 74 BC E9 3C 08 90 00`

G.2 DH BASED EXAMPLE

The second example is based on DH using the 1024-bit MODP Group with 160-bit Prime Order Subgroup specified by [RFC 5114]. The parameters of the group are:

Prime *p*	B10B8F96 A080E01D DE92DE5E AE5D54EC 52C99FBC FB06A3C6 9A6A9DCA 52D23B61 6073E286 75A23D18 9838EF1E 2EE652C0 13ECB4AE A9061123 24975C3C D49B83BF ACCBDD7D 90C4BD70 98488E9C 219A7372 4EFFD6FA E5644738 FAA31A4F F55BCCC0 A151AF5F 0DC8B4BD 45BF37DF 365C1A65 E68CFDA7 6D4DA708 DF1FB2BC 2E4A4371
Subgroup Generator *g*	A4D1CBD5 C3FD3412 6765A442 EFB99905 F8104DD2 58AC507F D6406CFF 14266D31 266FEA1E 5C41564B 777E690F 5504F213 160217B4 B01B886A 5E91547F 9E2749F4 D7FBD7D3 B9A92EE1 909D0D22 63F80A76 A6A24C08 7A091F53 1DBF0A01 69B6A28A D662A4D1 8E73AFA3 2D779D59 18D08BC8 858F4DCE F97C2A24 855E6EEB 22B3B2E5
Prime Order *q* of *g*	F518AA87 81A8DF27 8ABA4E7D 64B7CB9D 49462353

The first section introduces the `PACEInfo`. Subsequently, the exchanged APDUs including all generated nonces and ephemeral keys are listed and examined.

Diffie Hellman Parameters

The relevant information for PACE is given by the data structure `PACEInfo`.

PACEInfo	3012060A 04007F00 07020204 01020201 02020100

The detailed structure of `PACEInfo` is:

Tag	***Length***	***Value***	***ASN.1 Type***	***Comment***	
30	12		SEQUENCE	PACEInfo	
06	0A	04 00 7F 00 07 02 02 04 01 02	OBJECT IDENTIFIER		OID: PACE with DH, generic mapping and AES 128 session keys
02	01	02	INTEGER		Version 2
02	01	00	INTEGER		Standardized 1024-bit Group specified by RFC 5114

Application flow of the DH-based example

To initialize PACE, the terminal sends the command MSE:AT to the chip.

T>C :	`00 22 C1 A4 0F 80 0A 04 00 7F 00 07 02 02 04 01 02 83 01 01`
C>T :	`90 00`

The encoding of the command is described in the next table.

Command				
CLA	00		Plain	
INS	22		Manage security environment	
P1/P2	C1 A4		Set Authentication Template for mutual authentication	
Lc	0F		Length of data field	
Data	Tag	Length	Value	Comment
	80	0A	04 00 7F 00 07 02 02 04 01 02	OID: Cryptographic mechanism: PACE with DH, generic mapping and AES128
	83	01	01	Password: MRZ
Response				
Status Bytes	90 00		Normal operation	

Encrypted Nonce

Next, the terminal queries a nonce from the chip.

Decrypted Nonce s	`FA5B7E3E 49753A0D B9178B7B 9BD898C8`
Encrypted Nonce z	`854D8DF5 827FA685 2D1A4FA7 01CDDDCA`

The communication looks as follows.

T>C :	`10 86 00 00 02 7C 00 00`
C>T :	`7C 12 80 10 85 4D 8D F5 82 7F A6 85 2D 1A 4F A7 01 CD DD CA 90 00`

The encoding of the command APDU and the corresponding response is described in the following table.

Command				
CLA	10		Command chaining	
INS	86		General Authenticate	
P1/P2	00 00		Keys and protocol implicitly known	
Lc	02		Length of data	
Data	Tag	Length	Value	Comment
	7C	00	-	Absent
Le	00		Expected maximal byte length of the response data field is 256	
Response				
Data	Tag	Length	Value	Comment
	7C	12		Dynamic Authentication Data
	80	10	85 4D 8D F5 82 7F A6 85 2D 1A 4F A7 01 CD DD CA	Encrypted Nonce
Status Bytes	90 00		Normal operation	

Map Nonce

By means of the generic mapping, the nonce is mapped to an ephemeral group generator. For that purpose, the following ephemeral keys are randomly generated by terminal and chip.

