EMV®
Contactless Specifications for Payment Systems

Book E

Security and Key Management

Version 1.0
June 2023

Legal Notice

The EMV® Specifications are provided “AS IS” without warranties of any kind, and EMVCo neither assumes nor accepts any liability for any errors or omissions contained in these Specifications. EMVCO DISCLAIMS ALL REPRESENTATIONS AND WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AS TO THESE SPECIFICATIONS.

EMVCo makes no representations or warranties with respect to intellectual property rights of any third parties in or in relation to the Specifications. EMVCo undertakes no responsibility to determine whether any implementation of the EMV® Specifications may violate, infringe, or otherwise exercise the patent, copyright, trademark, trade secret, know-how, or other intellectual property rights of third parties, and thus any person who implements any part of the EMV® Specifications should consult an intellectual property attorney before any such implementation.

Without limiting the foregoing, the Specifications may provide for the use of public key encryption and other technology, which may be the subject matter of patents in several countries. Any party seeking to implement these Specifications is solely responsible for determining whether its activities require a license to any such technology, including for patents on public key encryption technology. EMVCo shall not be liable under any theory for any party’s infringement of any intellectual property rights in connection with the EMV® Specifications.

Contents

Tables

Figures

1 Scope

This specification, *EMV® Contactless Specifications for Payment Systems, Book E – Contactless Security*, describes the security mechanisms of EMV contactless transactions contained in the Kernel 8 specification [EMV Book C-8] and non-C-8 EMV contactless Kernels.

1.1 Audience

This specification is intended for use by manufacturers of contactless readers and terminals. It may also be of interest to manufacturers of contactless cards and to financial institution staff responsible for implementing financial applications in contactless cards.

1.2 Related Information

The following references are used in this specification. It is noted that the latest version applies unless a publication date is explicitly stated.

Table 1.1 – Related Information

Reference	Document Title
[EMV Book C-8]	EMV® Contactless Specifications for Payment Systems, Book C-8 – Kernel 8 Specification
[EMV Book 2]	EMV® Book 2: Security and Key Management
[ISO/IEC 9796-2]	Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Integer factorization based mechanisms
[ISO/IEC 9797-1]	Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher
[ISO/IEC 10116]	Information technology – Security techniques – Modes of operation for an *n*-bit block cipher
[ISO/IEC 10118-3]	Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions
[ISO/IEC 11770-6]	Information technology – Security techniques – Key management – Part 6: Key derivation

Reference	Document Title
[ISO/IEC 14888-3]	Information technology – Security techniques – Digital signatures with appendix – Part 3: Discrete logarithm-based mechanisms
[ISO/IEC 15946-1]	IT Security techniques – Cryptographic techniques based on elliptic curves – Part 1: General
[ISO/IEC 15946-5]	IT Security techniques – Cryptographic techniques based on elliptic curves – Part 5: Elliptic curve generation
[ISO/IEC 18031]	Information technology – Security techniques – Random bit generation
[ISO/IEC 18033-3]	Information technology – Security techniques – Encryption algorithms – Part 3: Block ciphers
[NIST SP800-22A]	A statistical test suite for random and pseudorandom number generators for cryptographic algorithms
[EMV SB144]	EMV® Specification Bulletin No. 144 – Terminal Unpredictable Number generation

1.3 Terminology

The following terms are used in this specification, carrying specialised meanings as indicated.

Table 1.2 – Terminology

Term	Description
Application Cryptogram	The Application Cryptogram allows the authentication by the Issuer of a subset of the transaction data exchanged between the Reader and the Card.
Card	The Card, as used in these specifications, is a consumer device supporting contactless transactions.
Cardholder	The Cardholder is the owner of the payment Card issued by the bank that holds the designated bank account.
Certification Authority	Trusted third party that establishes proof that links a public key and other related data to its owner via a certificate.
Issuer	The Issuer refers to the bank that holds the customer's account, issuing the payment Card and accepting transactions with this Card.

Term	Description
Kernel	The Kernel contains the interface routines, security and control functions to interact with the payment Card.
Local Cryptogram	The Local Cryptogram allows the authentication by the Reader of the transaction data exchanged between the Reader and the Card.
Payment System	The Payment System refers to the entity responsible for the rules and infrastructure used to perform, process and settle financial transactions.
Reader	The Reader is the part of the payment terminal that provides the interface to the Card, via the contactless Kernel.

1.4 Abbreviations

The following abbreviations are used in this specification.

Table 1.3 – Abbreviations

Abbreviation	Description
3DES	Triple DES
AC	Application Cryptogram
AES	Advanced Encryption Standard
AFL	Application File Locator
AID	Application Identifier
AIP	Application Interchange Profile
ARQC	Authorisation Request Cryptogram
ASI	Algorithm Suite Indicator
ATC	Application Transaction Counter
BDH	Blinded Diffie-Hellman
CBC	Cipher Block Chaining
CDOL	Card Risk Management Data Object List
CID	Cryptogram Information Data
CMAC	Cipher-based Message Authentication Code
CMC	Card Message Counter
CTR	CounTeR mode

Abbreviation	Description
CVD	Cardholder Verification Decision
CVM	Cardholder Verification Method
DDOL	Dynamic Data Authentication Data Object List
DES	Data Encryption Standard
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECSDSA	Elliptic Curve Schnorr Digital Signature Algorithm
EDA	Enhanced Data Authentication
ERRD	Exchange Relay Resistance Data
fDDA	fast Dynamic Data Authentication
HSM	Hardware Security Module
IAD	Issuer Application Data
ICC	Integrated Circuit Card
IMK	Issuer Master Key
KMC	Kernel Message Counter
L_{DD}	Length of the ICC Dynamic Data
M	Mandatory
MAC	Message Authentication Code
MK	Card Master Key
N_C	Length of Card Public Key Modulus
N_{CA}	Length of Certification Authority Public Key Modulus
N_I	Length of Issuer Public Key Modulus
N_{FIELD}	Elliptic Curve Field Size
N_{HASH}	Length of the Hash Algorithm
N_{SIG}	Length of the ECC Digital Signature
O	Optional
ODA	Offline Data Authentication
PAN	Primary Account Number
PDOL	Processing Options Data Object List
PKI	Public Key Infrastructure

Abbreviation	Description
PSN	PAN Sequence Number
RID	Registered Application Provider Identifier
RMAC	Retail Message Authentication Code
RRP	Relay Resistance Protocol
RSA	Rivest Shamir Adleman Algorithm
SDA	Static Data to be Authenticated
SHA	Secure Hash Algorithm
SK	Session Key
SKD	Session Key Derivation
TC	Transaction Certificate
TLV	Tag Length Value (of a data object)
UN	Unpredictable Number
var.	Variable

1.5 Notations

The following conventions are used in this specification.

Table 1.4 – Notations

Notation	Description
'6B75'	Hexadecimal notation. Values expressed in hexadecimal form are enclosed in straight single quotes.
1001b	Binary notation. Values expressed in binary form are followed by a lower case ‘b’.
27509	Decimal notation. Values expressed in decimal form are not enclosed in single quotes.
A mod n	The reduction of the integer A modulo the integer n, that is, the unique integer r, $0 \leq r < n$, for which there exists an integer d such that $A = dn + r$. Example: 54 mod 16 = 6.
||	Two binary data objects are concatenated. Example: A = 'AB34' B = A || 'FFFF' means that B is assigned the value 'AB34FFFF'.
A ⊕ B	A XOR B. Exclusive OR of A and B.

Notation	Description
$A \cdot B$	Multiplication of A and B, which may be either a modular multiplication (if A and B are integers) or a scalar multiplication (if A is an integer and B is a point on an elliptic curve).

2 Secure Channel

This section describes the security mechanisms to establish the secure channel between the Reader and the Card for each transaction.

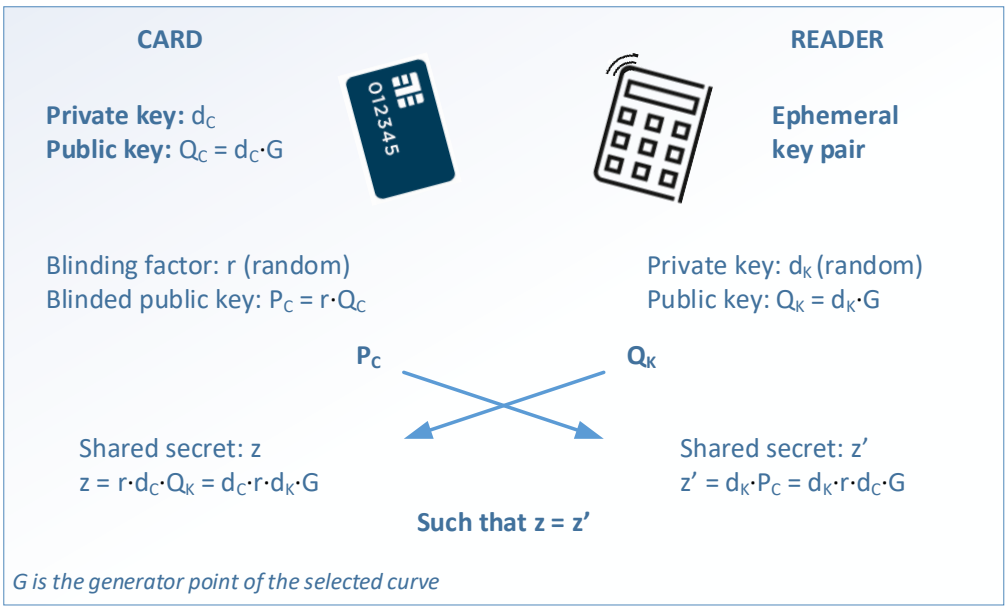
The secure channel provides:

- Privacy protection
- Secure data storage
- Local authentication, described separately in section 3.

2.1 BDH Key Agreement

The Blinded Diffie-Hellman (BDH) key agreement described in Figure 2.1 is a variant of the Elliptic Curve Diffie-Hellman (ECDH) protocol where the Reader generates an ephemeral ECC key pair while the Card uses a fixed key pair personalised by the Issuer. The Card public key is certified by the Issuer through the Card certificate.

Figure 2.1 – BDH Key Agreement



Note that the Card public key and the Card certificate correspond to the ICC ECC Public Key and the ICC ECC Public Key Certificate data objects respectively.

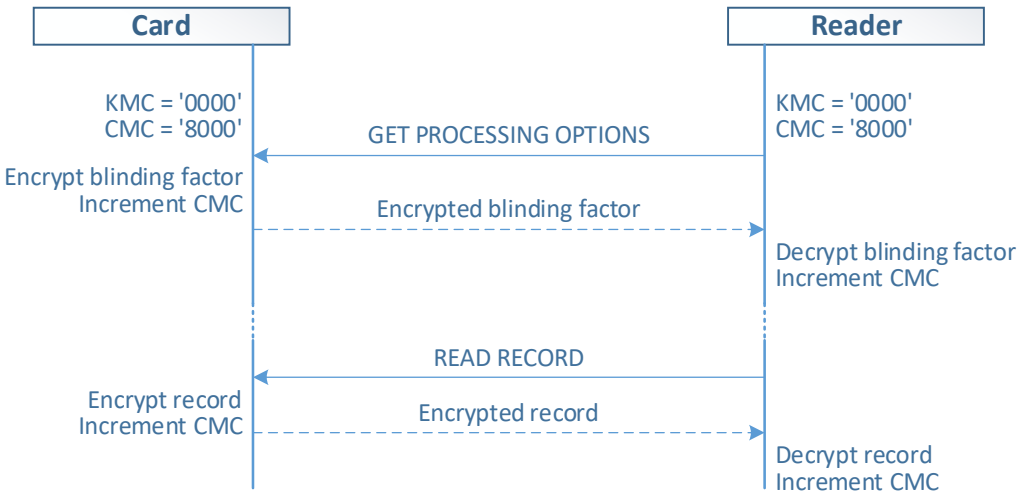
As the Card public key is fixed, for privacy purposes, it needs to be anonymised for each transaction. This is achieved by multiplying the Card public key by a random integer, called the blinding factor, $P_C = r \cdot Q_C$. The result is called the Card blinded public key. Alternatively, the Card blinded public key may be calculated as $P_C = (r \cdot d_C) \cdot G$.

