AN14137

NTAG X DNA - Features and hints

Rev. 1.0 — 27 May 2025 Application note

Document information

Information	Content
Keywords	NTAG X DNA
Abstract	This document serves system developers and describes main features of NTAG X DNA and how to integrate security features into the NFC system.

1 Introduction

1.1 About this document

This document addresses developers who are developing applications based on NTAG X DNA.

This document shall be used in addition to NTAG X DNA data sheet ref.[11]. The best use of this application note is achieved by reading the mentioned data sheet in advance.

Note: *This application note does not replace any of the relevant functional specifications, data sheets, or design guides.*

1.2 Key features of NTAG X DNA

- Dual interface:
  - Contact interface (**I²C** T=1 over I²C) for wired communication (for example, MCU)
  - Contactless interface (**NFC** Type 4 Tag) for interaction with NFC Mobile and NFC reader
- **Security:** AES 128/256 + PKI ECC p256, Common Criteria EAL6+
- Energy harvesting (20mW), battery-less: easy to meet regulations
- App-less: compatible with any NFC mobile
- Data rate up to 848kbit/s (HF), 1 MHz on I²C
- Memory: **16kB**
- NTAG services and EdgeLock® 2GO including credentials delivery
- Package: HVQFN20, WLCSP16, wafer delivery form

1.3 Use cases

- Secure authentication of devices
- Secure anti-counterfeit solution for accessories
- Secure device identity storage, tracking, and protection (Digital passport - DPP, battery passport)
- Secure device parameterization, diagnosis, and maintenance
- Secure (un)pairing of accessories: Wi-Fi, Bluetooth
- Secure (de)commissioning of IoT devices: Matter, Thread, Zigbee

1.4 Target applications

- Mobile accessories
- Gaming accessories/ peripherals
- Smart home devices
- IP Cameras
- Gateway/Routers
- Industrial
- Battery
- Smart Appliance
- Healthcare
- Retail

1.5 Standards compliancy

1.5.1 ISO 14443

NTAG X DNA is fully compliant to all layers (1, 2 ref.[1], 3 ref.[2], 4 ref.[3]) of ISO/IEC 14443.

1.5.2 ISO 7816-4

NTAG X DNA is fully compliant to ISO/IEC 7816-4 ref.[4].

1.5.3 NFC Forum compliancy

NFC tag is a contactless tag capable of storing NDEF data, which interoperates with ISO 14443 infrastructure (or other) and NFC devices as defined by the NFC Forum specifications. NFC Forum defines a logical data structure for storing NDEF message on a Tag.

The file structure on NTAG X DNA complies to NFC Forum Tag 4 Type ref.[9]. There are two required files:

- CC file size is 32 bytes, generally used for defining NDEF structure, info on access rights for NFC device, optionally presence of Proprietary Files. It is pre-configured as NFC Forum Tag 4 Type, NDEF V2.0.
- NDEF file size is 256 bytes. On delivery, it is empty and all types of NDEF messages/records can be programmed.

1.5.4 I²C compliancy

NTAG X DNA is compliant to Global Platform APDU Transport over SPI / I2C Version 1.0 ref.[10].

1.6 Notation used in this document

The following symbols are used to abbreviate operations in the examples:

Table 1. Notation

Symbol	Description
=	Preparation of data by SAM, PICC, or host
< or >	Direction of communication
||	The concatenation operation
$\oplus$	exclusive-OR operation
X << 1	The bit string that results from discarding the leftmost bit of the bit string X and appending a ‘0’ bit on the right
0^s	The bit string that consists of s ‘0’ bytes
E_{AES}(Kx, M)	AES-128 encipher in CBC mode, IV all 0x00, using key - K of number x, M is cipher input
D_{AES}(Kx, M)	AES-128 decipher in CBC mode, IV all 0x00, using key - K of number x, M is cipher input
MAC(K,M)	Message authentication code of message M using secret key K
MAC_t(K,M)	Truncated message authentication code of message M using secret key K. Truncated to 8 bytes, using S14 || S12 || S10 || S8 || S6 || S4 || S2 || S0. Even-numbered bytes shall be retained in MSB first order.
KDF: PRF(key, message) = CMAC(Kx, message)	A NIST recommended key derivation using pseudorandom functions. Pseudo random function: CMAC algorithm

1.7 Byte order

1.7.1 LSB representation

The represented least significant byte (LSB) first are:

- ISO/IEC 14443 parameters during the activation
- plain command parameters consisting of multiple bytes
  Example:
  ReadData command. Parameters, Offset, and Length are 3 bytes long. The values shown below read from FileNo 0x02, starting at offset 0x02 and 0x10 (16d) bytes to be read.

Table 2. ReadData

CLA	CMD	P1	P2	Lc	FileNo	Offset			Length			Le
90	AD	00	00	07	02	02	00	00	10	00	00	00

1.7.2 MSB representation

Represented as most significant byte (MSB) first are:

- cryptographic parameters
- keys
- random numbers exchanged during authentication
- TI (Transaction Identifier)
- computed MACs

1.8 Value presentation

Unless otherwise specified, all data, values, and cryptograms are expressed in a hexadecimal format for readability (01 means 01h, or 0x01).

2 Supporting tools

NXP offers supportive tools to ease implementation and configuration of the NTAG X DNA:

- AN14137 - NTAG X DNA Features and hints ref.[13]
- AN14362 - NTAG X DNA Energy Harvesting ref.[14]
- AN14513 - NTAG X DNA Dual Interface ref.[15]
- UG10083 - NTAG X DNA Quick start guide with support package ref.[16]

3 Personalization example

The following example shows how to personalize NTAG X DNA in order to perform ECC Unilateral Authentication in the field.

3.1 Example system

In the following example, the consumables authentication use case is shown. Verification side (NFC reader/host) checks for authenticity of the NTAG X DNA on the other side (either with Public key or AES-128/256 key).

Example setup:

1. rootCA is created in a secure environment/backend. rootCA.Priv signs the persoCA.Pub (intermediate certificate) and provides persoCA.Cert to the Personalization device
2. persoCA is in the Personalization device. persoCA.Priv signs the DeviceLeaf.Pub (leaf certificate - i.e. public key retrieved from a ManageKeyPair), generating DeviceLeaf.Cert. Personalization device writes the persoCA.Cert (parent/intermediate cert) and DeviceLeaf.Cert (leaf cert) to the NTAG X DNA device.
3. a reader in the field executing NFC transactions, holds the rootCA.Pub key, allowing him to authenticate the NTAG X DNA device by validating the certificate chain.

if the persoCA.Priv of a single perso device would leak (for example, device gets stolen), all certificates of that perso device can be revoke in one shot by providing the serial number of persoCA.Cert to NFC reader devices to list that certificate for denial. Without the need to replace the rootCA, which is still secure in the backend and/or other perso devices.