Terminal's Private Key	24C3C0E0 A3280ECB 943345D9 DC2A7B72 539FDA6F FDF99AB7 B6CDDDD1 BE425AF3 D02C4ED0 CDD73EBB 4B2EDF8C 07FB3A35 903F72B8 4F3771F4 EBFB4952 0D61A8F7 C7FB8C9E 2ABC24BF 4FF9D8DD F381A193 80C85B62 3AB02ACB F6D220F5 12BF4065 8322AD20 9AC0BF9E 6F8DB602 D5197D25 2BF6D148 510CA1B7 40AF0F99 F33CA5F1
Terminal's Public Key	23FB3749 EA030D2A 25B278D2 A562047A DE3F01B7 4F17A154 02CB7352 CA7D2B3E B71C343D B13D1DEB CE9A3666 DBCFC920 B49174A6 02CB4796 5CAA73DC 702489A4 4D41DB91 4DE9613D C5E98C94 160551C0 DF86274B 9359BC04 90D01B03 AD54022D CB4F57FA D6322497 D7A1E28D 46710F46 1AFE710F BBBC5F8B A166F431 1975EC6C
Chip's Private Key	4EC025E4 0C6D10B2 AAF6FCAC 98C4244F 57481A49 61F3ADC3 72A95E40 E0CC3555 F73CCFC6 5E9DB956 DD61B143 E0C7DC51 9E7DD8ED D8E3E46A 094CF226 4FD193D0 BC4BC05C DE6CA443 19C2439F D04A4644 3C8D0494 487F6F2F E9AC8BE9 B9EE16A3 D242668C BA4FFD42 EEAC3650 9E16B4D1 E6E8EE00 25FF8244 B190F57D 441EC328
Chip's Public Key	78879F57 225AA808 0D52ED0F C890A4B2 5336F699 AA89A2D3 A189654A F70729E6 23EA5738 B26381E4 DA19E004 706FACE7 B235C2DB F2F38748 312F3C98 C2DD4882 A41947B3 24AA1259 AC22579D B93F7085 655AF308 89DBB845 D9E6783F E42C9F24 49400306 254C8AE8 EE9DD812 A804C0B6 6E8CAFC1 4F84D825 8950A91B 44126EE6
Shared secret H	5BABEBEF 5B74E5BA 94B5C063 FDA15F1F 1CDE9487 3EE0A5D3 A2FCAB49 F258D07F 544F13CB 66658C3A FEE9E727 389BE3F6 CBBBD321 28A8C21D D6EEA3CF 7091CDDF B08B8D00 7D40318D CCA4FFBF 51208790 FB4BD111 E5A968ED 6B6F08B2 6CA87C41 0B3CE0C3 10CE104E ABD16629 AA48620C 1279270C B0750C0D 37C57FFF E302AE7F

Mapped generator $\hat{G}$	7C9CBFE9 8F9FBDDA 8D143506 FA7D9306 F4CB17E3 C71707AF F5E1C1A1 23702496 84D64EE3 7AF44B8D BD9D45BF 6023919C BAA027AB 97ACC771 666C8E98 FF483301 BFA4872D EDE9034E DFACB708 14166B7F 36067682 9B826BEA 57291B5A D69FBC84 EF1E7790 32A30580 3F743417 93E86974 2D401325 B37EE856 5FFCDEE6 18342DC5

The following APDUs are exchanged by terminal and chip to map the nonce.