As described in section 7, the Reader and Card exchange the Kernel public key and the Card blinded public key, compute a shared secret and then derive two session keys – one for confidentiality and one for integrity. The blinding factor, which is later sent encrypted by the Card, permits the Reader to authenticate the Card (in conjunction with the Card certificate), if required.

2.2 Privacy Protection

Once the secure channel is established between the Reader and the Card, sensitive data returned by the Card, as shown in Figure 2.2, is encrypted using AES-CTR with the session key for confidentiality SK_C.

Figure 2.2 – Data Encryption



Sensitive data includes the blinding factor returned by the GET PROCESSING OPTIONS command and any data returned by a READ RECORD command that uniquely identifies the Card – Application PAN, Card public key certificate, Card ECC public key, Card RSA public key remainder for instance.

Two message counters (MC) – in each device – are used with the message encryption: the Kernel Message Counter (KMC) for messages originated by the Reader, used in section 2.3, and the Card Message Counter (CMC) for messages originated by the Card.

The 2-byte counters KMC and CMC are initialised on each device to '0000' and '8000' respectively at the beginning of each transaction. CMC is incremented by the GET PROCESSING OPTIONS and READ RECORD (returning an encrypted record) commands. CMC is also incremented by the READ DATA and WRITE DATA commands, which together with KMC incrementation is described in section 2.3.

The maximum values of KMC and CMC are '7FFF' and 'FFFF' respectively (an implementation may impose lower limits to increase protection against side channel attacks).

For the GET PROCESSING OPTIONS and READ RECORD commands the message encryption is computed as follows:

Ciphertext = AES-CTR (SK_C) [CMC, Message]

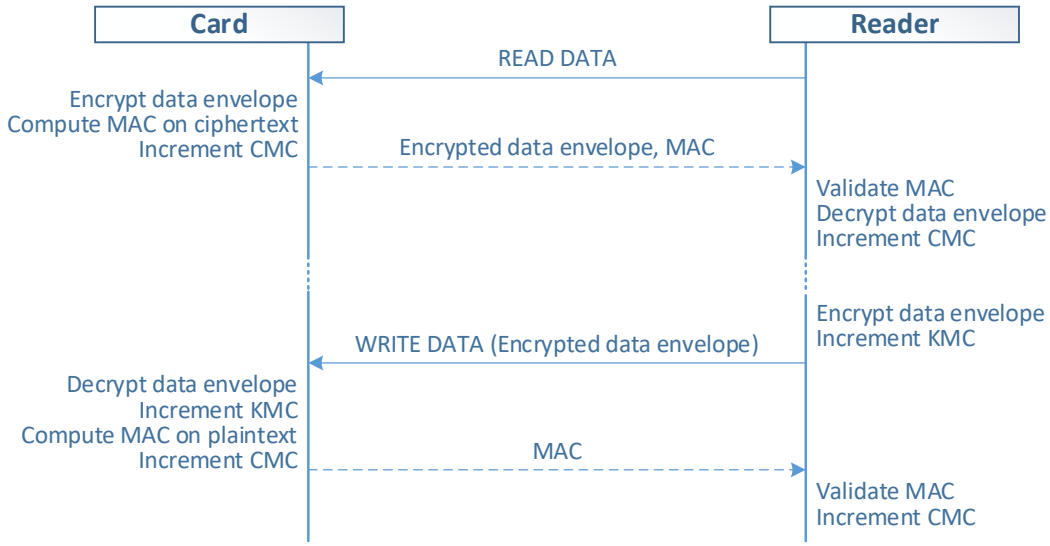
The message decryption is obtained by entering the ciphertext as the message.

The READ RECORD command is performed after the GET PROCESSING OPTIONS command, once the Card has completed the BDH key agreement, and during normal operation before the GENERATE AC command.

2.3 Secure Data Storage

The privacy protection is extended to secure the Card data storage based on the READ DATA and WRITE DATA commands. This may be used for reading from and writing to one of ten TLV encoded data envelopes, as shown in Figure 2.3.

Figure 2.3 – Data Encryption and Authentication



As mentioned in section 2.2, the message counters CMC and KMC are initialised on each device at the beginning of each transaction. CMC is incremented by the READ DATA and WRITE DATA commands. KMC is incremented only by the WRITE DATA command.

For the READ DATA command the plaintext data envelope is encrypted by the Card as follows:

Ciphertext = AES-CTR (SK_C) [CMC, Input Data]

For the WRITE DATA command the plaintext data envelope is encrypted by the Reader as follows:

Ciphertext = AES-CTR (SK_C) [KMC, Input Data]

In both cases the decryption is obtained by entering the ciphertext (the encrypted data envelope) as the Input Data.

The Card data storage also includes an authentication mechanism for the data sent or received by the Card. An 8-byte MAC is computed by the Card using AES-CMAC with the session key for integrity SK_I.

The READ DATA command response contains a MAC over the encrypted data envelope that is requested in the command message. The WRITE DATA command response contains a MAC over the plaintext data envelope decrypted by the Card.

As only the Card is generating MACs to be validated by the Reader, the MAC is computed on the concatenation of CMC and the Input Data as follows:

MAC = Leftmost 8 bytes of AES-CMAC (SK_I) [CMC || Input Data]

The READ DATA and WRITE DATA commands are performed after the GET PROCESSING OPTIONS command, once the Card has completed the BDH key agreement. During normal operation the READ DATA command is performed before the GENERATE AC command and the WRITE DATA command after, provided local authentication has been performed and is successful.

Note that to prevent collision the IAD-MAC and EDA-MAC (see section 3.2) are computed using a fixed 2-byte value of '0000' as the message counter.

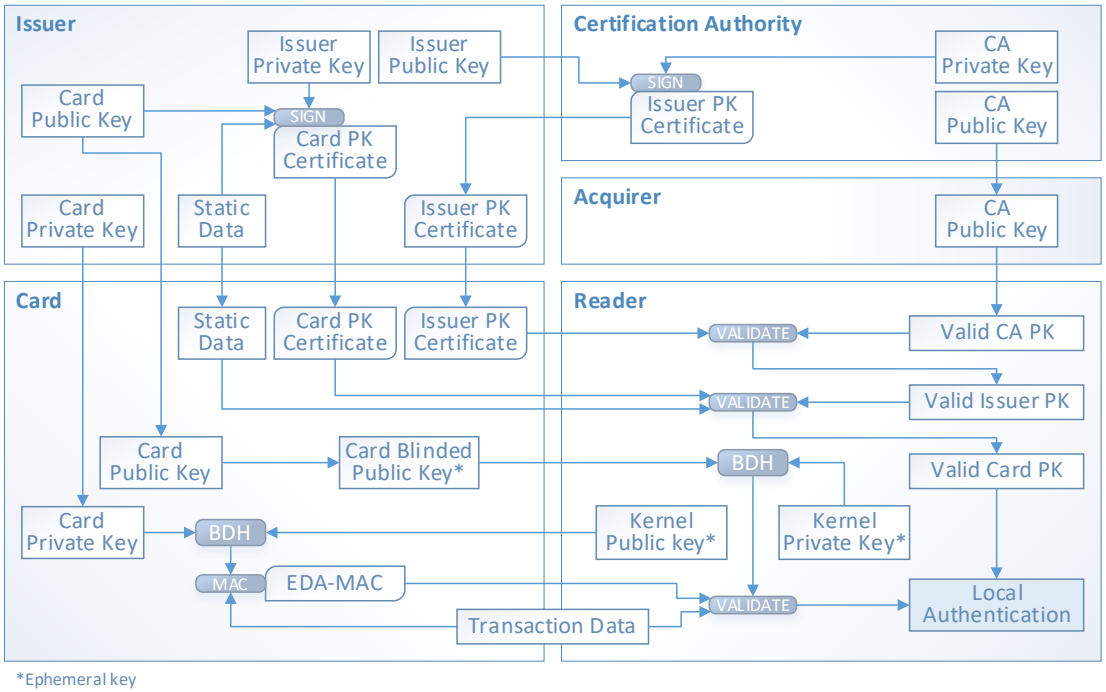
3 Local Authentication

This section describes the local authentication of the Card by the Reader when performed during a transaction, it authenticates the transaction data and ensures that the Card is genuine.

3.1 Public Key Management

Figure 3.1 describes the public key infrastructure (PKI) and local authentication of a Kernel 8 transaction. Every Card is personalised by the Issuer with a unique private/public key pair where the public key is certified by the Issuer.

Figure 3.1 – Local Authentication Diagram



A three-layer authentication scheme allows the Reader to validate the Local Cryptogram (EDA-MAC) generated by the Card:

- The Issuer public key is authenticated by the Certification Authority (CA) public key stored in the Reader.
- The Card public key is authenticated by the Issuer public key which is signed by the Certification Authority in the Issuer certificate. As mentioned in section 2.1, the Card authentication includes the authentication of the blinding factor which is sent encrypted by the Card.
- The Local Cryptogram is authenticated by the Reader using the session key for integrity SK_I obtained during the BDH key agreement, as described in section 2.1.

If the validation of the Local Cryptogram and the Issuer and Card certificates (including the blinding factor validation) is successful, the local authentication is successful and the Card is considered as genuine.

Issuer and Card certificates are personalised in the Card by the Issuer and transmitted by the Card to the Reader if local authentication is required. The Card typically holds a set of ECC certificates used to authenticate the Card ECC public key.

For migration purposes, it is possible to use instead a set of RSA certificates where the Card RSA certificate is used to authenticate the Card ECC public key. In that case the Card ECC public key is added to the Card static data signed by the Issuer.

3.2 Local Cryptogram

The Local Cryptogram is generated by the Card over the transaction data, including the Issuer Application Data, the Card Static Data to be Authenticated and the Application Cryptogram. The Application Cryptogram is generated over a subset of the transaction data as described in section 4.2.

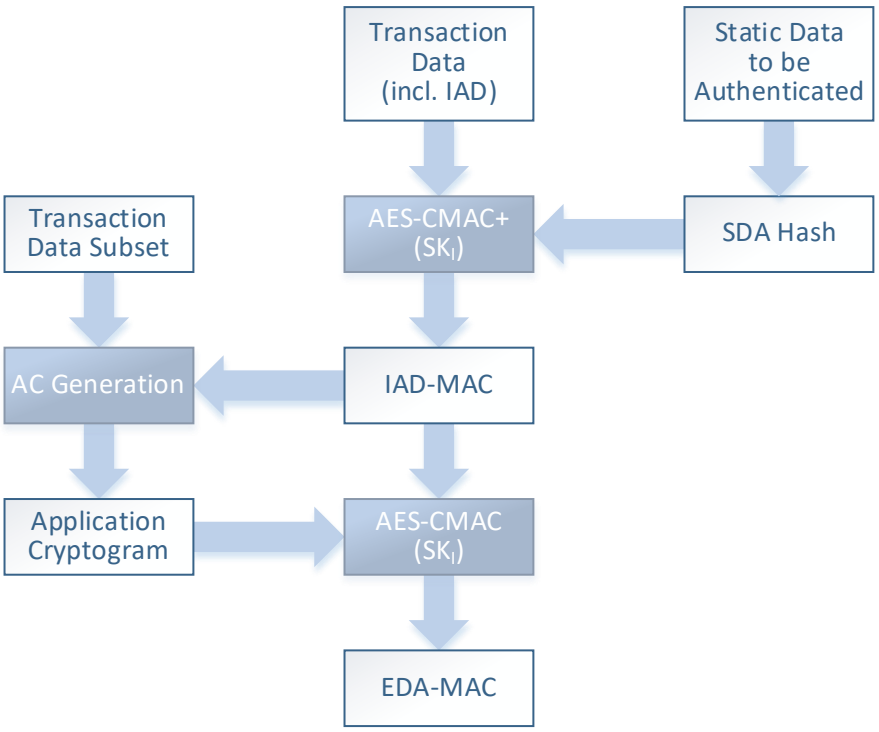
During local authentication the Reader validates the Local Cryptogram which ensures the authenticity of the transaction data, including the Application Cryptogram (which is validated by the Issuer during remote authentication).