An NFC reader device in the field will:

- receive the signature from NTAG X DNA, which NTAG X DNA created with DeviceLeaf.Priv;
- request DeviceLeaf.Cert and extract DeviceLeaf.Pub to validate the received signature;
- request PersoCA.Cert and extract PersoCA.Pub to validate the DeviceLeaf.Cert;
- use the pre-configured rootCA.Pub to validate the PersoCA.Cert.

3.2 Certificate chain creation

Certificates are issued and signed by certificates that reside higher in the certificate hierarchy. The validity and trustworthiness of a given certificate is determined by the corresponding validity of the certificate that signed it. The certificate chain of trust results in a root CA signing an intermediate CA, that in turn signs a leaf certificates.

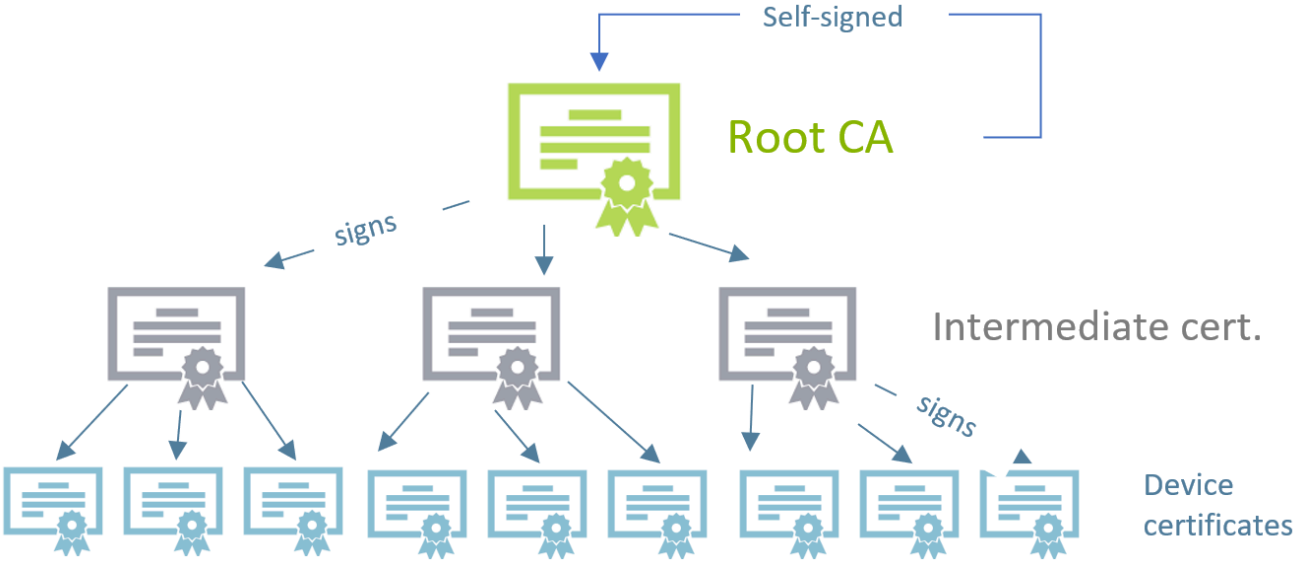

Figure 1. Certificate hierarchy

3.2.1 Create RootCA certificate

1. Check available ECC curve types.
   (prime256v1 and brainpoolP256r1 curves are supported by NTAG X DNA)

```
openssl ecparam -list_curves
```

2. Create a key pair with a chosen ECC curve and write it to a file in PEM format.

```
openssl ecparam -genkey -name prime256v1 ^
 -outform DER -noout -out rootCA_keypair.der
```

3. (optional) Extract the public key (rootCA.Pub) into a DER formatted file. This key will be programmed to NTAG X DNA as CARootKey.

```
openssl ec -in rootCA_keypair.der -pubout -outform DER -out rootCA_pubkey.der
```

4. Display the rootCA keypair to get the value of the Public key, which will be programmed as CARootKey.

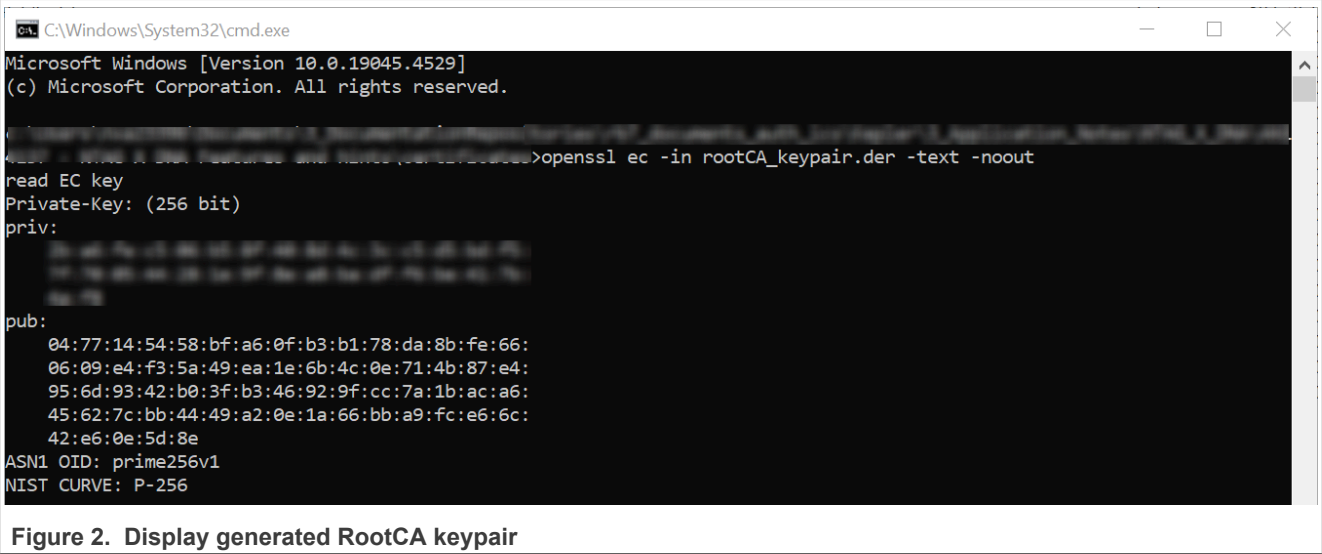

Figure 2. Display generated RootCA keypair

5. Create a (self) signing request.

```
openssl req -new -sha256 ^
 -key rootCA_keypair.der ^
 -out rootCA_cert.csr ^
 -subj "/O=ACME/OU=MightyComp/CN=ACME RootCA"
```