T>C :	10 86 00 00 86 7C 81 83 81 81 80 23 FB 37 49 EA 03 0D 2A 25 B2 78 D2 A5 62 04 7A DE 3F 01 B7 4F 17 A1 54 02 CB 73 52 CA 7D 2B 3E B7 1C 34 3D B1 3D 1D EB CE 9A 36 66 DB CF C9 20 B4 91 74 A6 02 CB 47 96 5C AA 73 DC 70 24 89 A4 4D 41 DB 91 4D E9 61 3D C5 E9 8C 94 16 05 51 C0 DF 86 27 4B 93 59 BC 04 90 D0 1B 03 AD 54 02 2D CB 4F 57 FA D6 32 24 97 D7 A1 E2 8D 46 71 0F 46 1A FE 71 0F BB BC 5F 8B A1 66 F4 31 19 75 EC 6C 00
C>T :	7C 81 83 82 81 80 78 87 9F 57 22 5A A8 08 0D 52 ED 0F C8 90 A4 B2 53 36 F6 99 AA 89 A2 D3 A1 89 65 4A F7 07 29 E6 23 EA 57 38 B2 63 81 E4 DA 1 9E0 04 70 6F AC E7 B2 35 C2 DB F2 F3 87 48 31 2F 3C 98 C2 DD 48 82 A4 19 47 B3 24 AA 12 59 AC 22 57 9D B9 3F 70 85 65 5A F3 08 89 DB B8 45 D9 E6 78 3F E4 2C 9F 24 49 40 03 06 25 4C 8A E8 EE 9D D8 12 A8 04 C0 B6 6E 8C AF C1 4F 84 D8 25 89 50 A9 1B 44 12 6E E6 90 00

The structure of the APDUs can be described as follows:

Command				
CLA	10		Command chaining	
INS	86		General Authenticate	
P1/P2	00 00		Keys and protocol implicitly known	
Lc	86		Length of data	
Data	Tag	Length	Value	Comment
	7C	81 83	-	Dynamic Authentication Data
	81	81 80	23 FB 37 49 EA 03 ... 75 EC 6C	Mapping Data
Le	00		Expected maximal byte length of the response data field is 256	

Response				
Data	Tag	Length	Value	Comment
	7C	81 83		Dynamic Authentication Data
	82	81 80	ED 0F C8 90 A4 B2 ... 12 6E E6	Mapping Data
Status Bytes	90 00		Normal operation	

Perform Key Agreement

Subsequently, chip and terminal perform an anonymous DH key agreement using the new domain parameters determined by the ephemeral group generator of the previous step.

Terminal's Private Key	4BD0E547 40F9A028 E6A515BF DAF96784 8C4F5F5F FF65AA09 15947FFD 1A0DF2FA 6981271B C905F355 1457B7E0 3AC3B806 6DE4AA40 6C1171FB 43DD939C 4BA16175 103BA3DE E16419AA 248118F9 0CC36A3D 6F4C3736 52E0C3CC E7F0F1D0 C5425B36 00F0F0D6 A67F004C 8BBA33F2 B4733C72 52445C1D FC4F1107 203F71D2 EFB28161
Terminal's Public Key	00907D89 E2D425A1 78AA81AF 4A7774EC 8E388C11 5CAE6703 1E85EECE 520BD911 551B9AE4 D04369F2 9A02626C 86FBC674 7CC7BC35 2645B616 1A2A42D4 4EDA80A0 8FA8D61B 76D3A154 AD8A5A51 786B0BC0 71470578 71A92221 2C5F67F4 31731722 36B7747D 1671E6D6 92A3C7D4 0A0C3C5C E397545D 015C175E B5130551 EDBC2EE5 D4
Chip's Private Key	020F018C 7284B047 FA7721A3 37EFB7AC B1440BB3 0C5252BD 41C97C30 C994BB78 E9F0C5B3 2744D840 17D21FFA 6878396A 6469CA28 3EF5C000 DAF7D261 A39AB886 0ED4610A B5343390 897AAB5A 7787E4FA EFA0649C 6A94FDF8 2D991E8E 3FC332F5 142729E7 040A3F7D 5A4D3CD7 5CBEE1F0 43C1CAD2 DD484FEB 4ED22B59 7D36688E
Chip's Public Key	075693D9 AE941877 573E634B 6E644F8E 60AF17A0 076B8B12 3D920107 4D36152B D8B3A213 F53820C4 2ADC79AB 5D0AEEC3 AEFB9139 4DA476BD 97B9B14D 0A65C1FC 71A0E019 CB08AF55 E1F72900 5FBA7E3F A5DC4189 9238A250 767A6D46 DB974064 386CD456 743585F8 E5D90CC8 B4004B1F 6D866C79 CE0584E4 9687FF61 BC29AEA1

