NFC-SEC-01: NFC-SEC Cryptography Standard using ECDH and AES
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scope</td>
</tr>
<tr>
<td>2</td>
<td>Conformance</td>
</tr>
<tr>
<td>3</td>
<td>Normative references</td>
</tr>
<tr>
<td>4</td>
<td>Terms and definitions</td>
</tr>
<tr>
<td>5</td>
<td>Conventions and notations</td>
</tr>
<tr>
<td>5.1</td>
<td>Concatenation</td>
</tr>
<tr>
<td>5.2</td>
<td>Hexadecimal numbers</td>
</tr>
<tr>
<td>6</td>
<td>Acronyms</td>
</tr>
<tr>
<td>7</td>
<td>General</td>
</tr>
<tr>
<td>8</td>
<td>Protocol Identifier (PID)</td>
</tr>
<tr>
<td>9</td>
<td>Primitives</td>
</tr>
<tr>
<td>9.1</td>
<td>Key agreement</td>
</tr>
<tr>
<td>9.1.1</td>
<td>Curve P-192</td>
</tr>
<tr>
<td>9.1.2</td>
<td>EC Key Pair Generation Primitive</td>
</tr>
<tr>
<td>9.1.3</td>
<td>EC Public key validation</td>
</tr>
<tr>
<td>9.1.4</td>
<td>ECDH secret value derivation Primitive</td>
</tr>
<tr>
<td>9.1.5</td>
<td>Random nonces</td>
</tr>
<tr>
<td>9.2</td>
<td>Key Derivation Functions</td>
</tr>
<tr>
<td>9.2.1</td>
<td>KDF for the SSE</td>
</tr>
<tr>
<td>9.2.2</td>
<td>KDF for the SCH</td>
</tr>
<tr>
<td>9.3</td>
<td>Key Usage</td>
</tr>
<tr>
<td>9.4</td>
<td>Key Confirmation</td>
</tr>
<tr>
<td>9.4.1</td>
<td>Key confirmation tag generation</td>
</tr>
<tr>
<td>9.4.2</td>
<td>Key confirmation tag verification</td>
</tr>
<tr>
<td>9.5</td>
<td>Data Encryption</td>
</tr>
<tr>
<td>9.5.1</td>
<td>Initial value of counter (IV)</td>
</tr>
<tr>
<td>9.5.2</td>
<td>Encryption</td>
</tr>
<tr>
<td>9.5.3</td>
<td>Decryption</td>
</tr>
<tr>
<td>9.6</td>
<td>Data Integrity</td>
</tr>
<tr>
<td>9.6.1</td>
<td>Protect data integrity</td>
</tr>
<tr>
<td>9.6.2</td>
<td>Check data integrity</td>
</tr>
<tr>
<td>9.7</td>
<td>Message Sequence Integrity</td>
</tr>
<tr>
<td>10</td>
<td>Data Conversions</td>
</tr>
<tr>
<td>10.1</td>
<td>Integer-to-Octet-String Conversion</td>
</tr>
<tr>
<td>10.2</td>
<td>Octet-String-to-Integer Conversion</td>
</tr>
<tr>
<td>10.3</td>
<td>Point-to-Octet-String Conversion</td>
</tr>
<tr>
<td>10.4</td>
<td>Octet-String-to-Point Conversion</td>
</tr>
<tr>
<td>11</td>
<td>SSE and SCH service invocation</td>
</tr>
<tr>
<td>11.1</td>
<td>Pre-requisites</td>
</tr>
<tr>
<td>11.2</td>
<td>Key Agreement</td>
</tr>
<tr>
<td>11.2.1</td>
<td>Sender (A) Transformation</td>
</tr>
<tr>
<td>11.2.2</td>
<td>Recipient (B) Transformation</td>
</tr>
<tr>
<td>11.3</td>
<td>Key Derivation</td>
</tr>
<tr>
<td>11.3.1</td>
<td>Sender (A) Transformation</td>
</tr>
<tr>
<td>11.3.2</td>
<td>Recipient (B) Transformation</td>
</tr>
<tr>
<td>11.4</td>
<td>Key Confirmation</td>
</tr>
<tr>
<td>11.4.1</td>
<td>Sender (A) Transformation</td>
</tr>
</tbody>
</table>
Introduction

The NFC Security series of standards comprise a common services and protocol Standard and NFC-SEC cryptography standards.