The Local Cryptogram is returned by the Card in the response message of the GENERATE AC command with the following Card transaction data:

- The Cryptogram Information Data (CID) that informs the Reader on the type of Application Cryptogram generated.
- The Application Transaction Counter (ATC) incremented by the Card for each transaction that ensures the freshness of the Application Cryptogram and Local Cryptogram on the Card side.
- The Cardholder Verification Decision (CVD) that indicates the Cardholder Verification Method (CVM) chosen by the Card, not the result of a Cardholder verification.
- Optional data objects authenticated by the Local Cryptogram.
- The Application Cryptogram (AC).
- The Issuer Application Data (IAD), including the results of the Card verification, that informs the Issuer about the Card application during the remote authentication.

The Local Cryptogram is an AES-CMAC called Enhanced Data Authentication MAC or EDA-MAC. EDA-MAC is computed with the session key for integrity SK_I in two steps; firstly the Issuer Application Data MAC or IAD-MAC is computed and then the EDA-MAC as shown in Figure 3.2.

Figure 3.2 – Local Cryptogram Computation



The Reader transaction data needed by the Card is specified in the Processing Options Data Object List (PDOL) and Card Risk Management Data Object List (CDOL1).

PDOL indicates the data objects needed to initialise the Card application such as the Kernel Qualifier and Kernel Key Data. The Kernel Qualifier indicates the secure channel Algorithm Suite Indicator (ASI) supported by the Reader and whether the Reader supports local authentication or not. The Kernel Key Data is the Reader ephemeral ECC public key used in the BDH key agreement.

CDOL1 in Table 3.1 indicates the recommended data objects needed by the Card application to complete the transaction. Data objects already specified in the PDOL are normally not repeated in CDOL1.

Table 3.1 – Recommended CDOL1

Reference	Length
Amount, Authorised (Numeric)	6
Amount, Other (Numeric)	6
Terminal Country Code	2
Terminal Verification Results	5
Transaction Currency Code	2

Reference	Length
Transaction Date	3
Transaction Type	1
Unpredictable Number	4
CVM Results	3
Terminal Risk Management Data	8

The Unpredictable Number generated by the Reader for each transaction ensures the freshness of the Local Cryptogram on the Reader side.

The PDOL may be signed by the Issuer through the Extended SDA Tag List as described in sections 5.3 and 6.3.

Table 3.2 – IAD-MAC Input Data

Reference	Origin
PDOL Values (Value field of PDOL Related Data)	Reader
CDOL1 Related Data	Reader
Terminal Relay Resistance Entropy (1)	Reader
Last ERRD Response (without tag '80' and length '0A') (1)	Card
GENERATE AC Response Message without: • Application Cryptogram • EDA-MAC • Tag '77' and length	Card
SDA Hash (hash over the Card Static Data to be Authenticated)	Card

(1) If RRP performed

The IAD-MAC authenticates the transaction data listed in Table 3.2.

The 8-byte IAD-MAC is computed by the Card as follows:

- Concatenate '0000' with the IAD-MAC Input Data in Table 3.2. Note that the Card is personalised with SDA Hash which is the hash of at least the AIP value.
- Compute IAD-MAC by applying AES-CMAC+ over the concatenated data with the session key for integrity SK_I (Card):

  IAD-MAC = Leftmost 8 bytes of AES-CMAC+ (SK_I) [0000 || Input Data]

The EDA-MAC authenticates the Application Cryptogram and the IAD-MAC to the Reader. If the IAD-MAC is included in the AC computation it must be transmitted to the Issuer.

The 8-byte EDA-MAC is computed by the Card as follows:

- Concatenate '0000' with the Application Cryptogram and the IAD-MAC (without tags and lengths).
- Compute EDA-MAC by applying AES-CMAC over the concatenated data with the session key for integrity SK_I (Card):

  EDA-MAC = Leftmost 8 bytes of AES-CMAC (SK_I) ['0000' || AC || IAD-MAC]

To validate EDA-MAC the Reader computes IAD-MAC and EDA-MAC with its session key for integrity SK_I (Reader) and compares the EDA-MAC with the one received from the Card.

4 Remote Authentication

This section describes the remote authentication of the Card by the Issuer when performed during a transaction, it authenticates the transaction data and ensures that the Card is genuine.

With remote authentication the Issuer validates the Application Cryptogram and by so doing authenticates the Card and a subset of transaction data that was included in the Application Cryptogram computation. If this subset includes the IAD-MAC computed in section 3.2 then the Issuer also authenticates the local transaction data included in the IAD-MAC.

4.1 Master Key Management

This section describes three recommended methods to derive a Card Master Key MK from an Issuer Master Key IMK and the Application PAN.

A. When using 3DES with a PAN of any length, MK is computed as follows:

Concatenate the PAN and PAN Sequence Number (PSN) to form X:

X = PAN || PSN (if not present, PSN is replaced by a zero-byte)

Pad X to the left with zero-digits to form a 16-digit number Y, if X is less than 16 digits long:

Y = Rightmost 16 digits of X

Calculate the 128-bit key MK using the 128-bit key IMK as follows:

Let Y* be Y XORed with 8 'FF' bytes

Z = 3DES (IMK) [Y] || 3DES (IMK) [Y*]

MK = Z', where Z' is Z but with the least significant bit of each byte of Z set to a value that ensures that each of the 16 bytes of MK has an odd number of non-zero bits.

B. When using 3DES with a PAN equal to 16 decimal digits or less apply method A; with a PAN greater than 16 decimal digits, MK is computed as follows:

If the PAN has an odd number of digits, then pad the PAN to the left with a zero digit to have an even number of digits

Concatenate the PAN (padded) and PAN Sequence Number (PSN) to form X:

X = PAN (padded) || PSN (if not present, PSN is replaced by a zero byte)

H = SHA-1 (X) gives a 20-byte hash result

If H does not contain 16 decimal digits, add the required number of non-decimal nibbles of H modulo 10 starting from the left side of H to obtain 16 decimal digits:

Y = Leftmost 16 digits of {H decimal digits || H non-decimal nibbles mod 10}

Calculate the 128-bit key MK using the 128-bit key IMK as follows:

Let Y* be Y XORed with 8 'FF' bytes

Z = 3DES (IMK) [Y] || 3DES (IMK) [Y*]

MK = Z', where Z' is Z but with the least significant bit of each byte of Z set to a value that ensures that each of the 16 bytes of MK has an odd number of non-zero bits.

C. When using AES, MK is computed as follows:

Concatenate the PAN and PAN Sequence Number (PSN) to form X:

X = PAN || PSN (if not present, PSN is replaced by a zero-byte)

Pad X to the left with zero-digits to form a 16-byte number Y

When using 128-bit keys, calculate MK as follows:

MK = AES (IMK) [Y]

When using 192-bit or 256-bit keys, calculate MK as follows:

Let Y* be Y XORed with 16 'FF' bytes

MK = Leftmost 24 or 32 bytes of AES (IMK) [Y] || AES (IMK) [Y*]

At least one Card Master Key is used for the following purpose:

- MK_{AC} for the Application Cryptogram, derived from Issuer Master Key IMK_{AC}

Additional Card Master Keys may be derived for other purposes. They are derived from the corresponding Issuer Master Keys.

3DES master keys are 128 bits long, i.e. two 64-bit keys.

AES master keys are either 128, 192, or 256 bits long. The method to derive the corresponding 128-bit, 192-bit, or 256-bit session keys from the master key is described below when it applies.

The session keys are derived by the Card application except for implementations where they are derived by the Issuer host (e.g. in the cloud).

4.2 Application Cryptogram

The methods to generate the Application Cryptogram (AC) are defined by each Payment System. However, this section describes the recommended methods to generate an Application Cryptogram (the same methods are also defined in [EMV Book 2]). AC is an 8-byte MAC generated by the Card over a subset of the transaction data in a way it can be verified by the Issuer to validate the transaction.

AC is generated by applying DES-RMAC or AES-CMAC over the transaction data listed in Table 4.1 using the AC session key SK_{AC}. SK_{AC} is derived from the AC master key MK_{AC} and the Application Transaction Counter (ATC).

Table 4.1 – Recommended AC Input Data

Reference	Length	Origin
Amount, Authorised	6	Reader
Amount, Other	6	Reader
Terminal Country Code	2	Reader
Terminal Verification Results	5	Reader
Transaction Currency Code	2	Reader
Transaction Date	3	Reader
Transaction Type	1	Reader
Unpredictable Number	4	Reader
Application Interchange Profile	2	Card
Application Transaction Counter	2	Card
Card Verification Results	var.	Card
IAD-MAC computed in section 3.2 (1)	8	Card

(1) If IAD-MAC is provided to the Issuer

When the IAD-MAC is included in the AC computation and copied by the Reader in the IAD (or another data object transmitted to the Issuer), it contributes to the remote authentication of the transaction data by the Issuer.

The freshness of the Application Cryptogram is ensured by the Application Transaction Counter in the Card and the Unpredictable Number from the Reader.

- When using DES-RMAC, AC is computed as follows:

  Let R be ATC followed by 6 zero-bytes

  Calculate the 128-bit key SK_{AC} as follows:

  $R = R_0 \,||\, R_1 \,||\, R_2 \,||\, R_3 \,||\, R_4 \,||\, R_5 \,||\, R_6 \,||\, R_7$

  $SK_{ACL} = 3DES\ (MK_{AC})\ [R_0 \,||\, R_1 \,||\, 'F0' \,||\, R_3 \,||\, R_4 \,||\, R_5 \,||\, R_6 \,||\, R_7]$

  $SK_{ACR} = 3DES\ (MK_{AC})\ [R_0 \,||\, R_1 \,||\, '0F' \,||\, R_3 \,||\, R_4 \,||\, R_5 \,||\, R_6 \,||\, R_7]$

  $SK_{AC} = SK_{ACL} \,||\, SK_{ACR}$

  Calculate the 8-byte number AC as follows:

  $AC = DES\text{-}RMAC\ (SK_{AC})\ [AC\ Input\ Data]$

- When using AES-CMAC, AC is computed as follows:

  Let R be ATC followed by 14 zero-bytes

When using 128-bit keys, calculate SK_{AC} as follows:

$SK_{AC} = AES (MK_{AC}) [R]$

When using 192-bit or 256-bit keys, calculate SK_{AC} as follows:

$R = R_0 || R_1 || R_2 || R_3 || R_4 || R_5 || \ldots || R_{15}$

$SK_{ACL} = AES (MK_{AC}) [R_0 || R_1 || 'F0' || R_3 || R_4 || R_5 || \ldots || R_{15}]$

$SK_{ACR} = AES (MK_{AC}) [R_0 || R_1 || '0F' || R_3 || R_4 || R_5 || \ldots || R_{15}]$

SK_{AC} = Leftmost 24 or 32 bytes of $SK_{ACL} || SK_{ACR}$

Calculate the 8-byte number AC as follows:

AC = Leftmost 8-bytes of AES-CMAC (SK_{AC}) [AC Input Data]

The Issuer validates the authorisation request by computing the Application Cryptogram and comparing it with the one received from the Card.