Shared Secret	6BABC7B3 A72BCD7E A385E4C6 2DB2625B D8613B24 149E146A 629311C4 CA6698E3 8B834B6A 9E9CD718 4BA8834A FF5043D4 36950C4C 1E783236 7C10CB8C 314D40E5 990B0DF7 013E64B4 549E2270 923D06F0 8CFF6BD3 E977DDE6 ABE4C31D 55C0FA2E 465E553E 77BDF75E 3193D383 4FC26E8E B1EE2FA1 E4FC97C1 8C3F6CFF FE2607FD

The key agreement is performed as follows:

T>C :	10 86 00 00 86 7C 81 83 83 81 80 90 7D 89 E2 D4 25 A1 78 AA 81 AF 4A 77 74 EC 8E 38 8C 11 5C AE 67 03 1E 85 EE CE 52 0B D9 11 55 1B 9A E4 D0 43 69 F2 9A 02 62 6C 86 FB C6 74 7C C7 BC 35 26 45 B6 16 1A 2A 42 D4 4E DA 80 A0 8F A8 D6 1B 76 D3 A1 54 AD 8A 5A 51 78 6B 0B C0 71 47 05 78 71 A9 22 21 2C 5F 67 F4 31 73 17 22 36 B7 74 7D 16 71 E6 D6 92 A3 C7 D4 0A 0C 3C 5C E3 97 54 5D 01 5C 17 5E B5 13 05 51 ED BC 2E E5 D4 00
C>T :	7C 81 83 84 81 80 07 56 93 D9 AE 94 18 77 57 3E 63 4B 6E 64 4F 8E 60 AF 17 A0 07 6B 8B 12 3D 92 01 07 4D 36 15 2B D8 B3 A2 13 F5 38 20 C4 2A DC 79 AB 5D 0A EE C3 AE FB 91 39 4D A4 76 BD 97 B9 B1 4D 0A 65 C1 FC 71 A0 E0 19 CB 08 AF 55 E1 F7 29 00 5F BA 7E 3F A5 DC 41 89 92 38 A2 50 76 7A 6D 46 DB 97 40 64 38 6C D4 56 74 35 85 F8 E5 D9 0C C8 B4 00 4B 1F 6D 86 6C 79 CE 05 84 E4 96 87 FF 61 BC 29 AE A1 90 00

Command				
CLA	10		Command chaining	
INS	86		General Authenticate	
P1/P2	00 00		Keys and protocol implicitly known	
Lc	86		Length of data	
Data	Tag	Length	Value	Comment
	7C	81 83	-	Dynamic Authentication Data
	83	81 80	90 7D 89 E2 D4 25 ... 2E E5 D4	Terminal's Ephemeral Public Key
Le	00		Expected maximal byte length of the response data field is 256	

Response				
Data	Tag	Length	Value	Comment
	7C	81 83		Dynamic Authentication Data
	84	81 80	07 56 93 D9 AE 94 ... 29 AE A1	Chip's Ephemeral Public Key
Status Bytes	90 00		Normal operation	

The AES 128 session keys KS_{Enc} and KS_{MAC} are derived from the shared secret using the KDF.

KS_{Enc}	2F7F46AD CC9E7E52 1B45D192 FAFA9126
KS_{MAC}	805A1D27 D45A5116 F73C5446 9462B7D8

Mutual Authentication

The authentication tokens are constructed from the following input data.

Input Data for T_{PCD}	7F49818F 060A0400 7F000702 02040102 84818007 5693D9AE 94187757 3E634B6E 644F8E60 AF17A007 6B8B123D 9201074D 36152BD8 B3A213F5 3820C42A DC79AB5D 0AEEC3AE FB91394D A476BD97 B9B14D0A 65C1FC71 A0E019CB 08AF55E1 F729005F BA7E3FA5 DC418992 38A25076 7A6D46DB 97406438 6CD45674 3585F8E5 D90CC8B4 004B1F6D 866C79CE 0584E496 87FF61BC 29AEA1
Input Data for T_{IC}	7F49818F 060A0400 7F000702 02040102 84818090 7D89E2D4 25A178AA 81AF4A77 74EC8E38 8C115CAE 67031E85 EECE520B D911551B 9AE4D043 69F29A02 626C86FB C6747CC7 BC352645 B6161A2A 42D44EDA 80A08FA8 D61B76D3 A154AD8A 5A51786B 0BC07147 057871A9 22212C5F 67F43173 172236B7 747D1671 E6D692A3 C7D40A0C 3C5CE397 545D015C 175EB513 0551EDBC 2EE5D4