6. Verify CSR information.

```
openssl req -text -in rootCA_cert.csr -noout -verify
```

7. Include an X509 v.3 extension. Extension can be saved in a separate file (for example, x509.ext) and referred in the command line later.

```
[ ca ]
# X509 extensions for a ca
basicConstraints       = CA:TRUE
```

Optionally ARExtension can be used, which maps the Certificate's Access Rights of NTAG X DNA. Access Rights of certain public key can be also set during programming of it (CARootKey).

```
[ ca ]
# X509 extensions for a ca - NXP ARExtension
1.3.6.1.4.1.28137.64.1 = critical,DER:82:07:D2:76:00:00:85:01:01:03:FF
```

Where:

- 1.3.6.1.4.1.28137.64.1. is an NXP OID
- 82:07 is ARType - Access rights for application with the given DFName and length
- D2:76:00:00:85:01:01 is NDEF Application ID
- 03:FF are Access Rights mapping of granted access conditions

8. Self-Sign CSR (certificate signing request).

```
openssl x509 ^
 -req ^
 -extfile x509.ext ^
 -extensions ca ^
 -signkey rootCA_keypair.der ^
 -in rootCA_cert.csr ^
 -days 3650 ^
 -outform DER ^
 -out rootCA_cert.der
```

9. Display the cert content in human readable form (-text) and without PEM output (-noout).

```
openssl x509 -in rootCA_cert.der -text -noout
```

3.2.2 Create PersoCA (intermediate) certificate

1. Create a key pair with choosen ECC curve and write it to a file in DER format.

```
oopenssl ecparam -genkey -name prime256v1 ^
 -outform DER -noout -out personalizationCA_keypair.der
```

2. Create self-signing request (CSR).

```
openssl req -new -sha256 ^
 -key personalizationCA_keypair.der ^
 -out personalizationCA_cert.csr ^
 -subj "/O=ACME/OU=PersoCompany/CN=ACME P1 IntermediateCA"
```

3. Verify CSR information.

```
openssl req -text -in personalizationCA_cert.csr -noout -verify
```

4. Include an X509 v.3 extension. Extension can be saved in a separate file (e.g. x509.ext) and reffered in the command line later.

```
[ ca ]
# X509 extensions for a ca
basicConstraints       = CA:TRUE
```

Optionally, ARExtension can be used, which maps Certificate's Access Rights of NTAG X DNA. Access Rights of certain public key can be also set during programming of it (CARootKey).

```
[ ca ]
# X509 extensions for a ca - NXP ARExtension
1.3.6.1.4.1.28137.64.1 = critical,DER:82:07:D2:76:00:00:85:01:01:03:FF
```

Where:

- 1.3.6.1.4.1.28137.64.1. is an NXP OID
- 82:07 is ARType - Access rights for application with the given DFName and length
- D2:76:00:00:85:01:01 is NDEF Application ID
- 03:FF are Access Rights mapping of granted access conditions

5. Sign CSR (certificate signing request) with RootCA's Private Key.

```
openssl x509 ^
 -req ^
 -extfile x509.ext ^
 -extensions ca ^
 -CA rootCA_cert.der ^
 -CAkey rootCA_keypair.der ^
 -in personalizationCA_cert.csr ^
 -days 3650 ^
 -outform DER ^
 -out personalizationCA_cert.der
```

6. Display the cert content in human reeadable form (-text) and without PEM output (-noout).

```
openssl x509 -in personalizationCA_cert.pem -text -noout
```

3.2.3 Create DeviceLeaf certificate

For the Device's Leaf certificate creation an ECC key pair is needed. There are 2 options to create a keypair for the Device Leaf certificate:

1. Create it off-IC (not with NTAG X DNA) on some host and then program it into respective certificate repository slot (ManageKey command, targeting ECCKey and KeySlot number)
2. Create it on-IC (with NTAG X DNA) (ManageKey command with option to create a keypair on the NTAG X DNA) - use of OpenSSL Provider plug-in.

This example shows option 2 as it is considered as safest. Private key generated by key pair generation in NXP X DNA will stay secured in the IC.

1. Create a key pair with choosen ECC curve and write it to a file in DER format.

```
openssl ecparam -genkey -name prime256v1 ^
 -outform DER -noout -out dl_keypair.der
```

2. Create Public Key file in ASN.1 format (dl_pubkey.der) with Public Key contents.

```
30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00
 04 A9 D7 DD 8F 03 F6 38 E1 CC BD 31 8C 0E F3 12 54 2E 45 88 1F D9 A3 47 5D
 8A F1 CB 50 1A 98 0C 1D 8D 3C A5 9C 93 5A E9 C8 DC 5E B9 D1 91 EB F8 BF 48
 5A D3 FA B3 66 00 43 BD 14 A3 B4 4D DA 50 0D
```

3. Create CSR.

```
openssl req -new -sha256 -key dl_keypair.der ^
        -out dl.csr -subj "/O=ACME/OU=ApparelManuf/CN=Device Leaf
 uid-010203040506"
```

4. Create DeviceLeaf certificate, force to use Public Key returned from NTAG X DNA.

```
openssl x509 -req -days 3650 -extfile x509_no_ca.ext -in dl.csr
 -CAcreateserial -CAkey personalizationCA_keypair.der -CA
 personalizationCA_cert.der -set_serial 0x010203040506 -out
 deviceLeaf_cert.der -force_pubkey dl_pubkey.der
```

5. Verify the whole certificate chain.

```
openssl verify -CAfile rootCA_cert.der -untrusted personalizationCA_cert.der
 deviceLeaf_cert.der
```