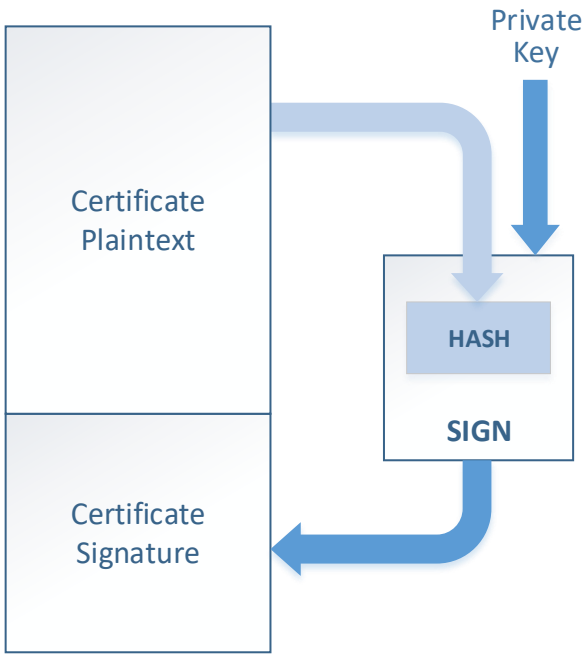
5 ECC Certificates

This section describes the security mechanisms to generate ECC certificates and authenticate ECC public keys. The RSA certificate option is described in section 6.

An ECC certificate consists of the certificate plaintext and the certificate signature.

In an ECSDSA signature the certificate plaintext of an arbitrary length is hashed, then the resulting hash is signed with the signer's private key. The length of the certificate signature (N_{SIG}) is equal to the length of the hash value (N_{HASH}) plus the length of an element of the curve field (N_{FIELD}).

Figure 5.1 – ECC Certificate Signature



The ECSDSA signature is verified with the signer's public key. The entire certificate plaintext is required to verify the hash value and thereby the certificate signature.

5.1 Issuer ECC Public Key Certificate

The Issuer ECC Public Key Certificate has the following format:

Table 5.1 – Issuer ECC Public Key Certificate

	Name	Description	Length
1	Issuer Certificate Format	Always Hex value '12' for this version of the specification.	1

	Name	Description	Length
2	Issuer Certificate Encoding	Always Hex value '00' for this version of the specification.	1
3	Issuer Identifier	Leftmost 3-10 digits from the PAN padded to the right with hex 'F's.	5
4	Issuer Public Key Algorithm Suite Indicator	Identifies the algorithm suite to be used with the Issuer Public Key when verifying Issuer signatures as defined in Table 8.3. Always Hex value '10' for this version of the specification.	1
5	Issuer Certificate Expiration Date	YYYYMMDD (UTC) after which this certificate is invalid.	4
6	Issuer Certificate Serial Number	Number unique to the Certification Authority that signs the Issuer certificate.	3
7	RID	Identifies the Payment System to which the Issuer Public Key is associated.	5
8	Certification Authority Public Key Index	In conjunction with the RID, identifies which Certification Authority Public Key and associated algorithms to use when verifying the Issuer certificate.	1
9	Issuer Public Key	Representation of Issuer Public Key (*x*-coordinate of Issuer Public Key point) on the curve identified by Issuer Public Key Algorithm Suite Indicator.	N_{FIELD}
10	Issuer Public Key Certificate Signature	A digital signature on items 1 to 9 in this table. Verified using the Certification Authority Public Key and associated algorithms identified by the Certification Authority Public Key Index (in this table).	N_{SIG}

The Issuer Public Key Certificate Signature is the result of signing the Issuer certificate plaintext (items 1 to 9 in Table 5.1) with the Certification Authority Private Key as described in section 8.8.7. When using the P-256 curve and the hash algorithm SHA-256, the length of the Issuer certificate is 117 bytes.

5.2 Issuer ECC Public Key Validation

The Reader first retrieves the Certification Authority Public Key (*x*, *y*) coordinates from the RID and the Certification Authority Public Key Index obtained from the Card, the Issuer Public Key validation is aborted if the Reader does not have the associated key.

If the Certification Authority Public Key is present the Reader validates the Issuer Public Key as follows:

1. Check whether the length of the Issuer certificate is at least 21 bytes.
2. Check whether the Issuer Certificate Format is '12'.
3. Check whether the Issuer Certificate Encoding is '00'.
4. Check whether the Issuer Identifier matches the leftmost 3-10 digits from the Application PAN obtained from the Card.
5. Check whether the Issuer Public Key Algorithm Suite Indicator is '10'.
6. Check whether the Certificate Expiration Date is equal to or later than the current date.
7. Check whether the RID matches the RID in the first 5 bytes of AID obtained from the Card (DF Name).
8. Check whether the Certification Authority Public Key Index obtained from the Card is the same as the one in Table 5.1.
9. Check whether the concatenation of the RID, Certification Authority Public Key Index, and Issuer Certificate Serial Number is not present on the Certification Revocation List that lists the Issuer Public Key Certificates revoked by Payment Systems.
10. Check whether the length of the Issuer certificate is equal to $N_{FIELD} + N_{SIG} + 21$ bytes.
11. Check whether the Issuer Public Key Certificate Signature is valid using the Certification Authority Public Key as described in section 8.8.7.

If any step above fails, the validation is aborted.

If all the steps were successful, the Issuer ECC Public Key is considered as genuine.

The Reader recovers the *y*-coordinate of the Issuer ECC Public Key from the *x*-coordinate in the Issuer certificate using the function described in section 8.8.4.

5.3 ICC ECC Public Key Certificate

The ICC (or Card) ECC Public Key Certificate has the following format:

Table 5.2 – ICC ECC Public Key Certificate

	Name	Description	Length
1	ICC Certificate Format	Always Hex value '14' for this version of the specification.	1
2	ICC Certificate Encoding	Always Hex value '00' for this version of the specification.	1
3	ICC Public Key Algorithm Suite Indicator	Identifies the algorithm suite to be used with the certified ICC Public Key when establishing the secure channel as defined in Table 8.4. Always Hex value '00' for this version of the specification.	1
4	ICC Certificate Expiration Date	YYYYMMDD (UTC) after which this certificate is invalid.	4
5	ICC Certificate Expiration Time	HHMM (UTC) after which this certificate is invalid.	2
6	ICC Certificate Serial Number	Number unique to the Issuer that signs the Card certificate.	6
7	ICCD Hash Encoding	Always Hex value '01' for this version of the specification, identifying TLV encoding of the input (except for the AIP where just the value is included) is used when computing the ICCD Hash.	1
8	ICCD Hash Algorithm Indicator	Identifies the hash algorithm used to compute the ICCD Hash. Hex value '02' identifying that SHA-256 is used.	1
9	ICCD Hash	Hash over the Static Data to be Authenticated using the hash algorithm identified by the ICCD Hash Algorithm Indicator.	N_{HASH}
10	ICC Public Key	Representation of ICC Public Key (*x*-coordinate of ICC Public Key point) on the curve identified by the ICC Public Key Algorithm Suite Indicator.	N_{FIELD}

	Name	Description	Length
11	ICC Public Key Certificate Signature	A digital signature on items 1 to 10 in this table. Verified using the Issuer Public Key and associated algorithms identified by the Issuer Public Key Algorithm Suite Indicator (in the Issuer certificate).	N_{SIG}

The ICC Public Key Certificate Signature is the result of signing the ICC certificate plaintext (items 1 to 10 in Table 5.2) with the Issuer ECC private key as described in section 8.8.7. When using the P-256 curve and the hash algorithm SHA-256, the length of the Card certificate is 145 bytes.

The ICCD Hash is a hash calculated over the Static Data to be Authenticated formed from the signed records identified by the AFL, followed by any data identified by the Extended SDA Tag List (including the tags and length) and the AIP value.

Note that the ICC certificate does not contain the PAN which is added to a signed record in order to be authenticated by the ICC Public Key Certificate Signature.

5.4 ICC ECC Public Key Validation

The Reader validates the ICC (or Card) Public Key with the Issuer Public Key validated in section 5.2 and the SDA Hash computed by the Kernel as follows:

1. Check whether the length of the ICC certificate described in Table 5.2 length is at least 17 bytes.
2. Check whether the ICC Certificate Format is '14'.
3. Check whether the ICC Certificate Encoding is '00'.
4. Check whether the ICC Certificate Expiration Date and ICC Certificate Expiration Time is equal to or later than the current date and time.
5. Check whether the ICC Public Key Algorithm Suite Indicator is '00'.
6. Check whether the ICCD Hash Encoding is '01'.
7. Check whether the ICCD Hash Algorithm Indicator is '02'.
8. Check whether the length of the ICC certificate described in Table 5.2 length is equal to $N_{HASH} + N_{FIELD} + N_{SIG} + 17$ bytes.
9. Check whether the SDA Hash already computed by the Kernel is the same as the ICCD Hash recovered from the ICC certificate.
10. Check whether the ICC Public Key Certificate Signature is valid using the Issuer Public Key as described in section 8.8.7.

If any step above fails, the validation is aborted.

If all the steps were successful, the ICC ECC Public Key is considered as genuine.

The Reader recovers the *y*-coordinate of the ICC ECC Public Key from the *x*-coordinate in the ICC certificate using the function described in section 8.8.4.

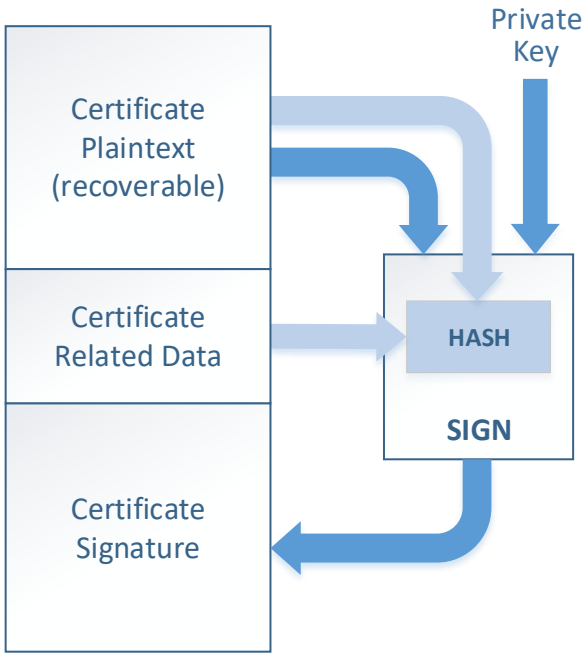
6 RSA Certificates

This section describes the security mechanisms to generate RSA certificates and authenticate RSA public keys.

In this specification the term RSA certificate consists of the recoverable certificate plaintext, certificate related data and certificate signature. In [EMV Book 2] the term “certificate” corresponds to the certificate signature only.

In an RSA signature, the length of the certificate signature is equal to the length of the public key modulus of the signer. In order to sign a certificate of an arbitrary length, the certificate plaintext and certificate related data are hashed, then the certificate plaintext and the resulting hash are signed with the signer’s private key.

Figure 6.1 – RSA Certificate Signature



The certificate signature is verified with the signer’s public key so that the certificate plaintext and the hash value are recovered from the certificate signature. The certificate related data is required to verify the hash value and thereby the certificate signature.