The encoding of the input data is shown below:

Tag	***Length***	***Value***	***ASN.1 Type***	***Comment***
7F49	81 8F		PUBLIC KEY	Input data for T_{PCD}
06	0A	04 00 7F 00 07 02 02 04 01 02	OBJECT IDENTIFIER	PACE with DH, generic mapping and AES 128 session keys
84	81 80	07 56 93 D9 AE ... 29 AE A1	UNSIGNED INTEGER	Chip's Ephemeral Public Key

Tag	***Length***	***Value***	***ASN.1 Type***	***Comment***
7F49	81 8F		PUBLIC KEY	Input data for T_{IC}
06	0A	04 00 7F 00 07 02 02 04 01 02	OBJECT IDENTIFIER	PACE with DH, generic mapping and AES 128 session keys
84	81 80	90 7D 89 E2 D4 ... 2E E5 D4	UNSIGNED INTEGER	Terminal's Ephemeral Public Key

The computed authentication tokens are:

T_{PCD}	B46DD9BD 4D98381F
T_{IC}	917F37B5 C0E6D8D1

Finally, these tokens are exchanged and verified.

T>C :	00 86 00 00 0C 7C 0A 85 08 B4 6D D9 BD 4D 98 38 1F 00
C>T :	7C 1B 86 08 91 7F 37 B5 C0 E6 D8 D1 87 0F 44 45 54 45 53 54 43 56 43 41 30 30 30 30 33

Command				
CLA	00		Plain	
INS	86		General Authenticate	
P1/P2	00 00		Keys and protocol implicitly known	
Lc	0C		Length of data	
Data	Tag	Length	Value	Comment
	7C	0A	-	Dynamic Authentication Data
	85	08	B4 6D D9 BD 4D 98 38 1F	Terminal's Authentication Token
Le	00		Expected maximal byte length of the response data field is 256	
Response				
Data	Tag	Length	Value	Comment
	7C	0A		Dynamic Authentication Data
	86	08	91 7F 37 B5 C0 E6 D8 D1	Chip's Authentication Token
Status Bytes	90 00		Normal operation	

— — — — — — — —

Appendix H to Part 11

WORKED EXAMPLE: PACE – INTEGRATED MAPPING (INFORMATIVE)

This Appendix provides two examples for the PACE protocol with Integrated Mapping. The first one is based on Elliptic Curve Diffie-Hellman (ECDH) and the second one on Diffie-Hellman (DH). The MRZ-derived key *K* from the previous Example is used.

H.1 ECDH BASED EXAMPLE

This example is based on the BrainpoolP256r1 elliptic curve. The block cipher used in this example is AES-128. For reminder, the curve parameters are the following:

Prime p	A9FB57DB A1EEA9BC 3E660A90 9D838D72 6E3BF623 D5262028 2013481D 1F6E5377
Parameter a	7D5A0975 FC2C3057 EEF67530 417AFFE7 FB8055C1 26DC5C6C E94A4B44 F330B5D9
Parameter b	26DC5C6C E94A4B44 F330B5D9 BBD77CBF 95841629 5CF7E1CE 6BCCDC18 FF8C07B6
x-coordinate of the group generator G	8BD2AEB9 CB7E57CB 2C4B482F FC81B7AF B9DE27E1 E3BD23C2 3A4453BD 9ACE3262
y-coordinate of the group generator G	547EF835 C3DAC4FD 97F8461A 14611DC9 C2774513 2DED8E54 5C1D54C7 2F046997
Group order n	A9FB57DB A1EEA9BC 3E660A90 9D838D71 8C397AA3 B561A6F7 901E0E82 974856A7
Cofactor f	01

The encryption key is the following:

K	591468CD A83D6521 9CCCB856 0233600F

Encrypted Nonce

A nonce s is randomly chosen by the chip and encrypted using K . The encrypted nonce z is then sent to the terminal.