This NFC-SEC cryptography Standard specifies cryptographic mechanisms that use the Elliptic Curves Diffie-Hellman (ECDH) protocol for key agreement and the AES algorithm for data encryption and integrity.

This Standard addresses secure communication of two NFC devices that do not share any common secret data ("keys") before they start communicating with each other.

This Ecma Standard has been adopted by the General Assembly of June 2010.
"DISCLAIMER

This document and possible translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to Ecma International, except as needed for the purpose of developing any document or deliverable produced by Ecma International (in which case the rules applied to copyrights must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by Ecma International or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and ECMA INTERNATIONAL DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."
NFC-SEC-01: NFC-SEC Cryptography Standard using ECDH and AES

1 Scope

This Standard specifies the message contents and the cryptographic methods for PID 01.

This Standard specifies cryptographic mechanisms that use the Elliptic Curves Diffie-Hellman (ECDH) protocol for key agreement and the AES algorithm for data encryption and integrity.

2 Conformance

Conformant implementations employ the security mechanisms specified in this NFC-SEC cryptography Standard (identified by PID 01) and conform to ECMA-385.

The NFC-SEC security services shall be established through the protocol specified in ECMA-385 and the mechanisms specified in this Standard.

3 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ECMA-340, Near Field Communication Interface and Protocol (NFCIP-1)

ECMA-385, NFC-SEC: NFCIP-1 Security Services and Protocol

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


IEEE 1363, IEEE Standard Specifications for Public-Key Cryptography

FIPS 186-2, Digital Signature Standard (DSS)

4 Terms and definitions

For the purposes of this Standard, all terms and definitions from ECMA-385 apply.
5 Conventions and notations

The conventions and notations of ECMA-385 as well as the following apply in this document unless otherwise stated.

5.1 Concatenation

A || B represents the concatenation of the fields A and B: content of A followed by content of B.

5.2 Hexadecimal numbers

(XY) denotes a hexadecimal number XY (i.e. with the Radix of 16) and each pair of characters is encoded in one octet.

6 Acronyms

For the purposes of this Standard, all acronyms from ECMA-385 apply. Additionally, the following acronyms apply.

A Sender, as specified in ECMA-385
AES Advanced Encryption Standard
B Receiver, as specified in ECMA-385
dA Sender’s private EC key
dB Recipient's private EC key
DataLen Length of the UserData
EC Elliptic Curve
ECDH Elliptic Curve Diffie-Hellman
EncData Encrypted data
G The base point on EC
ID_A Sender nfcid3
ID_B Recipient nfcid3
ID_R Any Recipient identification number (e.g. ID_B)
ID_S Any Sender identification number (e.g. ID_A)
IV Initial Value
K Key
KDF Key Derivation Function
KE Encryption Key
KI Integrity Key
MAC Message Authentication Code
MacA /MacB Integrity protection value of Sender/ Recipient
MacTagA Key confirmation tag from Sender
MacTagB Key confirmation tag from Recipient
MK Master Key
NA / NB Nonce generated by Sender/Recipient
7 General

This Standard specifies mechanisms for the Shared Secret Service (SSE) and the Secure Channel Service (SCH) in ECMA-385.

To enable secure communication between NFC devices that do not share any common secret data ("keys") before they start communicating with each other, public key cryptography is used to establish a shared secret between these devices, and more specifically the Elliptic Curve Diffie-Hellman key exchange scheme. This shared secret is used to establish the SSE and the SCH.

8 Protocol Identifier (PID)

This Standard shall use the one octet protocol identifier PID with value 1.

9 Primitives

This Clause specifies cryptographic primitives. Clauses 11 and 12 specify the actual use of these primitives.