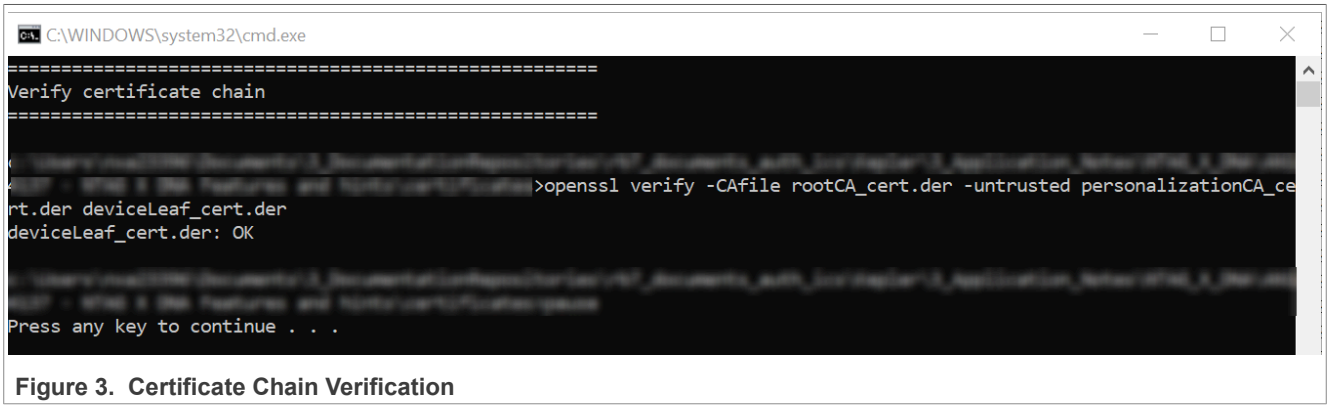

Figure 3. Certificate Chain Verification

3.3 ISO14443-4 PICC Activation

Reader supporting ISO14443-4 or NFC is required.

Table 3. ISO14443-4 PICC Activation

Step	ISO 14443 Command	NFC Forum Command		Data Message
1	RF field ON		=	true
2	REQA	SENS_REQ	>	26
3	ATQA		<	4403
4	Anticollision CL1	SDD_REQ CL1	>	9320
5	CT, UID0, UID1, UID2, BCC		<	8804168913
6	Select cascade level 1	SEL_REQ CL1	>	93708804168913
7	Data		<	04
8	Anticollision CL 2	SDD_REQ CL2	>	9520
9	UID0, UID1, UID2, BCC		<	AA5C5E8028
10	Select cascade level 2	SEL_REQ CL2	>	9570AA5C5E8028
11	SAK		<	20
12	RATS	RATS	>	E080
13	ATS		<	067777610280
14	PPS	ISO-DEP	>	D01100 (1)
15	PPSS		<	D0

(1) Higher speed data transfer can be set in this step (downlink and uplink).

3.4 (Optional when using I²C) Get CIP

This step is optional. It is available only on I²C interface. The CIP contains the communication interface parameters (i.e. both Physical Layer and Data Link Layer parameters) that the HD shall use to communicate with the NTAG X DNA, as well as Historical Bytes. This procedure may be used anytime (although rather expected to be used upon power on or after a software reset). More detailed info in ref.[10].

Table 4. Get CIP

Step	Command		Data Message
1	S(CIP request)	>	90 71 00 00 02 00 00 00
2	CIP	<	00 00 00 16 01 04 63 07 00 93 02 08 00 02 03 E8 00 01 00 64 04 03 E8 00 FE 00

Table 5. CIP parameters

Name	Length	Description	Value
PVER	1	Protocol Version: GP spec. definesversion ‘01’ oftheprotocol.	0x01
Length of IIN	1	Length of Issuer Identification Number	0x04
IIN	3 - 4	Issuer Identification Number (according to [7812-1], BCD encoded)	0x63070093

Table 5. CIP parameters*...continued*

Name	Length	Description	Value
PLD	1	Physical Layer ID: '01' for SPI / '02' for I2C	0x02
Length of PLP	1	Length of Physical Layer Parameters	0x08
Configuration	1	Characteristics supported by NTAG X DNA: b1 = 0: Clock stretching not supported b1 = 1: Clock stretching supported Other bits: RFU	0x00
PWT	1	Power Wake-up Time (ms)	0x02
MCF	2	Maximal Clock Frequency at which the SE may operate (in kHz)	0x03E8 (1 MHz)
PST	1	Power Saving Time-outs (in ms)	0x00
MPOT	1	Minimum Polling Time (conditional to Polling Mode support) (in ms)	0x01
RWGT	2	R/W Guard Time (in µs)	0x0064
Length of DLLP	1	Length of Data Link Layer Parameters	0x04
BWT	2	Block Waiting Time (in ms)	0x03E8 (ca. 1 sec)
IFSC	2	Maximum Information Field Size of the SE (in bytes) (i.e. initial value)	0x00FE (256 Bytes)
Length of HB	1	Length of Historical Bytes (Max. 32 bytes)	0x00
Historical Bytes	Var	Empty	-

3.5 ECC Originality Check

On the PICC level, is for Originality check purposes trust-provisioned by NXP in the secure environment within NXP production. It shall be used to verify authenticity of ICs delivered. Customers can also use their own scheme. The used key pair/cert is shared across all produced ICs and can be retrieved from https://www.gp-ca.nxp.com/CA/getCA?caid=63709320110001.

Note:

There are 2 different repositories to fetch NXP Signing certificates:

	Common Name	Purpose	Issuer Serial Number	
1	NXP Auth RootCAvE201	Pre-provisioned Application Certificate verification	63709320101001	https://www.gp-ca.nxp.com/CA/getCA?caid=63709320101001

	Common Name	Purpose	Issuer Serial Number	
2	**NXP Orig RootCAvE201**	ECC Originality Check	**63709320110001**	**https://www.gp-ca.nxp.com/CA/getCA?caid=63709320110001**

Further, ECC Originality check can also be disabled, to limit tracking (i.e. public key and UID can be freely read from NTAG X DNA's pre-provisioned certificate).

ECC Unilateral Authentication feature is used for the NXP ECC Originality Check purposes.

Note: *No need to select NDEF Application is needed as the NXP Originality certificate is on PICC level.*

Process flow is:

1. Read out pre-provisioned certificate from NTAG X DNA
2. Verify pre-provisioned certificate from NTAG X DNA with ECC Public Key fetched from NXP service (certificate can be trusted now)
3. Perform ECC Unilateral authentication
4. Verify Signature (got from step before) with Public Key from a pre-provisioned certificate from NTAG X DNA

Prerequisites: ECDSA Digital Signature Generation and Verification algorithm as defined in ref.[7].
SHA256 hashing algorithm as defined.
Random number generator (16 bytes).