Decrypted Nonce s	2923BE84 E16CD6AE 529049F1 F1BBE9EB
Encrypted Nonce z	143DC40C 08C8E891 FBED7DED B92B64AD

Map Nonce

A nonce t is randomly chosen and sent in clear. t and s are then used to compute the Integrated Mapping. First, the pseudo-random function R_p, derived from AES, is applied to s and t. Then, the point encoding f_G is used on the result to compute the Mapped Generator $\hat{G}=f_G(R_p(s,t))$.

Nonce t	5DD4CBFC 96F5453B 130D890A 1CDBAE32
Pseudo-random R(s,t)	E4447E2D FB3586BA C05DDB00 156B57FB B2179A39 49294C97 25418980 0C517BAA 8DA0FF39 7ED8C445 D3E421E4 FEB57322
$R_p(s,t)$	A2F8FF2D F50E52C6 599F386A DCB595D2 29F6A167 ADE2BE5F 2C3296AD D5B7430E
x-coordinate of the Mapped Generator $\hat{G}$	8E82D315 59ED0FDE 92A4D049 8ADD3C23 BABA94FB 77691E31 E90AEA77 FB17D427
y-coordinate of the Mapped Generator $\hat{G}$	4C1AE14B D0C3DBAC 0C871B7F 36081693 64437CA3 0AC243A0 89D3F266 C1E60FAD

Perform Key Agreement

The chip and the terminal perform an anonymous Diffie-Hellman key agreement using their secret keys and the mapped generator $\hat{G}$. The shared secret K is the x-coordinate of agreement.

Chip's private key SK_{IC}	107CF586 96EF6155 053340FD 633392BA 81909DF7 B9706F22 6F32086C 7AFF974A
Chip's public key PK_{IC}	67F78E5F 7F768608 2B293E8D 087E0569 16D0F74B C01A5F89 57D0DE45 691E51E8 932B69A9 62B52A09 85AD2C0A 271EE6A1 3A8ADDDC D1A3A994 B9DED257 F4D22753
Terminal's private key SK_{PCD}	A73FB703 AC1436A1 8E0CFA5A BB3F7BEC 7A070E7A 6788486B EE230C4A 22762595
Terminal's public key PK_{PCD}	89CBA23F FE96AA18 D824627C 3E934E54 A9FD0B87 A95D1471 DC1C0ABF DCD640D4 6755DE9B 7B778280 B6BEBD57 439ADFEB 0E21FD4E D6DF4257 8C13418A 59B34C37

Shared secret K	4F150FDE 1D4F0E38 E95017B8 91BAE171 33A0DF45 B0D3E18B 60BA7BEA FDC2C713

Using the specifications from [1], the session keys K_{Enc} and K_{MAC} are derived from K using the hash function SHA-1: K_{Enc}=SHA-1(K||0x00000001) and K_{MAC}=SHA-1(K||0x00000002). Then, only the first 16 octets of the digest are used with the following result:

K_{Enc}	0D3FEB33 251A6370 893D62AE 8DAAF51B
K_{MAC}	B01E89E3 D9E8719E 586B50B4 A7506E0B

Mutual Authentication

The authentication tokens are computed using a CMAC on the following inputs with the key K_{MAC}.

Input data for T_{IC}	7F494F06 0A04007F 00070202 04040286 410489CB A23FFE96 AA18D824 627C3E93 4E54A9FD 0B87A95D 1471DC1C 0ABFDCD6 40D46755 DE9B7B77 8280B6BE BD57439A DFEB0E21 FD4ED6DF 42578C13 418A59B3 4C37
Input data for T_{PCD}	7F494F06 0A04007F 00070202 04040286 410467F7 8E5F7F76 86082B29 3E8D087E 056916D0 F74BC01A 5F8957D0 DE45691E 51E8932B 69A962B5 2A0985AD 2C0A271E E6A13A8A DDDCD1A3 A994B9DE D257F4D2 2753

The corresponding authentication tokens are:

T_{IC}	75D4D96E 8D5B0308
T_{PCD}	450F02B8 6F6A0909

H.2 DH BASED EXAMPLE

This example is based on the 1024-bit MODP Group with 160-bit Prime Order Subgroup. The block cipher used in this example is AES-128.