Table 1 summarizes the features.
Table 1 – Summary of features

<table>
<thead>
<tr>
<th>Supported services</th>
<th>SSE (see ECMA-385)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SCH (see ECMA-385)</td>
</tr>
<tr>
<td>Key agreement</td>
<td>ECDH P-192</td>
</tr>
<tr>
<td>KDF</td>
<td>AES-XCBC-PRF-128</td>
</tr>
<tr>
<td>Key confirmation</td>
<td>AES-XCBC-MAC-96</td>
</tr>
<tr>
<td>Data encryption</td>
<td>AES128-CTR</td>
</tr>
<tr>
<td></td>
<td>IV Init: AES-XCBC-PRF-128</td>
</tr>
<tr>
<td>Data integrity</td>
<td>AES-XCBC-MAC-96</td>
</tr>
<tr>
<td>Sequence integrity</td>
<td>SN (see ECMA-385)</td>
</tr>
<tr>
<td>Encryption order</td>
<td>Encryption (9.5) before MAC calculation (9.6)</td>
</tr>
</tbody>
</table>

9.1 Key agreement

Peer NFC-SEC entities shall agree on a shared secret using Key agreement mechanism 4 from ISO/IEC 11770-3 and the Elliptic Curves Diffie-Hellman primitives from IEEE 1363 as further specified below.

9.1.1 Curve P-192

Curve P-192 as specified in FIPS 186-2 shall be used.

9.1.2 EC Key Pair Generation Primitive

The private key \( d \) shall be obtained from a random or pseudo-random process conforming to ISO/IEC 18031.

a) Obtain the private key, \( d \), from a random or pseudo-random process conforming to ISO/IEC 18031.

b) Compute the public key, PK, as a point on EC, \( PK = dG \).

9.1.3 EC Public key validation

The EC public key shall be validated as specified in Public Key Validation of ISO/IEC 15946-1.

9.1.4 ECDH secret value derivation Primitive

The ECDH primitive as specified in 7.2.1 ECSVDP-DH of IEEE 1363 shall output the ‘valid’ shared secret \( z \) and ‘invalid’ otherwise.

9.1.5 Random nonces

Each peer NFC-SEC entity should send fresh random nonces with the EC public key of the entity.

The nonces are used to provide more entropy to the keys derived from the shared secret \( z \), and to facilitate the EC key pair management.

The correct generation of these nonces is under the responsibility of the entity.

The entity should guarantee that the nonces it generates have 96 bits of entropy valid for the duration of the protocol. The nonces used in an NFC-SEC transaction shall be cryptographically uncorrelated with the nonces from a previous transaction.
See ISO/IEC 18031 for further recommendations on random number generation.

9.2 Key Derivation Functions

Two Key Derivation Functions (KDF) are specified; one for the SSE and one for the SCH.

The KDFs shall use AES in XCBC-PRF-128 mode as specified in A.1.

For the following sections KDF is:

\[
\text{KDF} (K, S) = \text{AES-XCBC-PRF-128}_K (S)
\]

The random source (nonces + shared secret z obtained from 9.1.4) used for the SCH shall be different from the random source used for the SSE.

9.2.1 KDF for the SSE

The KDF for the SSE is:

\[
\text{MKSSE} = \text{KDF-SSE} (\text{Nonce}_S, \text{Nonce}_R, \text{SharedSecret}, \text{IDS}, \text{IDR})
\]

Detail of the KDF-SSE function:

\[
S = (\text{Nonce}_S[0..63] || \text{Nonce}_R[0..63])
\]

\[
\text{SKEYSEED} = \text{KDF} (S, \text{SharedSecret})
\]

\[
\text{MK}_{\text{SSE}} = \text{KDF} (\text{SKEYSEED}, S || \text{IDS} || \text{IDR} || (01))
\]

9.2.2 KDF for the SCH

The KDF for the SCH is:

\[
\{\text{MK}_{\text{SCH}}, \text{KE}_{\text{SCH}}, \text{KI}_{\text{SCH}}\} = \text{KDF-SCH} (\text{Nonce}_S, \text{Nonce}_R, \text{SharedSecret}, \text{IDS}, \text{IDR})
\]

Detail of the KDF-SCH function:

\[
S = (\text{Nonce}_S[0..63] || \text{Nonce}_R[0..63])
\]

\[
\text{SKEYSEED} = \text{KDF}(S, \text{SharedSecret})
\]

\[
\text{MK}_{\text{SCH}} = \text{KDF} (\text{SKEYSEED}, S || \text{IDS} || \text{IDR} || (01))
\]

\[
\text{KE}_{\text{SCH}} = \text{KDF} (\text{SKEYSEED}, \text{MK}_{\text{SCH}} || S || \text{IDS} || \text{IDR} || (02))
\]

\[
\text{KI}_{\text{SCH}} = \text{KDF} (\text{SKEYSEED}, \text{KE}_{\text{SCH}} || S || \text{IDS} || \text{IDR} || (03))
\]

9.3 Key Usage

Each derived key \(\text{MK}_{\text{SCH}}, \text{KE}_{\text{SCH}}, \text{KI}_{\text{SCH}}\) and \(\text{MK}_{\text{SSE}}\) should be used only for the purpose specified in Table 2.