Key used: ECC Public Key (Pub.CA.Orig = Pub.B). It can be fetched from https://www.gp-ca.nxp.com/CA/getCA?caid=63709320110001

Length [bytes]: 64

Algorithm: $ECDSA_{Verify}$(Pub.B, M, Sig.B)

Output:
- true (signature is valid)
- false

Table 6. ECC Originality Check

Step	Command		Data
1	Fetch NXP Device root certificate	=	curl https://www.gp-ca.nxp.com/CA/getCA?caid=63709320110001 ^ -o NXPOrigRootCAvE201.crt
2	Extract the ECC Public key from it	=	openssl x509 -in NXPOrigRootCAvE201.crt -text -noout
3	Originality CA Public key - Pub.CertOrig Key	=	36 23 95 20 48 F3 59 65 11 AD B4 4F 7A 85 23 77 E6 B1 40 3B 55 D3 86 5C 4B F4 A4 A2 2A 0A E0 18 87 B2 F8 DB E2 9D AC B9 95 74 F0 20 F1 17 82 86 53 05 B0 71 73 B8 71 0B C1 BE 15 33 FE 4C 51 70
	Read out pre-provisioned certificate Cert.Orig from NTAG X DNA		
4	Read certificate	>	90 AD 00 00 00 00 07 01 00 00 00 00 00 00 00 00
5	Cert.Orig [ASN1] + SW1 SW2	<	30 82 01 51 30 81 F7 A0 03 02 01 02 02 06 01 31 8C 95 80 18 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 46 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 4F 72 69 67 20 52 6F 6F 74 43 41 76 45 32 30 31 31 17 30 15 06 03 55 04 05 13 0E 36 33 37 30 39 33

Table 6. ECC Originality Check*...continued*

Step	Command		Data
			32 30 31 31 30 30 30 31 30 1E 17 0D 32 34 31 32 31 32 30 30 30 30 30 5A 17 0D 34 34 31 32 31 32 30 30 30 30 30 30 5A 30 19 31 17 30 15 06 03 55 04 0D 0C 0E 30 34 41 30 30 30 30 34 30 31 30 30 30 31 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 4C 7B 08 CE 90 E4 09 2C 81 64 94 CB 7E 1D 49 C7 A2 5E 49 7B FC CE 3A 99 82 31 73 46 52 43 A5 80 16 DA BF B0 BF 2A 88 6A 66 37 8E A4 BC 8B BF A5 3B 92 84 4B 51 B2 14 12 4F 26 55 99 A3 78 64 D0 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 49 00 30 46 02 21 00 87 1B 50 4A E4 29 2F 89 ED F3 26 DA CC 11 4F E8 3F E5 61 C0 62 D7 23 03 C4 DC A5 85 57 2F 40 80 02 21 00 B3 D3 23 91 84 46 22 A6 48 B9 0F 20 C0 8B 0F 30 2A D7 4C 9A D7 38 E4 25 41 BF 1E CA A8 7F 03 F6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 91 00
6	Get TBSCertificate SEQUENCE from Cert.Orig	=	30 81 F7 A0 03 02 01 02 02 06 01 31 8C 95 80 18 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 46 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 4F 72 69 67 20 52 6F 6F 74 43 41 76 45 32 30 31 31 17 30 15 06 03 55 04 05 13 0E 36 33 37 30 39 33 32 30 31 31 30 30 30 31 30 1E 17 0D 32 34 31 32 31 32 30 30 30 30 30 30 5A 17 0D 34 34 31 32 31 32 30 30 30 30 30 30 5A 30 19 31 17 30 15 06 03 55 04 0D 0C 0E 30 34 41 30 30 30 30 34 30 31 30 30 30 31 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 4C 7B 08 CE 90 E4 09 2C 81 64 94 CB 7E 1D 49 C7 A2 5E 49 7B FC CE 3A 99 82 31 73 46 52 43 A5 80 16 DA BF B0 BF 2A 88 6A 66 37 8E A4 BC 8B BF A5 3B 92 84 4B 51 B2 14 12 4F 26 55 99 A3 78 64 D0
7	SHA256(TBSCertificate SEQUENCE from Cert. Orig) - part which needs to be signed	=	0C 56 50 A2 E1 68 AC D6 59 6E A6 87 90 50 FE 79 DB 0C AD EB D6 35 7F EF 6B 7F 68 C3 AF 40 CF D3
8	Get public key (Pub.B) from Cert.Orig [ASN1] (without compression point 04)	=	4C 7B 08 CE 90 E4 09 2C 81 64 94 CB 7E 1D 49 C7 A2 5E 49 7B FC CE 3A 99 82 31 73 46 52 43 A5 80 16 DA BF B0 BF 2A 88 6A 66 37 8E A4 BC 8B BF A5 3B 92 84 4B 51 B2 14 12 4F 26 55 99 A3 78 64 D0
9	Get signature from Cert.Orig - Sig.Cert Orig	=	87 1B 50 4A E4 29 2F 89 ED F3 26 DA CC 11 4F E8 3F E5 61 C0 62 D7 23 03 C4 DC A5 85 57 2F 40 80 B3 D3 23 91 84 46 22 A6 48 B9 0F 20 C0 8B 0F 30 2A D7 4C 9A D7 38 E4 25 41 BF 1E CA A8 7F 03 F6
	Verify pre-provisioned certificate Cert. Orig		

Table 6. ECC Originality Check*...continued*

Step	Command		Data
10	$ECDSA_{Verify}$(Pub.CertOrigKey (step 3), SHA256(Cert. Orig) (step 7), Sig.CertOrig (step 9))	=	Signature valid.
	Perform ECC Unilateral authentication		
11	Generate RndA [16 bytes]	=	D5 00 5B F6 59 94 68 72 ED 63 44 AE 8D CB BD 4C
12	OptsA (optional) - whole TLV has to be included for SHA256 (step 18)	=	80 01 00
13	AuthDOHdr	=	7C 12
14	ISOInternalAuthenticate.Cmd	>	00 88 00 01 00 00 17 80 01 00 7C 12 81 10 D5 00 5B F6 59 94 68 72 ED 63 44 AE 8D CB BD 4C 00 00

Table 7. ISOInternalAuthenticate

CLA	CMD	P1	P2	Lc			OptsA			AuthDOHdr		RndA			Le	
00	88	00	01	00	00	17	80	01	00	7C	12	81	10	RndA	00	00

15	ISOInternalAuthenticate response [ASN1]	<	7C 54 81 10 D2 84 F4 42 60 C0 D4 36 04 B0 C0 2D 09 73 CA 9F 82 40 44 97 A5 14 B8 6E C6 1A 09 E1 7E 9D FD FD DD 38 B5 BE 1A F8 AA 5F 4D A9 E9 9F 49 12 38 C4 18 A1 B2 4C 83 50 1F C0 2B DC 2F CB 18 7C F5 B5 15 9A DD FE 60 B7 AC A1 DA 53 27 B6 D7 C4 13 B2 D9 2A 90 00
16	RndB	=	D2 84 F4 42 60 C0 D4 36 04 B0 C0 2D 09 73 CA 9F
17	Sig.B	=	44 97 A5 14 B8 6E C6 1A 09 E1 7E 9D FD FD DD 38 B5 BE 1A F8 AA 5F 4D A9 E9 9F 49 12 38 C4 18 A1 B2 4C 83 50 1F C0 2B DC 2F CB 18 7C F5 B5 15 9A DD FE 60 B7 AC A1 DA 53 27 B6 D7 C4 13 B2 D9 2A
	Verify Sig.B with Public Key from certificate (Pub.B)		
18	0xF0F0[||OptsA]||RndB||RndA		F0 F0 80 01 00 D2 84 F4 42 60 C0 D4 36 04 B0 C0 2D 09 73 CA 9F D5 00 5B F6 59 94 68 72 ED 63 44 AE 8D CB BD 4C
19	SHA256(0xF0F0[||OptsA]||RndB||RndA)	=	E2 08 30 28 E0 78 F1 DC F7 B4 90 AB 9D 16 10 A4 09 EE 4A 29 14 F4 D1 E0 91 02 37 BB 51 69 C2 C8
20	$ECDSA_{Verify}$(Pub.B, M, Sig.B)	=	Signature valid.