The group parameters are:

Prime p	B10B8F96 A080E01D DE92DE5E AE5D54EC 52C99FBC FB06A3C6 9A6A9DCA 52D23B61 6073E286 75A23D18 9838EF1E 2EE652C0 13ECB4AE A9061123 24975C3C D49B83BF ACCBDD7D 90C4BD70 98488E9C 219A7372 4EFFD6FA E5644738 FAA31A4F F55BCCC0 A151AF5F 0DC8B4BD 45BF37DF 365C1A65 E68CFDA7 6D4DA708 DF1FB2BC 2E4A4371
Subgroup generator g	A4D1CBD5 C3FD3412 6765A442 EFB99905 F8104DD2 58AC507F D6406CFF 14266D31 266FEA1E 5C41564B 777E690F 5504F213 160217B4 B01B886A 5E91547F 9E2749F4 D7FBD7D3 B9A92EE1 909D0D22 63F80A76 A6A24C08 7A091F53 1DBF0A01 69B6A28A D662A4D1 8E73AFA3 2D779D59 18D08BC8 858F4DCE F97C2A24 855E6EEB 22B3B2E5
Prime order q of g	F518AA87 81A8DF27 8ABA4E7D 64B7CB9D 49462353

The following encryption key is used:

K	591468CD A83D6521 9CCCB856 0233600F

Encrypted Nonce

A nonce s is randomly chosen by the chip and encrypted using K . The encrypted nonce z is then sent to the terminal.

Decrypted Nonce s	FA5B7E3E 49753A0D B9178B7B 9BD898C8
Encrypted Nonce z	9ABB8864 CA0FF155 1E620D1E F4E13510

Map Nonce

A nonce t is randomly chosen and sent in clear. t and s are then used to compute the Integrated Mapping. First, the pseudo-random function R_p, derived from AES, is applied to s and t. Then, the point encoding f_g is used on the result.

Nonce t	B3A6DB3C 870C3E99 245E0D1C 06B747DE
Pseudo-random R(s,t)	EAB98D13 E0905295 2AA72990 7C3C9461 84DEA0FE 74AD2B3A F506F0A8 3018459C 38099CD1 F7FF4EA0 A078DB1F AC136550 5E3DC855 00EF95E2 0B4EEF2E 88489233 BEE0546B 472F994B 618D1687 02406791 DEEF3CB4 810932EC 278F3533 FDB860EB 4835C36F A4F1BF3F A0B828A7 18C96BDE 88FBA38A 3E6C35AA A1095925 1EB5FC71 0FC18725 8995944C 0F926E24 9373F485
R_p(s,t)	A0C7C50C 002061A5 1CC87D25 4EF38068 607417B6 EE1B3647 3CFB800D 2D2E5FA2 B6980F01 105D24FA B22ACD1B FA5C8A4C 093ECDFA FE6D7125 D42A843E 33860383 5CF19AFA FF75EFE2 1DC5F6AA 1F9AE46C 25087E73 68166FB0 8C1E4627 AFED7D93 570417B7 90FF7F74 7E57F432 B04E1236 819E0DFE F5B6E77C A4999925 328182D2
Mapped Generator $\hat{g} = f_g(R_p(s,t))$	1D7D767F 11E333BC D6DBAEF4 0E799E7A 926B9697 3550656F F3C83072 6D118D61 C276CDCC 61D475CF 03A98E0C 0E79CAEB A5BE2557 8BD4551D 0B109032 36F0B0F9 76852FA7 8EEA14EA 0ACA87D1 E91F688F E0DFF897 BBE35A47 2621D343 564B262F 34223AE8 FC59B664 BFEDFA2B FE7516CA 5510A6BB B633D517 EC25D4E0 BBAA16C2

Perform Key Agreement

The chip and the terminal perform an anonymous Diffie-Hellman key agreement using their secret keys and the mapped generator $\hat{g}$.