The Keys \(\text{MK}_{\text{SCH}}, \text{KE}_{\text{SCH}}, \text{KI}_{\text{SCH}}\) and \(\text{MK}_{\text{SSE}}\) shall be different for each NFC-SEC transaction.
### 9.4 Key Confirmation

When a key is derived using one of the KDF processes described in 9.2 both NFC-SEC entities check that they indeed have the same key. Each entity shall generate a key confirmation tag as specified in 9.4.1 and shall send it to the peer entity. Entities shall verify the key confirmation tag upon reception as specified in 9.4.2.

This key confirmation mechanism is according to 9 Key Confirmation of ISO/IEC 11770-3.

The MAC used for Key Confirmation (MacTag) shall be AES in XCBC-MAC-96 mode as specified in A.2.

#### 9.4.1 Key confirmation tag generation

MacTag, the Key confirmation tag, equals

\[
\text{MAC-KC} (K, \text{MsgID}, \text{IDS}, \text{IDR}, \text{PKS}, \text{PKR})
\]

and shall be calculated using

\[
\text{AES-XCBC-MAC-96}_K (\text{MsgID} \| \text{IDS} \| \text{IDR} \| \text{PKS} \| \text{PKR}), \text{ specified in Annex A.2, with key K.}
\]

#### 9.4.2 Key confirmation tag verification

'status', the return value of

\[
\text{MAC-KC-VER} (K, \text{MsgID}, \text{IDS}, \text{IDR}, \text{PKS}, \text{PKR}, \text{MacTag'})
\]

is true

if MacTag' equals MAC-KC (K, \text{MsgID}, \text{IDS}, \text{IDR}, \text{PKS}, \text{PKR})

### 9.5 Data Encryption

The data encryption algorithm used is AES as specified in 5.1 AES of ISO/IEC 18033-3.

The data encryption mode shall be CTR mode as specified in 10 Counter (CTR) Mode of ISO/IEC 10116.

#### 9.5.1 Initial value of counter (IV)

To avoid having to send the initial value of the counter, it shall be computed by both entities from the nonces.

IV, the initial value of the counter, equals

\[
\text{MAC-IV} (\text{MK}, \text{KI}, \text{NonceS}, \text{NonceR})
\]

and shall be calculated using

\[
\text{AES-XCBC-PRF-128MK} (\text{KI} \| \text{NonceS} \| \text{NonceR} \| (04)), \text{ specified in Annex A.1, with key MK.}
\]

#### 9.5.2 Encryption

The data shall be encrypted using the Encryption Key KE as specified in 10.2 Encryption of ISO/IEC 10116:

\[
\text{EncData} = \text{ENC}_{KE} (\text{Data})
\]

---

**Table 2 – Key usage**

<table>
<thead>
<tr>
<th>Key</th>
<th>Key description</th>
<th>Key usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{MK}_{\text{SCH}}$</td>
<td>Master Key for SCH</td>
<td>Key Verification for the Secure Channel Keys</td>
</tr>
<tr>
<td>$\text{KE}_{\text{SCH}}$</td>
<td>Encryption Key for SCH</td>
<td>Encryption of data packets sent through SCH</td>
</tr>
<tr>
<td>$\text{KI}_{\text{SCH}}$</td>
<td>Integrity protection Key for SCH</td>
<td>Integrity protection of data packets sent through SCH</td>
</tr>
<tr>
<td>$\text{MK}_{\text{SSE}}$</td>
<td>Master Key for SSE</td>
<td>Master Key for SSE used as Shared secret to be passed to the upper layer and as Key Verification</td>
</tr>
</tbody>
</table>
Since the mode is CTR, no padding of the data shall be applied.

9.5.3 Decryption

The encrypted data shall be decrypted using the Encryption Key KE as specified in 10.3 Decryption of ISO/IEC 10116:

\[ \text{Data'} = \text{DEC}_\text{KE} (\text{EncData}) \]

9.6 Data Integrity

Integrity of all data transferred on the SCH shall be preserved through a MAC.