Middleware example

Below CMAKE options shall be set:

```
NXMW_Auth:STRING=None
NXMW_HostCrypto:STRING=MBEDTLS
NXMW_NX_Type:STRING=NX_PICC
NXMW_Secure_Tunneling:STRING=None
```

Example name: "ex_originality_check"

3.6 ISO SELECT NDEF application using DF Name

Table 8. Select NDEF Application using Cmd.ISOSelect

Step	Command		Data Message
1	ISO7816 AID – DF Application Name	=	D2 76 00 00 85 01 01
2	CLA	=	00
3	INS	=	A4
4	P1	=	04 (select by DF name)
5	P2	=	0C
6	Lc	=	07
7	Command header	=	00 A4 04 0C 07
8	Command data (ISO7816 AID – DF Name)	=	D2 76 00 00 85 01 01
9	Le	=	00
10	Cmd.Select C-APDU	>	00 A4 04 0C 07 D2 76 00 00 85 01 01 00
11	R-APDU	<	90 00

3.7 Symmetric mutual authentication - AuthenticateEV2First with key 0x00

Table 9. Cmd.AuthenticateEV2First using Key No 0x00

Step	Command		Data Message
1	Key No	=	00
2	Key0 value	=	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3	Cmd (INS)	=	71
4	pcdCap2Len		06
5	PCDCap2		00 00 00 00 00 00
6	Command header (Key No || LenCap)	=	0000
7	Cmd.AuthenticateFirst C-APDU	>	90 71 00 00 08 00 06 00 00 00 00 00 00 00
8	R-APDU (E(K_0, RndB) || Response Code)	<	3F 35 CC 1F 2E EE 2A 05 6A AE B1 F4 D3 0D 2B E6 91 AF
9	E(K_0, RndB)	=	3F 35 CC 1F 2E EE 2A 05 6A AE B1 F4 D3 0D 2B E6
10	Response Code (SW1 SW2)	=	91 AF (AF means additional frame)
11	D(K_0, RndB)	=	BF A4 80 0A D8 52 37 06 BE BC 84 3C BE 7A 6A AE
12	PCD generates RndA	=	95 E7 32 33 4D AB 8A F6 21 FC 6C B8 BA 13 2D 56
13	PCD prepares RndB' (rotate left by 1 byte)	=	A4 80 0A D8 52 37 06 BE BC 84 3C BE 7A 6A AE BF
14	RndA || RndB'	=	95 E7 32 33 4D AB 8A F6 21 FC 6C B8 BA 13 2D 56 A4 80 0A D8 52 37 06 BE BC 84 3C BE 7A 6A AE BF
15	E(K_0, RndA || RndB')	=	90 AF 1A 75 B3 4E 04 0C FF A3 98 F5 B3 12 0F 52 63 6A 49 1E E7 9F 8A 43 77 93 BD 30 4D A0 75 47
16	Cmd.AuthenticatePart2 C-APDU (INS = AF)	>	90 AF 00 00 20 90 AF 1A 75 B3 4E 04 0C FF A3 98 F5 B3 12 0F 52 63 6A 49 1E E7 9F 8A 43 77 93 BD 30 4D A0 75 47 00
17	R-APDU E(Kx, TI || RndA' || PDcap2 || PCDcap2)	<	15 3E 71 26 1C 15 88 04 B2 2D 80 1B 91 9C AA 2A 26 E5 2F 27 DC 2B 2E F9 13 53 96 B0 E0 C6 5D 67 91 00
18	E(Kx, TI || RndA' || PDcap2 || PCDcap2)	=	15 3E 71 26 1C 15 88 04 B2 2D 80 1B 91 9C AA 2A 26 E5 2F 27 DC 2B 2E F9 13 53 96 B0 E0 C6 5D 67
19	Response Code (SW1 SW2)		91 00
20	D(K_0 , TI || RndA´ || PDcap2 || PCDcap2)	=	70 5E 70 FB E7 32 33 4D AB 8A F6 21 FC 6C B8 BA 13 2D 56 95 00 00 00 00 00 00 00 00 00 00 00 00
21	TI (4 byte)	=	70 5E 70 FB
22	RndA´ (16 byte)	=	E7 32 33 4D AB 8A F6 21 FC 6C B8 BA 13 2D 56 95
23	PDcap2 (6 byte)	=	00 00 00 00 00 00
24	PCDcap2 (6 byte)	=	00 00 00 00 00 00
25	RndA (rotate right for 1 byte)	=	95 E7 32 33 4D AB 8A F6 21 FC 6C B8 BA 13 2D 56
26	PCD compares sent RndA (from step 12) and received RndA (from step 25)	=	95 E7 32 33 4D AB 8A F6 21 FC 6C B8 BA 13 2D 56 == 95 E7 32 33 4D AB 8A F6 21 FC 6C B8 BA 13 2D 56

Table 9. Cmd.AuthenticateEV2First using Key No 0x00*...continued*

Step	Command		Data Message
27	SV 1 = [0xA5][0x5A][0x00][0x01][0x00] [0x80][RndA[15:14] || [(RndA[13:8] ⊕ Rnd B[15:10])] || [RndB[9:0] || RndA[7:0]	=	A5 5A 00 01 00 80 95 E7 8D 97 CD A1 52 A4 37 06 BE BC 84 3C BE 7A 6A AE 21 FC 6C B8 BA 13 2D 56
28	SV 2 = [0x5A][0xA5][0x00][0x01][0x00] [0x80][RndA[15:14] || [(RndA[13:8] ⊕ Rnd B[15:10])] || [RndB[9:0] || RndA[7:0]	=	5A A5 00 01 00 80 95 E7 8D 97 CD A1 52 A4 37 06 BE BC 84 3C BE 7A 6A AE 21 FC 6C B8 BA 13 2D 56
29	Encryption Session Key ($K_{SesAuthENC}$) = CMAC (K_0, SV1)	=	18 F7 7C 75 A1 0D 55 75 5F B3 73 B1 BB CF 52 64
30	CMAC Session Key ($K_{SesAuthMAC}$) = CMAC(K_0, SV2)	=	7D 97 4D 25 C6 07 81 88 92 EE 9E 7C 39 C9 07 16

3.8 ECC Key Management

3.8.1 Read "Host root certificate"

In this step, the Public key from the Host Root certificate will be retrieved and programmed to NTAG X DNA key slot.