Chip's private key SK_{IC}	020F018C 7284B047 FA7721A3 37EFB7AC B1440BB3 0C5252BD 41C97C30 C994BB78 E9F0C5B3 2744D840 17D21FFA 6878396A 6469CA28 3EF5C000 DAF7D261 A39AB886 0ED4610A B5343390 897AAB5A 7787E4FA EFA0649C 6A94FDF8 2D991E8E 3FC332F5 142729E7 040A3F7D 5A4D3CD7 5CBEE1F0 43C1CAD2 DD484FEB 4ED22B59 7D36688E

Chip's public key PK_{IC}	928D9A0F 9DBA450F 13FC859C 6F290D1D 36E42431 138A4378 500BEB4E 0401854C FF111F71 CB6DC1D0 335807A1 1388CC8E AA87B079 07AAD9FB A6B169AF 6D8C26AF 8DDDC39A DC3AD2E3 FF882B84 D23E9768 E95A80E4 746FB07A 9767679F E92133B4 D379935C 771BD7FB ED6C7BB4 B1708B27 5EA75679 524CDC9C 6A91370C C662A2F3
Terminal's private key SK_{PCD}	4BD0E547 40F9A028 E6A515BF DAF96784 8C4F5F5F FF65AA09 15947FFD 1A0DF2FA 6981271B C905F355 1457B7E0 3AC3B806 6DE4AA40 6C1171FB 43DD939C 4BA16175 103BA3DE E16419AA 248118F9 0CC36A3D 6F4C3736 52E0C3CC E7F0F1D0 C5425B36 00F0F0D6 A67F004C 8BBA33F2 B4733C72 52445C1D FC4F1107 203F71D2 EFB28161
Terminal's public key PK_{PCD}	0F0CC629 45A80292 51FB7EF3 C094E12E C68E4EF0 7F27CB9D 9CD04C5C 4250FAE0 E4F8A951 557E929A EB48E5C6 DD47F2F5 CD7C351A 9BD2CD72 2C07EDE1 66770F08 FFCB3702 62CF308D D7B07F2E 0DA9CAAA 1492344C 85290691 9538C98A 4BA4187E 76CE9D87 832386D3 19CE2E04 3C3343AE AE6EDBA1 A9894DC5 094D22F7 FE1351D5
Shared secret K	419410D6 C0A17A4C 07C54872 CE1CBCEB 0A2705C1 A434C8A8 9A4CFE41 F1D78124 CA7EC52B DE7615E5 345E48AB 1ABB6E7D 1D59A57F 3174084D 3CA45703 97C1F622 28BDFDB2 DA191EA2 239E2C06 0DBE3BBC 23C2FCD0 AF12E0F9 E0B99FCF 91FF1959 011D5798 B2FCBC1F 14FCC24E 441F4C8F 9B08D977 E9498560 E63E7FFA B3134EA7

The session keys K_{Enc} and K_{MAC} are derived from K using the hash function SHA-1: K_{Enc}=SHA-1(K\|\|0x00000001) and K_{MAC}=SHA-1(K\|\|0x00000002). Then, only the first 16 octets of the digest are used with the following result:

K_{Enc}	01AFC10C F87BE36D 8179E873 70171F07
K_{MAC}	23F0FBD0 5FD6C7B8 B88F4C83 09669061

Mutual Authentication

The authentication tokens are computed using a CMAC on the following inputs with the key K_{MAC}.

Input data for T_{IC}	7F49818F 060A0400 7F000702 02040302 8481800F 0CC62945 A8029251 FB7EF3C0 94E12EC6 8E4EF07F 27CB9D9C D04C5C42 50FAE0E4 F8A95155 7E929AEB 48E5C6DD 47F2F5CD 7C351A9B D2CD722C 07EDE166 770F08FF CB370262 CF308DD7 B07F2E0D A9CAAA14 92344C85 29069195 38C98A4B A4187E76 CE9D8783 2386D319 CE2E043C 3343AEAE 6EDBA1A9 894DC509 4D22F7FE 1351D5
Input data for T_{PCD}	7F49818F 060A0400 7F000702 02040302 84818092 8D9A0F9D BA450F13 FC859C6F 290D1D36 E4243113 8A437850 0BEB4E04 01854CFF 111F71CB 6DC1D033 5807A113 88CC8EAA 87B07907 AAD9FBA6 B169AF6D 8C26AF8D DDC39ADC 3AD2E3FF 882B84D2 3E9768E9 5A80E474 6FB07A97 67679FE9 2133B4D3 79935C77 1BD7FBED 6C7BB4B1 708B275E A7567952 4CDC9C6A 91370CC6 62A2F3

The corresponding authentication tokens are:

T_{IC}	C2F04230 187E1525
T_{PCD}	55D61977 CBF5307E

— END —

ISBN 978-92-9249-799-6