The MAC used for Data Integrity shall be AES in XCBC-MAC-96 mode as specified in A.2.

9.6.1 Protect data integrity

Mac, the Message Authentication Code, equals

\[ \text{MAC-DI (KI, SN, DataLen, EncData)} \]

and shall be calculated using

\[ \text{AES-XCBC-MAC-96}_\text{Kl} (\text{SN || DataLen || EncData}) \]

specified in Annex A.2, with key Kl.

9.6.2 Check data integrity

‘status’, the return value of

\[ \text{MAC-DI-VER (KI, SN, DataLen, EncData, Mac')} \]

is true

if Mac’ equals MAC-DI (KI, SN || DataLen || EncData)

9.7 Message Sequence Integrity

Message sequence integrity shall be handled as specified in 12.3 of ECMA-385.

The SNV value shall be in the range of 0 to \(2^{24} - 1\); with the initial value of 0.

Entities shall terminate the SCH when the SNV has reached \(2^{24} - 1\).

10 Data Conversions

10.1 Integer-to-Octet-String Conversion

Input: A non-negative integer \(x\), and the intended length \(k\) of the octet string satisfying: \(2^k > x\).

Output: An octet string \(M\) of length \(k\) octets.

Let \(M_1, M_2, ..., M_k\) be the octets of \(M\) from leftmost to rightmost.

The octets of \(M\) shall satisfy:

\[ x = \sum_{i=1}^{k} 2^{(k-i)} M_i \]

10.2 Octet-String-to-Integer Conversion

Input: An octet string \(M\) of length \(k\) octets.
Output: An integer $x$.

Let $M_1, M_2, \ldots, M_k$ be the octets of $M$ from leftmost to rightmost.

$M$ shall be converted to an integer $x$ satisfying:

$$x = \sum_{i=1}^{k} 2^{(k-i)} M_i$$

10.3 Point-to-Octet-String Conversion

The point on the EC shall be converted to an octet string in the following way:

Input: An elliptic curve point $P = (x_P, y_P)$.

Output: An octet string $PO$ with the $y$-coordinate in the leftmost octet and the $x$-coordinate in the remainder of the octet string.

1. Convert the field element $x_P$ to an octet string $X$ as specified in 10.1.

2. Compute the bit $\bar{y}_P$ as specified in 6.6: Elliptic curve point / octet string conversion: EC2OSPE and OS2ECPE of ISO/IEC 15946-1.

3. Assign the value (02) to the single octet $PC$ if $\bar{y}_P$ is 0, or the value (03) if $\bar{y}_P$ is 1.

4. The result is the octet string $PO = PC || X$.

10.4 Octet-String-to-Point Conversion

The octet string shall be converted to a point on the EC in the following way:

Input: An octet string $PO$, with the $y$-coordinate in the leftmost octet and the $x$-coordinate in the remainder of the octet string.

Output: An elliptic curve point $P = (x_P, y_P)$.

1. Parse $PO$ as follows: $PO = PC || X$, where $PC$ is a single octet, and $X$ is an octet string of length $k$ octets.

2. Convert $X$ to a field element $x_P$ as specified in 10.2.

3. Verify that $PC$ is either (02) or (03). It is an error if this is not the case.

4. Set the bit $\bar{y}_P$ to be equal to 0 if $PC = (02)$, or 1 if $PC = (03)$.

5. Convert $(x_P, \bar{y}_P)$ to an elliptic curve point $(x_P, y_P)$ as specified in 6.6: Elliptic curve point / octet string conversion: EC2OSPE and OS2ECPE of ISO/IEC 15946-1.

6. The result is $P = (x_P, y_P)$.

11 SSE and SCH service invocation

SSE and SCH are invoked by establishment of a shared secret between two NFC-SEC entities using the key agreement and key confirmation protocol specified in ECMA-385, in the way illustrated in Figure 1 and further specified in this Clause.
11.1 Pre-requisites

Before starting the service, the followings shall be available on each NFC-SEC entity:

- Its own EC public and private key, generated as specified in 9.1.2.
  
  NOTE It is outside the scope of this Standard when (and at which frequency) this EC key pair is generated.