Using certificate/key with X509 formatting.

Named: host_root_certificate.der

```
X509 Certificate:
Version: 3
Serial Number: 794d41862cdced1178789e53a211b6f84bc6059b
Signature Algorithm:
    Algorithm ObjectId: 1.2.840.10045.4.3.2 sha256ECDSA
    Algorithm Parameters: NULL
Issuer:
    CN=NXP Auth RootCAvE201
    O=NXP
    L=Eindhoven
    S=Eindhoven
    C=NL
  Name Hash(sha1): 96722d219aca9e678d7cdbe930852172d33045ec
  Name Hash(md5): fb0369a3aebbb29dfb7aef76764f709e

 NotBefore: 4. 06. 2024 07:25
 NotAfter: 2. 06. 2034 07:25

Subject:
    CN=NXP Auth RootCAvE201
    O=NXP
    L=Eindhoven
    S=Eindhoven
    C=NL
  Name Hash(sha1): 96722d219aca9e678d7cdbe930852172d33045ec
  Name Hash(md5): fb0369a3aebbb29dfb7aef76764f709e
```

```
Public Key Algorithm:
    Algorithm ObjectId: 1.2.840.10045.2.1 ECC
    Algorithm Parameters:
    06 08 2a 86 48 ce 3d 03  01 07
        1.2.840.10045.3.1.7 ECDSA_P256 (x962P256v1)
Public Key Length: 256 bits
Public Key: UnusedBits = 0
    0000  04 61 fa 9c da af c4 c8  9c 2e 1c 0c e9 87 28 0a
    0010  b5 dd 5e 2e 89 b8 e2 70  e3 bf b3 28 79 75 b2 58
    0020  72 2b b8 1d 0f 16 c7 4d  b1 c2 23 2e 4e b5 ca ce
    0030  b4 56 9f 52 47 18 14 7b  88 e0 5f e3 03 6f 4b c9
    0040  6c
Certificate Extensions: 1
    2.5.29.19: Flags = 0, Length = 5
    Basic Constraints
        Subject Type=CA
        Path Length Constraint=None

Signature Algorithm:
    Algorithm ObjectId: 1.2.840.10045.4.3.2 sha256ECDSA
    Algorithm Parameters: NULL
Signature: UnusedBits=0
    0000  16 ce de 6b fc 06 8e e4  7d 3b c9 e7 c5 8a c3 e0
    0010  2d 80 df 05 19 1c ce b3  1e e0 a0 64 10 cc b2 93
    0020  00 21 02 73 64 4e c4 e3  12 56 a0 6f 83 9b 96 de
    0030  65 13 64 30 e0 25 ad 6a  2f 9e b7 2f 7f a6 0b 21
    0040  36 f6 e6 00 21 02 46 30
Signature matches Public Key
Root Certificate: Subject matches Issuer
Key Id Hash(rfc-sha1): bbe9aae1d5eccabcf3a999b1f6cd084b1bb36896
Key Id Hash(sha1): e0845774dcd6b7c32e4b381515bbb0813e19364d
Key Id Hash(bcrypt-sha1): 4171d139659da2b9b37f582d6cd68fcd95386ff4
Key Id Hash(bcrypt-sha256):
 e866dee63069cf330acd33fe7b776ef2f1e18957d151726e366539ce44b5e451
Key Id Hash(md5): b5e41bed6c9d1b3fc4004c13ee796111
Key Id Hash(sha256):
 f48af46d5e467a2c8834284521312a0b56c86e0488831fd2452f81bf4d2d10a8
Key Id Hash(pin-sha256): YSaWf2SPmsTN5ZSmOvDGKcAKoImqAJjz6fbfFXTljIs=
Key Id Hash(pin-sha256-hex):
 6126967f648f9ac4cde594a63af0c629c00aa089aa0098f3e9f6df1574e58c8b
Cert Hash(md5): e040cf3a0945703b52c09f3a9fa0a23d
Cert Hash(sha1): 1b376c5330de0b728bab3dc8b6df713a64b08d15
Cert Hash(sha256):
 c53a594c9bebc3a95c107664349fdc22f296738af85957bb83bb0a95fb8f5fad
Signature Hash: a2b0668180e0421319a0131db2479cbb46c41b10713de238fb92f64d794688f9
```

Extracted public key (with compression point 04h) which will be programmed as CARootKey:

```
04 61 fa 9c da af c4 c8 9c 2e 1c 0c e9 87 28 0a b5 dd 5e 2e 89 b8 e2 70 e3 bf b3
 28 79 75 b2 58 72 2b b8 1d 0f 16 c7 4d b1 c2 23 2e 4e b5 ca ce b4 56 9f 52 47
 18 14 6c 7b 88 e0 5f e3 03 6f 4b c9
```

Optionally, issuer's metadata can be inserted as well. We will use the issuer's part of the certificate.

```
SEQ(30) LEN=0x62{
    SET(31) LEN=0xB{
        SEQ(30) LEN=0x9{
            OID(06) LEN=0x3 VAL=countryName
```

```
            String(13) LEN=0x2 VAL=NL
        };
    };
    SET(31) LEN=0x12{
        SEQ(30) LEN=0x10{
            OID(06) LEN=0x3 VAL=stateOrProvinceName
            UTF8(0C) LEN=0x9 VAL=Eindhoven
        };
    };
    SET(31) LEN=0x12{
        SEQ(30) LEN=0x10{
            OID(06) LEN=0x3 VAL=localityName
            UTF8(0C) LEN=0x9 VAL=Eindhoven
        };
    };
    SET(31) LEN=0xC{
        SEQ(30) LEN=0xA{
            OID(06) LEN=0x3 VAL=organizationName
            UTF8(0C) LEN=0x3 VAL=NXP
        };
    };
    SET(31) LEN=0x1D{
        SEQ(30) LEN=0x1B{
            OID(06) LEN=0x3 VAL=commonName
            UTF8(0C) LEN=0x14 VAL=NXP Auth RootCAvE201
        };
    };
};
```