6.1 Issuer RSA Public Key Certificate

The Issuer RSA Public Key Certificate has the following format:

Table 6.1 – Issuer RSA Public Key Certificate

	Name	Description	Length
1	Issuer Certificate Format	Hex value '02'.	1
2	Issuer Identifier	Leftmost 3-8 digits from the PAN padded to the right with hex 'F's.	4
3	Issuer Certificate Expiration Date	MMYY after which this certificate is invalid.	2
4	Issuer Certificate Serial Number	Binary number unique to this certificate assigned by the Certification Authority.	3
5	Issuer Hash Algorithm Indicator	Identifies the hash algorithm used to produce the hash value in the digital signature scheme. Hex value '01' identifying that SHA-1 is used.	1
6	Issuer Public Key Algorithm Indicator	Identifies the digital signature algorithm to be used with the Issuer Public Key. Hex value '01' identifying that RSA is used.	1
7	Issuer Public Key Length	Identifies the length of the Issuer Public Key modulus in bytes.	1
8	Issuer Public Key Exponent Length	Identifies the length of the Issuer Public Key exponent in bytes.	1
9a	Issuer Public Key Leftmost Digits	If $N_I \leq N_{CA} - 36$, consists of the full Issuer Public Key padded to the right with $N_{CA} - 36 - N_I$ bytes of value 'BB'. If $N_I > N_{CA} - 36$, consists of the $N_{CA} - 36$ most significant bytes of the Issuer Public Key.	$N_{CA} - 36$

	Name	Description	Length
10	Issuer Certificate Signature	A digital signature on items 1 to 9a in this table and certificate related data below. Verified using the Certification Authority Public Key and associated algorithms identified by the RID and the Certification Authority Public Key Index (obtained from the Card).	N_{CA}

Issuer certificate related data also authenticated by the Issuer RSA Public Key Certificate:

Table 6.2 – Issuer RSA Certificate Related Data

	Name	Description	Length
9b	Issuer Public Key Remainder	Present only if $N_I > N_{CA} – 36$ and consists of the $N_I – N_{CA} + 36$ least significant bytes of the Issuer Public Key.	0 or $N_I – N_{CA} + 36$
9c	Issuer Public Key Exponent	Issuer Public Key exponent equal to 3 or 65537.	1 or 3

N_{CA} and N_I are the lengths in bytes of the Certification Authority and Issuer Public Key moduli respectively.

The Issuer Certificate Signature is the result of signing the concatenation of the Issuer certificate plaintext (items 1 to 9a in Table 6.1) and Issuer certificate related data (items 9b and 9c in Table 6.2) with the Certification Authority Private Key as described in section 8.7.2. Thus, the length of the Issuer Certificate Signature is equal to N_{CA}.

6.2 Issuer RSA Public Key Validation

The Reader first retrieves the Certification Authority Public Key from the RID and the Certification Authority Public Key Index obtained from the Card, the Issuer Public Key validation is aborted if the Reader does not have the associated key.

If the Certification Authority Public Key is present the Reader validates the Issuer Public Key as follows:

1. Check whether the length of the Issuer Certificate Signature is equal to the length of the Certification Authority Public Key modulus N_{CA}.

2. Recover the Issuer certificate plaintext from the Issuer Certificate Signature as described in section 8.7.2. This includes checking that the hash computed over the concatenation of Table 6.3 – Recovered Issuer Certificate Plaintext and Table 6.2 – Issuer RSA Certificate Related Data obtained from the Card is the same as the hash value recovered from the Issuer Certificate Signature.

Table 6.3 – Recovered Issuer Certificate Plaintext

Reference	Length	Value
Issuer Certificate Format	1	'02'
Issuer Identifier	4	Leftmost 3-8 digits from PAN
Issuer Certificate Expiration Date	2	MMYY
Issuer Certificate Serial Number	3	–
Issuer Hash Algorithm Indicator	1	'01'
Issuer Public Key Algorithm Indicator	1	'01'
Issuer Public Key Length	1	N_I
Issuer Public Key Exponent Length	1	1 or 3
Issuer Public Key Leftmost Digits (padded as described in Table 6.1)	N_{CA} – 36	–

3. Check whether the concatenation of the RID and Certification Authority Public Key Index obtained from the Card and the Issuer Certificate Serial Number is not present on the Certification Revocation List that lists the Issuer Public Key Certificates revoked by Payment Systems.
4. Check whether the Issuer Certificate Format is '02'.
5. Check whether the Issuer Identifier matches the leftmost 3-8 digits from the Application PAN obtained from the Card.
6. Check whether the last day of the month of the Issuer Certificate Expiration Date is equal or later than the current date.
7. Check whether the Issuer Hash Algorithm Indicator is '01'.
8. Check whether the Issuer Public Key Algorithm Indicator is '01'.

If any step above fails, the validation is aborted.

If all the steps were successful, the Issuer RSA Public Key is considered as genuine.

The Reader concatenates the Issuer Public Key Leftmost Digits (without padding) and Issuer Public Key Remainder (if present) to obtain the Issuer Public Key Modulus. The Reader verifies that the resulting length of the Issuer Public Key Modulus is equal to the recovered value of N_I and that $N_I \leq N_{CA}$; if not, validation is aborted.

6.3 ICC RSA Public Key Certificate

The ICC (or Card) RSA Public Key Certificate has the following format:

Table 6.4 – ICC RSA Public Key Certificate

	Name	Description	Length
1	ICC Certificate Format	Hex value '04'.	1
2	Application PAN	PAN padded to the right with Hex 'F's.	10
3	ICC Certificate Expiration Date	MMYY after which this certificate is invalid.	2
4	ICC Certificate Serial Number	Binary number unique to this certificate assigned by the Issuer.	3
5	ICC Hash Algorithm Indicator	Identifies the hash algorithm used to produce the hash value in the digital signature scheme. Hex value '01' identifying that SHA-1 is used.	1
6	ICC Public Key Algorithm Indicator	Identifies the digital signature algorithm to be used with the ICC Public Key. Hex value '01' identifying that RSA is used.	1
7	ICC Public Key Length	Identifies the length of the ICC Public Key modulus in bytes.	1
8	ICC Public Key Exponent Length	Identifies the length of the ICC Public Key exponent in bytes.	1
9a	ICC Public Key Leftmost Digits	If $N_C \leq N_I - 42$, consists of the full ICC Public Key padded to the right with $N_I - 42 - N_C$ bytes of value 'BB'. If $N_C > N_I - 42$, consists of the $N_I - 42$ most significant bytes of the ICC Public Key.	$N_I - 42$

10	ICC Certificate Signature	A digital signature on items 1 to 9a in this table and certificate related data below. Verified using the Issuer Public Key and associated algorithms identified by the Issuer Public Key Algorithm Indicator (in the Issuer Public Key Certificate).	N_I

ICC certificate related data also authenticated by the ICC RSA Public Key Certificate:

Table 6.5 – ICC RSA Certificate Related Data

	Name	Description	Length
9b	ICC Public Key Remainder	Present only if $N_C > N_I - 42$ and consists of the $N_C - N_I + 42$ least significant bytes of the ICC Public Key.	0 or $N_C - N_I + 42$
9c	ICC Public Key Exponent	ICC Public Key exponent equal to 3 or 65537.	1 or 3
9d	SDA	Static Data to be Authenticated formed from the signed records identified by the AFL, any data identified by the Extended SDA Tag List, and the AIP value.	–

N_I and N_C are the lengths in bytes of the Issuer and ICC Public Key moduli respectively.

The ICC Certificate Signature is the result of signing the concatenation of the ICC certificate plaintext (items 1 to 9a in Table 6.4) and ICC certificate related data (items 9b to 9d in Table 6.5) with the Issuer RSA Private Key as described in section 8.7.2. Thus, the length of the ICC Certificate Signature is equal to N_I.

The ICC RSA Public Key Certificate is used to authenticate the ICC ECC Public Key. The ICC ECC Public Key (*x*-coordinate) is added to a record identified in the AFL as signed i.e. included in the Static Data to be Authenticated.

Additional data objects that are not in records can be authenticated by adding the optional Extended SDA Tag List in a (preferably signed) record identified by the AFL.

The Static Data to be Authenticated is then formed from the signed records identified by the AFL, followed by any data identified by the Extended SDA Tag List (including the tags and length) and the AIP value.

For non-C-8 EMV contactless Kernels the Static Data to be Authenticated is formed from the signed records identified by the AFL, followed by the AIP value if identified by SDA Tag List. If present, the SDA Tag List only contains the tag '82' identifying the AIP.

6.4 ICC RSA Public Key Validation

The Reader validates the ICC (or Card) Public Key with the Issuer Public Key validated in section 6.2 as follows:

1. Check whether the length of the ICC Certificate Signature is equal to the length of the Issuer Public Key modulus N_I.
2. Recover the ICC certificate plaintext from the ICC Certificate Signature as described in section 8.7.2. This includes checking that the hash computed over the concatenation of Table 6.6 – Recovered ICC Certificate Plaintext and Table 6.5 – ICC RSA Certificate Related Data obtained from the Card is the same as the hash value recovered from the ICC Certificate Signature.
3. *For non-C-8 EMV contactless Kernels, if the SDA Tag List is present in the Static Data to be Authenticated (item 9d of Table 6.5), check that the SDA Tag List only contains the tag '82'.*

Table 6.6 – Recovered ICC Certificate Plaintext

Reference	Length	Value
ICC Certificate Format	1	'04'
Application PAN	10	–
ICC Certificate Expiration Date	2	MMYY
ICC Certificate Serial Number	3	–
ICC Hash Algorithm Indicator	1	'01'
ICC Public Key Algorithm Indicator	1	'01'
ICC Public Key Length	1	N_C
ICC Public Key Exponent Length	1	1 or 3
ICC Public Key Leftmost Digits (padded as described in Table 6.4)	N_I – 42	–

4. Check whether the ICC Certificate Format is '04'.
5. Check whether the ICC Hash Algorithm Indicator is '01'.
6. Check whether the ICC Public Key Algorithm Indicator is '01'.
7. Check whether the Application PAN recovered from the ICC Public Key Certificate matches the PAN obtained from the Card.
8. Check whether the last day of the month of the ICC Certificate Expiration Date is equal to or later than current date.

If any step above fails, the validation is aborted.

If all the steps were successful, both the ICC RSA Public Key and the ICC ECC Public Key (*x*-coordinate) contained in the Static Data to be Authenticated are considered as genuine. The Reader discards the ICC RSA Public Key and recovers the *y*-coordinate of the ICC ECC Public Key from the *x*-coordinate using the function described in section 8.8.4.

For non-C-8 EMV contactless Kernels, the Reader concatenates the ICC Public Key Leftmost Digits (without padding) and ICC Public Key Remainder (if present) to obtain the ICC Public Key Modulus. The Reader verifies that the resulting length of the ICC Public Key Modulus is equal to the recovered value of N_C and that $N_C \leq N_I$; if not, validation is aborted.

7 BDH Primitives

In the following N_{FIELD} is the length of the ECC public key (size of the field element).

7.1 BDH Initialisation

The BDH initialisation is the process of computing and exchanging the Kernel public key and the Card blinded public key between the Reader and the Card as follows:

- The Card being personalised with the ECC key pair (d_C, Q_C) where $Q_C = d_C \cdot G$
- The Reader generates a Kernel ECC key pair (d_K, Q_K) on the curve as follows:
  - Generate a random integer d_K (private key) where $1 < d_K < n\text{-}1$
  - Calculate the Kernel public key $Q_K = d_K \cdot G$
- The Reader sends Q_K (x, y) coordinates to the Card in the GET PROCESSING OPTIONS command as a pair of byte strings (X, Y) of N_{FIELD} bytes each
- The Card verifies that Q_K is a valid point on the curve as follows:
  - Consider (X, Y) as a pair of integers (x, y)
  - Check that the point is valid with the function described in section 8.8.3
- The Card generates the blinding factor r
  - Where r is a random integer such as $1 < r < n\text{-}1$
  - If the implementation supports it, r may be simply generated as a random number of N_{FIELD} bytes
- The Card generates the Card blinded public key P_C with the blinding factor r as described in section 8.8.6
- The Card sends the x-coordinate of P_C to the Reader in the GET PROCESSING OPTIONS response as a byte string of N_{FIELD} bytes

7.2 BDH Key Derivation

The Reader and the Card derive two 128-bit session keys according to [ISO/IEC 11770-6] as follows:

- The Card calculates the shared secret from the Card private key, the blinding factor and the Kernel public key as being the x-coordinate of:
  - $z = d_C \cdot r \cdot Q_K = d_C \cdot r \cdot d_K \cdot G$
- The Reader recovers the y-coordinate of the Card blinded public key using the function described in section 8.8.4. If an error is detected by the function, the Reader terminates the transaction.