- Its own nfcid3 and the other NFC-SEC entity’s nfcid3 as specified in ECMA-340.
### 11.2 Key Agreement

<table>
<thead>
<tr>
<th>Sender (A)</th>
<th>PDU</th>
<th>Recipient (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate nonce NA</td>
<td></td>
<td>Generate nonce NB</td>
</tr>
<tr>
<td>Compress $Q_A$</td>
<td></td>
<td>Compress $Q_B$</td>
</tr>
<tr>
<td>Send to B</td>
<td>$A\rightarrow B$: ACT_REQ ($QA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Send to A</td>
</tr>
<tr>
<td>Reconstruct $Q_B'$ from $QB'$</td>
<td></td>
<td>Reconstruct $Q_A'$ from $QA'$</td>
</tr>
<tr>
<td>Check $Q_B'$</td>
<td></td>
<td>Check $Q_A'$</td>
</tr>
<tr>
<td>Compute shared secret: $Z$</td>
<td></td>
<td>Compute shared secret: $Z$</td>
</tr>
</tbody>
</table>

#### 11.2.1 Sender (A) Transformation

1. Generate a nonce NA as specified in 9.1.5.
2. Ensure $QA$ equals the octet string of $QA$ as specified in 10.3.
3. Send $QA || NA$ as the payload of the ACT_REQ PDU.
4. Receive $QB' || NB'$ from the payload of the ACT_RES PDU.
5. Reconstruct $Q_B'$ as specified in 10.4.
   a) If the public keys have already been received, the previously calculated and stored value $Q_B'$ may be reused and the following steps may be skipped.
6. Verify that $QB'$ is a valid key for the EC parameters as specified in 9.1.3. If it is invalid, then set the 'PDU content valid' to false in the Protocol Machine and skip step 7 and 8.
7. Use the Diffie-Hellman primitive in 9.1.4. If its output $z$ is 'invalid' then set the 'PDU content valid' to false in the Protocol Machine and skip step 8.

#### 11.2.2 Recipient (B) Transformation

1. Receive $QA' || NA'$ from the payload of the ACT_REQ PDU
2. Generate a nonce NB as specified in 9.1.5.
3. Ensure $QB$ equals the octet string of $QB$ as specified in 10.3.
4. Send $QB || NB$ as the payload of the ACT_RES PDU.
5. Reconstruct Qₐ' from QA' as specified in 10.4.
   a) If the public keys have already been received, the previously calculated and stored value Qₐ' may be reused and the following steps may be skipped.

6. Verify that QA' is a valid key for the EC parameters as specified in 9.1.3. If it is invalid, then set the ‘PDU content valid’ to false in the Protocol Machine and skip step 7 and 8.

7. Use the Diffie-Hellman primitive in 9.1.4. If its output z is ‘invalid’, then set the ‘PDU content valid’ to false in the Protocol Machine and skip step 8.


11.3 Key Derivation

11.3.1 Sender (A) Transformation

For the SSE service, derive MKₛₑₑ = KDF-SSE (NA, NB', Z, IDₐ, IDₐ) as specified in 9.2.1.

For the SCH service, derive {MKₛᶜ’h, KEₛᶜ’h, Kₛᶜ’h} = KDF-SCH (NA, NB', Z, IDₐ, IDₐ) as specified in 9.2.2.

11.3.2 Recipient (B) Transformation

For the SSE service, derive MKₛₑₑ = KDF-SSE (NA', NB, Z, IDₐ, IDₐ) as specified in 9.2.1.

For the SCH service, derive {MKₛᶜ’h, KEₛᶜ’h, Kₛᶜ’h} = KDF-SCH (NA', NB, Z, IDₐ, IDₐ) as specified in 9.2.2.

11.4 Key Confirmation

<table>
<thead>
<tr>
<th>Sender (A)</th>
<th>PDU</th>
<th>Recipient (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute key confirmation tag: MacTagₐ(MK)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Send to B</td>
<td>A→B: VFY_REQ (MacTagₐ)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Check key confirmation tag received from A: MacTagₐ(MK)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compute key confirmation tag: MacTagₐ(MK)</td>
</tr>
<tr>
<td></td>
<td>A→B: VFY_RES (MacTagₐ)</td>
<td>Send to A</td>
</tr>
<tr>
<td></td>
<td>Check key confirmation tag received from B: MacTagₐ(MK)</td>
<td></td>
</tr>
<tr>
<td>For SSE, set the Shared Secret Value to MK</td>
<td>For SSE, set the Shared Secret Value to MK</td>
<td></td>
</tr>
</tbody>
</table>

11.4.1 Sender (A) Transformation

1. Compute the key confirmation tag from A to B MacTagₐ = MAC-KC(MK, (03), IDE, IDₐ, QA, QB') as specified in 9.4.1.
2. Send MacTagA as the payload of the VFY_REQ PDU.

3. Receive MacTagB’ from the payload of the VFY_RES PDU.

4. Check the key confirmation tag from B to A. Set ‘PDU content valid’ in the Protocol Machine to the output of MAC-KC-VER(MK, (02), IDB, IDA, QB’, QA, MacTagB’) as specified in 9.4.2. If it is ‘invalid’ then skip step 5.

5. For the SSE service, set SharedSecret = MK_{SSE}.

11.4.2 Recipient (B) Transformation

1. Receive MacTagA’ from the payload of the VFY_REQ PDU.

2. Check the key confirmation tag from A to B. Set ‘PDU content valid’ in the Protocol Machine to the output of MAC-KC-VER (MK, (03), IDA, IDB, QA’, QB, MacTagA’) as specified in 9.4.2. If it is ‘invalid’ then skip step 3, 4 and 5.

3. Compute the key confirmation tag from B to A MacTagB = MAC-KC(MK, (02), IDB, IDA, QB, QA’) as specified in 9.4.1.

4. Send MacTagB as the payload of the VFY_RES PDU.

5. For the SSE service, set SharedSecret = MK_{SSE}.

12 SCH data exchange

After invocation of the SCH as specified in 11, the data exchange between two NFC-SEC entities uses the protocol specified in ECMA-385 as illustrated in Figure 2 and further specified in this Clause.

![Figure 2 – SCH: protocol overview](image-url)
12.1 Preparation

NFC-SEC entity (AA and BB) shall perform the following preparatory steps:

1. Generate the initial value of the CTR counter \( IV = MAC-IV (MK, KI, NAA, NBB) \) as specified in 9.5.1.
2. Initialise the Sequence Number variable (SNV) as specified in 9.7.

12.2 Data Exchange

<table>
<thead>
<tr>
<th>Sending peer entity AA (A or B)</th>
<th>PDU transmitted</th>
<th>Receiving peer entity BB (A or B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Receive UserData from SendData SDU</td>
<td>ENC (SNV</td>
<td></td>
</tr>
<tr>
<td>• Check SNV</td>
<td>Communication direction is indicated by arrow character Payload is between ()</td>
<td>• Check sequence integrity</td>
</tr>
<tr>
<td>• Increment SNV</td>
<td></td>
<td>• Check data integrity</td>
</tr>
<tr>
<td>• Encrypt Data: EncData</td>
<td></td>
<td>• Decrypt data</td>
</tr>
<tr>
<td>• Apply MAC: Mac</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

12.2.1 Send

To send data, the sending NFC-SEC peer entity AA (A or B) shall perform the following steps:

1. Receive UserData from the SendData SDU.
2. If SNV = 2\(^{24} \)-1, then set the ‘PDU content valid’ to false in the Protocol Machine, otherwise proceed to the next step.
3. Increment the SNV as specified in 12.3 of ECMA-385.
4. Compute the encrypted data EncData from UserData as specified in 9.5.2.
5. Compute the MAC Mac on SNV || DataLen || EncData as specified in 9.6.1.
6. Send SNV || DataLen || EncData || Mac as the payload of the ENC PDU.

12.2.2 Receive

To receive data, the receiving NFC-SEC peer entity BB (A or B) shall perform the following steps:

1. Receive SNV || DataLen || EncData || Mac from the payload of the ENC PDU.
2. If SNV = 2\(^{24} \)-1, then set the ‘PDU content valid’ to false in the Protocol Machine, otherwise proceed to the next step.
3. Check the sequence integrity as specified in 12.3 of ECMA-385.
4. Check the data integrity of SNV || DataLen || EncData as specified in 9.6.2. If it is invalid, then set the 'PDU content valid' to false in the Protocol Machine; otherwise proceed to the next step.

5. Compute the decrypted data UserData from EncData as specified in 9.5.3.
Annex A  
(normative)

AES-XCBC-PRF-128 and AES-XCBC-MAC-96 algorithms

A.1 AES-XCBC-PRF-128

The AES-XCBC-PRF-128 algorithm is a variant of the basic CBC-MAC with obligatory “10* padding”, which makes it secure for messages of arbitrary length.