- The Reader calculates the shared secret from the Kernel private key and the Card blinded public key as being the *x*-coordinate of:
  - $z' = d_K \cdot P_C = d_K \cdot r \cdot Q_C = d_K \cdot r \cdot d_C \cdot G = z$
- The Card and the Reader calculate the 16-byte derivation key K_D using AES-CMAC described in section 8.6.4 as follows:
  - Consider *z* (*x*-coordinate) as a byte string Z of N_{FIELD} bytes
  - Let V be 16 zero bytes
  - K_D = AES-CMAC (V) [Z]
- Derive the session key for confidentiality SK_C as follows:

  SK_C = AES (K_D) ['01' || '01' || '00' || '54334A325957773D' || 'A5A5A5' || '0180']
- Derive the session key for integrity SK_I as follows:

  SK_I = AES (K_D) ['02' || '01' || '00' || '54334A325957773D' || 'A5A5A5' || '0180']

7.3 BDH Blinding Factor Validation

The Reader validates the blinding factor with the Card public key Q_C and the Card blinded public key P_C as follows:

- Consider SK_C and SK_C' as the confidentiality session keys derived by the Card and the Reader respectively
- The Card encrypts the blinding factor *r* as follows:
  - Consider the blinding factor *r* as a byte string R of N_{FIELD} bytes
  - Encrypt R as E(R) using AES-CTR with the message counter CMC and session key for confidentiality SK_C
- The Card sends E(R) to the Reader in the GET PROCESSING OPTIONS response as a byte string of N_{FIELD} bytes
- The Reader decrypts the blinding factor *r'* as follows:
  - Decrypt E(R) as R' using AES-CTR with counter CMC and SK_C'
  - Consider R' as an integer *r'* mod *n* where $0 < r' < n$
- The Reader regenerates the Card blinded public key P_C' with the Card public key Q_C and the blinding factor *r'* as described in section 8.8.6, where Q_C is recovered from the Card public key certificate or Static Data to be Authenticated
  - If P_C and P_C' *x*-coordinates are equal, then the blinding factor is valid

8 Cryptographic Algorithms

8.1 Random Numbers

For Kernel 8, the Relay Resistance Protocol, Application Cryptogram computation, and Blinded Diffie-Hellman key agreement require the Reader to generate at least two random numbers on every transaction, one used for the RRP entropy and the Unpredictable Number, and one used for the Kernel ECC private key. The random numbers should be generated by a hardware random number generator (RNG) compliant with [ISO/IEC 18031] and tested using [NIST SP800-22A]. If a software RNG is used it must be seeded from an unpredictable source of data.

All random number values must be equally likely to occur, and the value of the random numbers must be unpredictable given knowledge of previous values. The generation process must be defended from “debug” type attacks that can uncover intermediate values that might enable outsiders to predict the output of the RNG. The RNG must be protected from external perturbation that might reduce its quality, for example, electromagnetic fields or glitches. It must not be possible for an attacker to deliberately cause a fallback from the hardware RNG to a software one.

[EMV SB144] introduced a specific method for generating Reader Unpredictable Numbers (UNs).

8.2 Bit String Interpretation

RSA and ECC operations in this specification (e.g. RSA and ECC signatures, BDH key agreement) require interpreting bit strings either as byte strings for logic operations or as non-negative integers for arithmetic operations. The RSA or ECC public keys for instance are interpreted as byte strings when integrated in the message to hash, or as an integer when used to verify the certificate signatures.

A bit string x of length $8k$ bits $<x_{8k-1}, \ldots, x_0>$ is interpreted as a byte string

$$X = X_{k-1} \,||\, X_{k-2} \,||\, \ldots \,||\, X_0 \text{ where } X_i = <x_{8i+7}, \ldots, x_{8i}>$$

or as a positive integer

$$x = \sum_{j=0}^{8k-1} x_j 2^j$$

When an integer is interpreted as a byte string, zero-bytes are appended to the left (most significant byte) to reach the number of bytes required.

The actual representation of a multi-precision integer is implementation dependent. On a 32-bit CPU for instance, a positive integer may be represented as follows:

$$x = \sum_{j=0}^{h-1} a_j (2^{32})^j \text{ where } a_j \in (0, 1, \ldots, 2^{32} - 1)$$

8.3 Hash Algorithms

Hash Algorithm Indicators are allocated by EMVCo. EMVCo has partitioned the set of values: those in the range '00' – '7F' are specified by EMV and subject to type approval, whereas those in the range '80' – 'FF' are for proprietary use (e.g. regional/domestic systems) – implementation specifications and testing are out of the scope of EMVCo.

Values indicated as “assigned” are for future proofing. Whilst EMV testing and type approval are not currently available, implementers should be aware that support for the indicated algorithms may be introduced in the future.

List of hash algorithms:

Table 8.1 – Hash Algorithms

Hash Algorithm Indicator	Hash Algorithm	Hash Length N_{HASH}	Block Size
'01'	SHA-1	20 bytes	64 bytes
'02'	SHA-256	32 bytes	64 bytes
'03' (assigned)	SHA-512	64 bytes	128 bytes

SHA-1, SHA-256, and SHA-512 are standardised in [ISO/IEC 10118-3].

SHA-1 is included for support of RSA certificates.

8.4 Algorithm Suite Indicators

Algorithm Suite Indicators are allocated by EMVCo. EMVCo has partitioned the set of values: those in the range '00' – '7F' are specified by EMV and subject to type approval, whereas those in the range '80' – 'FF' are for proprietary use (e.g. regional/domestic systems) – implementation specifications and testing are out of the scope of EMVCo.

Values indicated as “assigned” are for future proofing. Whilst EMV testing and type approval are not currently available, implementers should be aware that support for the indicated algorithms and key lengths may be introduced in the future.

The RSA public key certificates are signed using the following algorithm suite:

Table 8.2 – RSA Signature Algorithm Suite

Certificate Algorithm Suite Indicator	Public Key Algorithm	Public Key Modulus Max Length	Public Key Exponent
'01'	RSA	248 bytes	3 or 65537

The ECC public key certificates are signed using the following algorithm suites:

Table 8.3 – ECC Signature Algorithm Suites

Certificate Algorithm Suite Indicator	Signature Scheme	Hash Algorithm	N_{HASH}	ECC Curve	N_{FIELD} / N_{SIG}
'10'	ECSDSA	SHA-256	32 bytes	P-256	32 / 64 bytes
'11' (assigned)	ECSDSA	SHA-512	64 bytes	P-521	66 / 130 bytes
'80'	SM2DSA	SM3	32 bytes	SM2-P256	32 / 64 bytes

Where N_{HASH} is the output length of the hash function, N_{FIELD} is the length of the ECC public key (size of the field element) and N_{SIG} is the length of the ECC digital signature.

The secure channel uses the following algorithm suites:

Table 8.4 – Secure Channel Algorithm Suites

Secure Channel Algorithm Suite Indicator	Key Agreement	ECC Curve	N_{FIELD}	Authentication Algorithm	Encryption Algorithm
'00'	BDH	P-256	32 bytes	AES-CMAC	AES-CTR
'88'	BDH	SM2-P256	32 bytes	SM4-CMAC	SM4-CTR

Where N_{FIELD} is the length of the ECC public key (size of the field element).

8.5 DES Cryptography

8.5.1 DES

DES and Triple DES (3DES) (see [ISO/IEC 18033-3]) are the approved cryptographic algorithms to be used in encryption and MAC mechanisms where an 8-byte block cipher is used.

3DES encryption is the process of encrypting an 8-byte plaintext block to form an 8-byte ciphertext block with a 128-bit double-length secret key $K = K_L \,||\, K_R$ as follows:

$$Y = 3DES\ (K)\ [X] = DES\ (K_L)\ [DES^{-1}\ (K_R)\ [DES\ (K_L)\ [X]]]$$

Decryption of the ciphertext takes place as follows:

$$X = 3DES^{-1}\ (K)\ [Y] = DES^{-1}\ (K_L)\ [DES\ (K_R)\ [DES^{-1}\ (K_L)\ [Y]]]$$

8.5.2 DES-RMAC

DES-RMAC (commonly known as the ANSI Retail MAC) is the approved algorithm based on the DES 8-byte block cipher to generate the Application Cryptogram.

The MAC over a message M consisting of an arbitrary number of bytes with a 128-bit double-length session key $K = K_L \,||\, K_R$ is computed as follows:

1. Pad the message according to Padding Method 2 of [ISO/IEC 9797-1].

   Add a mandatory byte with hexadecimal '80' to the right, and then add the smallest number of zero-bytes to the right (possibly none), so that the length of the resulting message M is a multiple of 8 bytes:

   M = M || '80' || '00' || '00' || ... || '00'

2. Split the message M, output of step 1, into 8-byte data blocks labelled $D_1, D_2, \ldots, D_n$.
3. Compute the MAC according to Algorithm 3 of [ISO/IEC 9797-1] as follows:
   - Calculate the intermediate results H_i for $i = 1, 2, \ldots, n$ with $H_0 = 0$

     $H_i = DES\ (K_L)\ [D_i \oplus H_{i-1}]$

   - Calculate the MAC as follows:

     $MAC = DES\ (K_L)\ [DES^{-1}\ (K_R)\ [H_n]]$

8.6 AES Cryptography

8.6.1 AES

AES (see [ISO/IEC 18033-3]) is the approved cryptographic algorithm to be used in the encryption and MAC mechanisms where a 16-byte block cipher is used.

The AES based mechanisms in this specification use a 128-bit, 192-bit, or 256-bit secret key as appropriate.

8.6.2 AES-CTR

The data encryption of records and data envelopes uses AES in counter mode (CTR) as described in [ISO/IEC 10116] without ciphertext expansion (padding). This function is considered insecure if the message is longer than 2^{112} blocks or 2^{116} bytes but in practice it is never used for more than 256 bytes in Kernel 8.

Considering a message counter MC, an input message P and the session key SK, the output message C defined as

C = AES-CTR (SK) [MC, P]

is obtained by encrypting the input message P as follows:

1. Let SV be the 2-byte MC followed by 14 zero-bytes
2. Split the input message P into 16-byte data blocks labelled $P_1, P_2, \ldots, P_n$ with the final data block P_n of z bytes ($0 < z \leq 16$)
3. Generate the encryption variables as follows:

E_i = AES (SK) [SV + (*i* - 1)] for *i* = 1, 2, …, *n*

4. Truncate the final encryption variable E_n to the leftmost *z* bytes
5. Encrypt the input message P with the encryption variables as follows:

   $C_i = P_i \oplus E_i$ for *i* = 1, 2, …, *n*

6. The output message C is the byte string $C_1 \| C_2 \| \ldots \| C_n$

The decryption of a ciphertext is obtained by applying the ciphertext as the input message in the encryption steps above.

8.6.3 AES-CBC

AES encryption in CBC mode (AES-CBC) is the process of encrypting a 16*n*-byte block X with a secret key K and a 16-byte initial value IV as follows:

1. Split the 16*n*-byte block X in *n* 16-byte data blocks labelled $X_1, X_2, \ldots, X_n$.
2. Encrypt each block X_i as follows:

   Y_1 = AES (K) [$X_1 \oplus$ IV]

   Y_i = AES (K) [$X_i \oplus Y_{i-1}$] for *i* = 2, …, *n*

3. Concatenate the *n* 16-byte results Y_i to form the 16*n*-byte value Y:

   $Y = Y_1 \| Y_2 \| \ldots \| Y_n$

The decryption of the Y value is the process of decrypting each block Y_i as follows:

$X_1 = AES^{-1}$ (K) [Y_1] $\oplus$ IV

$X_i = AES^{-1}$ (K) [Y_i] $\oplus$ Y_{i-1} for *i* = 2, …, *n*

8.6.4 AES-CMAC

AES-CMAC is the approved algorithm based on the AES 16-byte block cipher to generate the BDH key for derivation, the Application Cryptogram (optionally), the EDA-MAC, and the secure data storage MACs.

The 16-byte MAC over a message M consisting of an arbitrary number of bytes with a session key SK is computed as follows:

1. Pad the message M according to Padding Method 4 of [ISO/IEC 9797-1].

   If the length of the message M is a multiple of 16 bytes, then no padding is added.

   If the length of the message M is not a multiple of 16 bytes, add a mandatory byte with hexadecimal '80' to the right, and then add the smallest number of zero-bytes to the right, so that the length of the resulting message is a multiple of 16 bytes:

   M = M || '80' || '00' || '00' || ... || '00'

2. Split the message M, output of step 1, into 16-byte data blocks labelled $D_1, D_2, \ldots, D_n$.

3. Derive two 16-byte masking keys K_1 and K_2 from SK according to Key Derivation Method 2 of [ISO/IEC 9797-1] as follows:

   Let V be 16 zero-bytes and let C be 15 zero-bytes followed by '87'.

   $L = AES (SK) [V]$

   $K_1 = L << 1$, if msb (L) = 1 then $K_1 = K_1 \oplus C$

   $K_2 = K_1 << 1$, if msb (K_1) = 1 then $K_2 = K_2 \oplus C$

   Where the notation << 1 is left-shift one bit and msb is the most significant bit.

   Note that all intermediate values must be kept secret and not used for other purposes.

4. Mask the final data block D_n:

   $D_n = D_n \oplus K_1$ if no padding was added

   $D_n = D_n \oplus K_2$ if padding was added

5. Compute the MAC according to Algorithm 5 of [ISO/IEC 9797-1] as follows:

   - Calculate the intermediate results H_i for i = 1, 2, …, n with $H_0 = 0$

     $H_i = AES (SK) [D_i \oplus H_{i-1}]$

   - And MAC = H_n

8.6.5 AES-CMAC+

AES-CMAC+ is the approved algorithm based on the AES-CMAC to generate the IAD-MAC value.

The 16-byte MAC over a message M consisting of an arbitrary number of bytes with a session key SK is computed as described in section 8.6.4 with an extra operation on the last block H_n as follows:

$H_n = AES (SK) [J_n]$ with $J_n = D_n \oplus H_{n-1}$

$MAC = H_n \oplus J_n$

Or using AES-CBC:

$MAC = AES\text{-}CBC\text{-}Decrypt (SK) [H_n]$ with $IV = H_n$

8.7 RSA Cryptography

8.7.1 RSA Algorithm

The RSA algorithm is the reversible asymmetric cryptographic algorithm used in the digital signature scheme described in section 8.7.2.

The algorithm consists of a signing function (Sign) depending on a private key S_K and a recovery function (Recover) depending on a public key P_K. Both functions map N-byte numbers onto N-byte numbers (as defined in section 8.2), where N is the size of the public key modulus *n* used.

- $S = Sign\ (S_K)\ [X] = X^d \bmod n,\ 0 < X < n$
- $X = Recover\ (P_K)\ [S] = S^e \bmod n$

where X is the message to be signed and S the corresponding digital signature. The public key P_K is represented by the public key exponent *e* and the public key modulus *n*, and the private key S_K by the private exponent *d* (also in combination with *n*).

8.7.2 RSA Signature

The RSA signature is a digital signature scheme with message recovery using a hash algorithm according to [ISO/IEC 9796-2]. SHA-1 is the hash algorithm used to produce a 20-byte hash value over the message to be signed.

The following computations are based on the Sign and Recover functions described in section 8.7.1.

Let N be the length of the public key modulus. The signature S on a message M consisting of an arbitrary number L of at least N – 21 bytes with the signer's private key S_K is computed as follows:

- Compute the 20-byte hash value H = Hash [M]
- Split message $M = M_1 \| M_2$ where
  - M_1 = Leftmost (N – 22) bytes of M
  - M_2 = Remaining (L – N + 22) bytes of M
- Define the byte B = '6A'
- Define the byte E = 'BC'
- Define the N-byte number $X = B \| M_1 \| H \| E$
- Compute the N-byte signature $S = Sign\ (S_K)\ [X]$

The corresponding verification of the signature S with the signer's public key P_K, for the given value of M_2, is computed as follows:

- Check whether the signature S consists of N bytes

- Retrieve the N-byte number X = Recover (P_K) [S]
- Parse X = B || M_1 || H || E where
  - B and E are one byte long
  - M_1 is N – 22 bytes long
  - H is 20 bytes long
- Check whether B = '6A'
- Check whether E = 'BC'
- Recreate M = M_1 || M_2
- Check whether Hash [M] = H

If all these checks are correct the message M is considered as genuine.

8.8 ECC Cryptography

This section defines the P-256 and P-521 curves selected from [ISO/IEC 15946-5]. Other curves may be used for domestic use, but the details are outside the scope of this specification. P-521 is supported for contingency purposes only.

An elliptic curve is built on (or *defined over*) a field. In the scope of this specification, this is always the finite field F_p for a prime p. An ECC Public Key is a point on the curve represented in this specification by the x-coordinate of the point.

8.8.1 P-256 Curve

The P-256 curve is defined over F_p for a 256-bit prime p with the curve equation:
$y^2 = x^3 – 3x + b$ (i.e. parameter $a = –3$)

In Table 8.5 the prime p and (integer) order n are given in both decimal and hexadecimal form; field elements (a, b parameters, generator point G coordinates) are given in hexadecimal only.

Table 8.5 – P-256 Curve Parameters

Parameter	Value
p =	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$ = 115 79208 92103 56248 76269 74469 49407 57353 00861 43415 29031 41955 33631 30886 70978 53951 FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF
n =	115 79208 92103 56248 76269 74469 49407 57352 99969 55224 13576 03424 22259 06106 85120 44369 FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551
a =	FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFC
b =	5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6 3BCE3C3E 27D2604B
G_x =	6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C296
G_y =	4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5

Note that because the P-256 curve operates over finite fields with a prime number p of elements, each element of the field is naturally interpreted as a non-negative integer less than p and points are represented as pairs of such integers (x, y).

8.8.2 P-521 Curve

The P-521 curve is defined over F_p for a 521-bit Mersenne prime p with the curve equation: $y^2 = x^3 - 3x + b$ (i.e. parameter $a = -3$)

In Table 8.6 the prime p and (integer) order n are given in both decimal and hexadecimal form; field elements (a, b parameters, generator point G coordinates) are given in hexadecimal only.

Table 8.6 – P-521 Curve Parameters

Parameter	Value
p =	$2^{521} - 1$ = 68 64797 66013 06097 14981 90079 90813 93217 26943 53001 43305 40939 44634 59185 54318 33976 56052 12255 96406 61454 55497 72963 11391 48085 80371 21987 99971 66438 12574 02829 11150 57151 01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
n =	68 64797 66013 06097 14981 90079 90813 93217 26943 53001 43305 40939 44634 59185 54318 33976 55394 24505 77463 33217 19753 29639 96371 36332 11138 64768 61244 03803 40372 80889 27070 05449 01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFA 51868783 BF2F966B 7FCC0148 F709A5D0 3BB5C9B8 899C47AE BB6FB71E 91386409
a =	01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFC
b =	51 953EB961 8E1C9A1F 929A21A0 B68540EE A2DA725B 99B315F3 B8B48991 8EF109E1 56193951 EC7E937B 1652C0BD 3BB1BF07 3573DF88 3D2C34F1 EF451FD4 6B503F00
G_x =	C6 858E06B7 0404E9CD 9E3ECB66 2395B442 9C648139 053FB521 F828AF60 6B4D3DBA A14B5E77 EFE75928 FE1DC127 A2FFA8DE 3348B3C1 856A429B F97E7E31 C2E5BD66
G_y =	0118 39296A78 9A3BC004 5C8A5FB4 2C7D1BD9 98F54449 579B4468 17AFBD17 273E662C 97EE7299 5EF42640 C550B901 3FAD0761 353C7086 A272C240 88BE9476 9FD16650

Note that because the P-521 curve operates over finite fields with a prime number p of elements, each element of the field is naturally interpreted as a non-negative integer less than p and points are represented as pairs of such integers (x, y).

8.8.3 Point Verifying

To verify that the point with (x, y) coordinates is on the curve, the (x, y) coordinates satisfy:

$y^2 = x^3 + ax + b \bmod p$

8.8.4 Point Finding

In this specification, an elliptic curve public key is a point on the curve and is represented by the x-coordinate of the point, with the exception that the ephemeral Kernel public key is represented by the (x, y) coordinates.

To determine the *y*-coordinate of a point on the curve that has a given *x*-coordinate, it is necessary to find an integer *y* such that $y^2 = x^3 + ax + b$. Assuming the *x*-coordinate is interpreted as an integer and the field size $p = 3 \bmod 4$, the *y*-coordinate is computed as follows:

1. $y' = (x^3 + ax + b)^{((p+1)/4)} \bmod p$
2. $y = \min(y', p-y')$

Note that when the recovered point is to be used for Diffie-Hellman key agreement rather than signature verification either of the two possible *y* values may be used and therefore step 2 may be omitted and *y'* can be used directly.

Implementations check that the resultant point (*x*, *y*) is on the curve using the function described in section 8.8.3. Steps 1 and 2 above will always give a *y* value for any input *x* but it is possible *x* is invalid not giving a point on the curve and hence the need for the check.

8.8.5 Key Generation

The generation of elliptic curve key pair (*d*, *Q*) follows [ISO/IEC 15946-1]:

- The private key *d* for a given elliptic curve *E* with base point *G* of order *n* consists of a positive integer *d*, satisfying $1 < d < n-1$
- The corresponding public key point is $Q = d \cdot G$ on the curve *E*

Given an elliptic curve, all parameters including the base point are fixed. Therefore, key generation consists solely of random or pseudorandom generation of *d* and subsequent computation of the public key point $Q = d \cdot G$.

In certificates, this specification represents public keys using only their *x*-coordinate. For any non-zero point (*x*, *y*) on the curve, (*x*, -*y*) will also be on the curve. The function that is sensitive to the value of the *y*-coordinate is the certificate signature verification, specifically verification of the Issuer Public Key Certificate Signature using the Certification Authority Public Key and verification of the ICC Public Key Certificate Signature using the Issuer Public Key.

To ensure that the verifier can determine the correct value of the *y*-coordinate of the public key from knowledge of only the *x*-coordinate, the Certification Authority and Issuer keys are generated so that the *y*-coordinate, when represented as an integer modulo *p*, is less than $(p+1)/2$. The function described in section 8.8.4 will then generate the correct *y*-coordinate given the *x*-coordinate.