The encryption operations must be accomplished using AES with a 128-bit key.

Given a 128-bit secret key K, AES-XCBC-PRF-128 is calculated as follows for a message M that consists of n blocks, M[1] ... M[n], in which the block size of blocks M[1] ... M[n-1] is 128 bits and the block size of block M[n] is between 1 and 128 bits:

1. Derive 3 128-bit keys (K1, K2 and K3) from the 128-bit secret key K, as follows:
   K1 = (01010101010101010101010101010101) encrypted with Key K
   K2 = (02020202020202020202020202020202) encrypted with Key K
   K3 = (03030303030303030303030303030303) encrypted with Key K
2. Define E[0] = 0x00000000000000000000000000000000
3. For each block M[i], where i = 1 ... n-1:
   XOR M[i] with E[i-1],
   then encrypt the result with Key K1, yielding E[i].
4. For block M[n]:
   a. If the block size of M[n] is 128 bits:
      XOR M[n] with E[n-1] and Key K2,
      then encrypt the result with Key K1, yielding E[n].
   b. If the block size of M[n] is less than 128 bits:
      i. Pad M[n] with a single "1" bit, followed by the number of "0" bits (possibly none) required to increase M[n]'s block size to 128 bits (this is the “10* padding”)
      ii. XOR M[n] with E[n-1] and Key K3,
      then encrypt the result with Key K1, yielding E[n].
5. The output is the last 128 bits block E[n].

A.2 AES-XCBC-MAC-96

The AES-XCBC-MAC-96 algorithm is the AES-XCBC-PRF-128 algorithm, followed by a truncation step:

1. Take the first 96 bits of E[n].

Upon sending, the truncated value is stored within the authenticator field (Mac).

Upon receipt, the entire 128-bit value is computed and the first 96 bits are compared to the value stored in the authenticator field (Mac).
Annex B  
(normative)

Fields sizes

Table B.1 – Fields sizes

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>NA</td>
<td>96 bits</td>
</tr>
<tr>
<td>NB</td>
<td>96 bits</td>
</tr>
<tr>
<td>dA</td>
<td>192 bits</td>
</tr>
<tr>
<td>dB</td>
<td>192 bits</td>
</tr>
<tr>
<td>DataLen</td>
<td>24 bits</td>
</tr>
<tr>
<td>QA</td>
<td>384 bits</td>
</tr>
<tr>
<td>QB</td>
<td>384 bits</td>
</tr>
<tr>
<td>QA</td>
<td>200 bits</td>
</tr>
<tr>
<td>QB</td>
<td>200 bits</td>
</tr>
<tr>
<td>Z</td>
<td>192 bits</td>
</tr>
<tr>
<td>MK</td>
<td>128 bits</td>
</tr>
<tr>
<td>KE</td>
<td>128 bits</td>
</tr>
<tr>
<td>KI</td>
<td>128 bits</td>
</tr>
<tr>
<td>MacTagA</td>
<td>96 bits</td>
</tr>
<tr>
<td>MacTagB</td>
<td>96 bits</td>
</tr>
<tr>
<td>IV</td>
<td>128 bits</td>
</tr>
<tr>
<td>SNV</td>
<td>24 bits</td>
</tr>
<tr>
<td>Mac</td>
<td>96 bits</td>
</tr>
</tbody>
</table>
Annex C
(informative)

Informative references

<table>
<thead>
<tr>
<th>RFC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4303</td>
<td>IP Encapsulating Security Payload (ESP)</td>
</tr>
<tr>
<td>4306</td>
<td>Internet Key Exchange (IKEv2) Protocol</td>
</tr>
<tr>
<td>4434</td>
<td>The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol (IKE)</td>
</tr>
<tr>
<td>3566</td>
<td>The AES-XCBC-MAC-96 Algorithm and Its Use With IPSec</td>
</tr>
</tbody>
</table>

The AES-XCBC-PRF-128 algorithm is specified in RFC 4434 (IPSEC v2).

The AES-XCBC-MAC-96 algorithm is specified in RFC 3566 (IPSEC v2).

The KDF is specified in RFC 4306 (IPSEC v2).

The ENC then MAC protection mechanism is specified in RFC 4303 (IPSEC v2).