Two examples of how to generate such Certification Authority and Issuer keys are provided below. The first approach (1) will take longer, but the second approach (2) may require special HSM functionality:

(1) Repeatedly generate a random integer *d* satisfying $1 < d < n-1$ until the *y*-coordinate of $d \cdot G$ is less than $(p+1)/2$

(2) Generate a random integer d' satisfying $1 < d' < n-1$, and if the y-coordinate of $d' \cdot G$ is less than $(p+1)/2$ then $d = d'$ else set $d = n - d'$

The public key is then calculated to be the point $Q = d \cdot G$ and its representation is the x-coordinate of Q. Note that this construction of the Certification Authority and Issuer keys reduces the key space of the private key by one bit.

When generating the ICC keys and the Kernel ephemeral keys which are used in BDH key agreement it is not necessary to ensure that the y-coordinate is less than $(p+1)/2$ as only the x-coordinate is used.

For security reasons, a strong pseudo-random or random number generator is required as described in section 8.1.

8.8.6 Public Key Blinding

To anonymise the public key of an ECC key pair (d, Q) where $Q = d \cdot G$, it is necessary to calculate the blinded public key P using the blinding factor r as follows:

$$P = r \cdot Q = r \cdot d \cdot G$$

Note that $P = (r \cdot d) \cdot G$ is easier for the Card to compute as it does not require that the Card stores Q in a specific personalisation data object and the Card needs to calculate $(r \cdot d)$ during the BDH initialisation anyway.

8.8.7 ECSDSA Signature

The ECSDSA signature is an ECC version of the Schnorr digital signature scheme with appendix according to [ISO/IEC 14888-3]. This is the optimised version where only the x-coordinate is hashed rather than the (x, y) coordinates.

The ECSDSA algorithm relies on a signer's key pair (d, Q) generated on the curve, where the signer's public key $Q = d \cdot G$.

The generation of the signature (R, S) on a message M, consisting of an arbitrary length of bytes, with the signer's private key d is computed as follows:

- Generate a random integer k where $0 < k < n$
- Calculate the public key $k \cdot G = (x_1, y_1)$
- Consider x_1 as a byte string X_1 of N_{FIELD} bytes
- Calculate R = Hash $[X_1 \,||\, M]$ of N_{HASH} bytes
- Consider R as an integer r [with $0 < r$]
- Reduce r modulo n
- Calculate $s = (k + r \cdot d) \bmod n$ [with $0 < s$]
- Consider s as a byte string S of N_{FIELD} bytes
- Concatenate the signature (R, S) of $N_{HASH} + N_{FIELD}$ bytes

The verification of the signature (R, S) on a message M, consisting of an arbitrary length of bytes, with the signer's public key *Q* is computed as follows:

- Check that the length of (R, S) is of $N_{HASH} + N_{FIELD}$ bytes
- Consider R and S as integers *r* and *s* respectively
- Reduce *r* modulo *n*
- Check that $0 < s < n$ and $0 < r$
- Calculate $s \cdot G - r \cdot Q = (x_2, y_2)$
- Consider x_2 as a byte string X_2 of N_{FIELD} bytes
- Calculate R' = Hash [X_2 || M] of N_{HASH} bytes
- Check that R' = R which should be true because

  $$s \cdot G - r \cdot Q = (k + r \cdot d) \cdot G - r \cdot d \cdot G = k \cdot G$$

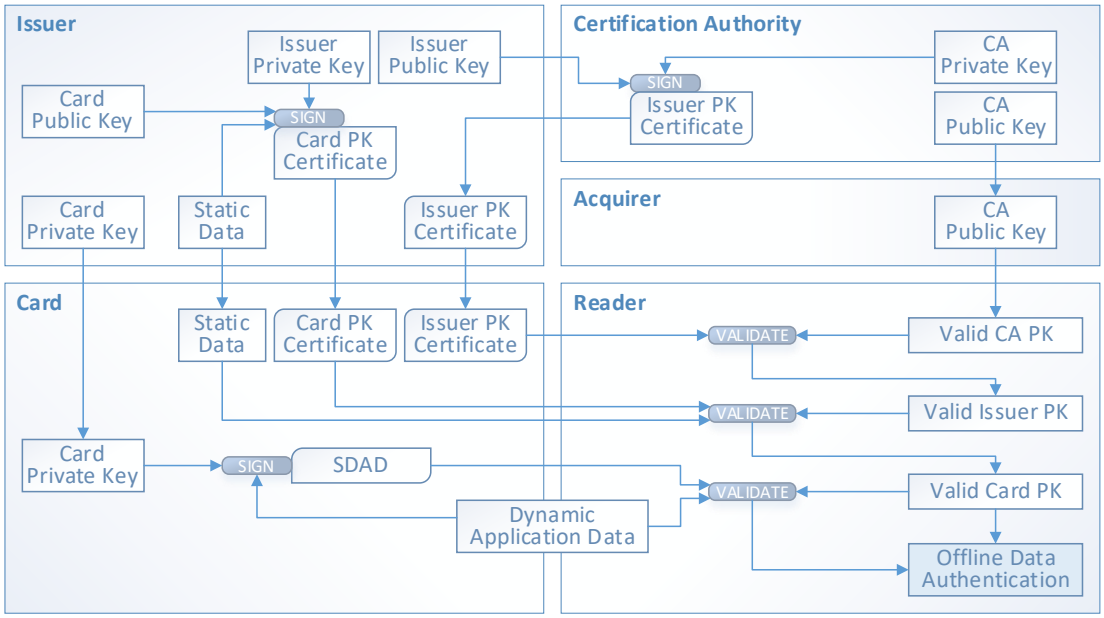
If all these checks are correct, then the signature (R, S) is considered to be a valid signature on message M with respect to the public key *Q* and the given curve.

Annex A Offline Data Authentication

A.1 Keys and Certificates

Figure A.1 describes the public key infrastructure (PKI) and offline data authentication (ODA) of contactless transactions (except Kernel 8). Every Card is personalised by the Issuer with a unique RSA private/public key pair where the Card public key is certified by the Issuer.

Figure A.1 – ODA Diagram



A three-layer authentication scheme allows the Reader to validate the signature of the dynamic application data generated by the Card:

- The Issuer public key is authenticated by the Certification Authority (CA) public key stored in the Reader.
- The Card public key is authenticated by the Issuer public key which is signed by the Certification Authority in the Issuer certificate.
- The dynamic application data is authenticated by the Card public key which is signed by the Issuer in the Card certificate.

In non-C-8 EMV contactless Kernels, the Signed Dynamic Application Data (SDAD) is the result of either of the two methods of RSA dynamic data signature generation:

- Dynamic Data Authentication (DDA) where Card and Reader dynamic application data is signed by the Card. This annex describes an accelerated variant of DDA known as the fast Dynamic Data Authentication (fDDA).

- Combined Dynamic Data Authentication/Application Cryptogram Generation (CDA), similar to DDA but including the Application Cryptogram and all other transaction data in the dynamic application data signed by the Card.

If the validation of SDAD and the Issuer and Card certificates is successful, the offline data authentication is successful and the Card is considered as genuine.

Issuer and Card certificates are personalised in the Card by the Issuer and transmitted by the Card to the Reader if ODA is required.

A.2 Dynamic Data Signature Generation

The generation of the ICC dynamic data signature is the result of signing the dynamic application data listed in Table A.1 with the Card private key to obtain the Signed Dynamic Application Data (SDAD), as described in section 8.7.2.

The RSA signature algorithm first computes the hash over the concatenation of the dynamic application data using the hash algorithm derived from the Hash Algorithm Indicator (SHA-1). The items originated from the Card (N_C – 22 bytes long) concatenated with the resulting hash are then input to the RSA Sign function. The length of SDAD has the length of the Card public key modulus N_C.

Table A.1 – Dynamic Application Data

Reference	Length	Origin
Signed Data Format ('05')	1	Card
Hash Algorithm Indicator ('01')	1	Card
ICC Dynamic Data Length (L_{DD})	1	Card
ICC Dynamic Data	L_{DD}	Card
Padding Bytes ('BB')	$N_C – L_{DD} – 25$	Card
Terminal Dynamic Data	var.	Reader

The length of the ICC Dynamic Data generated by or stored in the Card should satisfy $0 \leq L_{DD} \leq N_C - 25$.

For fDDA, the 3-9 leftmost bytes of the ICC Dynamic Data consist of the concatenation of the data object listed in Table A.2.

Table A.2 – ICC Dynamic Data (DDA)

Reference	Length	Origin
ICC Dynamic Number Length	1	Card
ICC Dynamic Number	2-8	Card

The ICC Dynamic Number is a time-variant parameter generated by the Card like for instance an unpredictable number or a counter incremented at each transaction.

For CDA, the 32-38 leftmost bytes of the ICC Dynamic Data consist of the concatenation of the data object listed in Table A.3.

Table A.3 – ICC Dynamic Data (CDA)

Reference	Length	Origin
ICC Dynamic Number Length	1	Card
ICC Dynamic Number	2-8	Card
Cryptogram Information Data	1	Card
Application Cryptogram (TC or ARQC)	8	Card
Transaction Data Hash Code	20	Card

For fDDA, the Terminal Dynamic Data is the concatenation of the data objects listed in the default Dynamic Data Authentication Data Object List (DDOL) specified by the Payment System that is present in the Reader. The Terminal Dynamic Data contains the 4-byte Unpredictable Number generated by the Reader.

Note that DDOL in the Card is not supported by the non-C-8 EMV contactless Kernels. Hence the Terminal Dynamic Data is transmitted to the Card by the GET PROCESSING OPTIONS command.

For CDA, the Terminal Dynamic Data is only the 4-byte Unpredictable Number generated by the Reader. The Transaction Data Hash Code in Table A.3 is generated over the concatenation of the data objects listed in Table A.4 using the hash algorithm derived from the Hash Algorithm Indicator (SHA-1).

Table A.4 – Transaction Data

Reference	Origin
PDOL Values (Value field of PDOL Related Data)	Reader
CDOL1 Related Data	Reader
GENERATE AC Response Message • TLV encoded data objects in the order they are returned • With the exception of SDAD	Card

A.3 Dynamic Data Signature Validation

After the RSA certificates are validated as described in sections 6.2 and 6.4, the Reader verifies SDAD as follows:

- Check whether the length of SDAD is equal to the length of the Card public key modulus N_C.
- Recover the data from SDAD, as described in section 8.7.2. This includes computing the hash over the N_C – 22 leftmost bytes of the recovered dynamic application data listed in Table A.5 (i.e. all items except the hash result) concatenated with the Terminal Dynamic Data in the Reader, and comparing the result with the recovered hash result.

Table A.5 – Recovered Dynamic Application Data

Reference	Length	Value
Signed Data Format	1	'05'
Hash Algorithm Indicator	1	'01'
ICC Dynamic Data Length	1	L_{DD}
ICC Dynamic Data	L_{DD}	–
Padding Bytes	$N_C – L_{DD} – 25$	'BB'
Hash Result	20	–

- Check whether the Signed Data Format is '05'.

Note that for fDDA, when an online approval is requested, check that the Signed Data Format is '95' instead of '05'.

- Check whether the Hash Algorithm Indicator is '01'.

For CDA only, the SDAD validation continues as follows:

- Check whether the Cryptogram Information Data in the ICC Dynamic Data recovered from SDAD matches the one returned by the Card in the GENERATE AC Response Message.
- Build the transaction data listed in Table A.4 with the PDOL Values and CDOL1 Related Data sent to the Card, and the GENERATE AC Response Message (except SDAD).
- Apply the hash algorithm derived from the Hash Algorithm Indicator in Table 8.1 (SHA-1) over the transaction data built in the previous step.
- Check whether the resulting hash is the same as the Transaction Data Hash Code retrieved from the recovered ICC Dynamic Data in Table A.5.

If any step above fails, the validation is aborted.

If all the steps above were successful, the SDAD is considered as genuine and the offline data authentication process is successful.

*** END OF DOCUMENT ***