Which is in ASN1 coding equal to:

```
30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45
 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76
 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C
 14 4E 58 50 20 41 75 74 68 20 52 6F 6F 74 43 41 76 45 32 30 31
```

3.8.2 GetKeySettings - CARootKeyList

Retrieve the metadata of certain key type (CARootKeys)

Table 10. GetKeySettings - CARootKeyList

Step	Command		Data
1	Cmd (INS) of GetKeySettings	=	45
2	CommMode	=	CommMode.MAC
3	CmdHeader (Option for CARootKeys metadata)	=	02
4	$K_{SesAuthMAC}$	=	7D 97 4D 25 C6 07 81 88 92 EE 9E 7C 39 C9 07 16
5	CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData]	=	45 00 00 70 5E 70 FB 02
6	MAC($K_{SesAuthMAC}$; CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData])	=	69 33 F3 29 D0 6E B5 EC E9 B1 8F 50 BD EE EF 1B

Table 10. GetKeySettings - CARootKeyList*...continued*

Step	Command		Data
7	MACt	=	33 29 6E EC B1 50 EE 1B
8	APDU	>	90 45 00 00 09 02 33 29 6E EC B1 50 EE 1B 00
9	R-APDU	<	01 00 0C FF 3F 30 3F 00 DA 92 30 3C 4E C1 71 5A 91 00
10	PICC's MAC	=	DA 92 30 3C 4E C1 71 5A
11	Response	=	01 00 0C FF 3F 30 3F 00
12	SW1 SW2	=	91 00
13	Verify Response MAC RC (SW2) || CmdCtr++ || TI [|| RespData]	=	00 01 00 70 5E 70 FB 01 00 0C FF 3F 30 3F 00
14	MAC($K_{SesAuthMAC}$; RC || CmdCtr || TI [|| RespData])	=	63 DA CA 92 6B 30 33 3C 23 4E D4 C1 D7 71 D9 5A
15	MACt	=	DA 92 30 3C 4E C1 71 5A
16	Compare MACt-s (from step 10 and step 15)	=	DA 92 30 3C 4E C1 71 5A == DA 92 30 3C 4E C1 71 5A

Table 11. CARootKey's metadata (from step 11)

Name	Length	Value	Description
EntryCount	1	01	Number of entries
CARootKeyList	[n * 7]		
List with metadata			
- Entry[0] KeyNo	1	00	Key slot number
- Entry[1] CurveID	1	0C (nist-p256)	CurveID
- Entry[2..3] AccessRights	2	FF 3F	Access conditions that can be granted when authenticated with this CARootKey are encoded in this bitmap, refer to chapter CARootKey access rights in ref.[12]
- Entry[4] WriteAccess	1	30 (CommMode.FULL, WriteAR = 0x00)	Defines the CommMode and access rights required to update the key with Manage CARootKey.
- Entry[5] Reserved	1	3F	-
- Entry[6] Reserved	1	00	-

3.8.3 Program the "Host root public key" to the IC - ManageCARootKey

Insert CARootKeys from Section 3.8.1 into ECC key slot 1.

Table 12. ManageCARootKey - Write CARootKey into ECC key slot

Step	Command		Data
1	Cmd (INS) of Cmd.ManageCARootKey	=	48
2	CommMode	=	CommMode.Full (default)
3	KeyNo	=	00
4	curveID	=	0C
5	AccessRights	=	3FFF
6	WriteAccess	=	30
7	Reserved	=	3F
8	Reserved	=	00
9	PublicKey	=	04 61 FA 9C DA AF C4 C8 9C 2E 1C 0C E9 87 28 0A B5 DD 5E 2E 89 B8 E2 70 E3 BF B3 28 79 75 B2 58 72 2B B8 1D 0F 16 C7 4D B1 C2 23 2E 4E B5 CA CE B4 56 9F 52 47 18 14 7B 88 E0 5F E3 03 6F 4B C9 6C
10	IssuerLen	=	64
11	Issuer (in ASN1 coding from step Section 3.8.1)	=	30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 41 75 74 68 20 52 6F 6F 74 43 41 76 45 32 30 31
12	CmdData (PublicKey || IssuerLen || Issuer || Padding (80 .. ..))	=	04 61 FA 9C DA AF C4 C8 9C 2E 1C 0C E9 87 28 0A B5 DD 5E 2E 89 B8 E2 70 E3 BF B3 28 79 75 B2 58 72 2B B8 1D 0F 16 C7 4D B1 C2 23 2E 4E B5 CA CE B4 56 9F 52 47 18 14 7B 88 E0 5F E3 03 6F 4B C9 6C 64 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 41 75 74 68 20 52 6F 6F 74 43 41 76 45 32 30 31 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
13	E($K_{SesAuthENC}$, CmdData)	=	AA 03 B7 09 3B D7 16 C2 5F 9A 92 98 97 DF 4A 90 8D 2D AA 37 C0 7D AD 23 07 A4 9D C3 97 CF 54 9A B5 F4 87 25 7D F6 49 D7 38 45 B1 8C 35 8F 2C E3 E2 36 0F 3F 0B 3D E6 C8 72 37 97 1F A0 17 FE DF C8 84 6F 5C B6 99 3A 11 53 31 25 A4 32 EE 31 79 18 4E 38 A0 24 D9 28 5F BE A8 1C 29 F5 F8 FE 11 72 30 75 97 92 E3 6C 15 C2 28 32 BD 7C C0 39 68 34 23 B3 D8 2B 2E AC 4E E9 22 DF C7 FF 03 96 EF

Table 12. ManageCARootKey - Write CARootKey into ECC key slot*...continued*

Step	Command		Data
			62 EF E4 F5 B0 37 FD 94 21 9F C3 01 E2 B8 51 B7 82 3B CD 2E B2 1E A9 4F D9 C1 D8 8C 85 E5 05 06 E7 8B 2C 0A 1A 57 51 1C C6 3B D9 3E 65 A8 B8 63
14	CMD || CmdCtr || TI [|| CmdHeader] [|| $E(K_{SesAuthENC}, CmdData)$]	=	48 01 00 70 5E 70 FB 00 0C FF 3F 30 3F 00 AA 03 B7 09 3B D7 16 C2 5F 9A 92 98 97 DF 4A 90 8D 2D AA 37 C0 7D AD 23 07 A4 9D C3 97 CF 54 9A B5 F4 87 25 7D F6 49 D7 38 45 B1 8C 35 8F 2C E3 E2 36 0F 3F 0B 3D E6 C8 72 37 97 1F A0 17 FE DF C8 84 6F 5C B6 99 3A 11 53 31 25 A4 32 EE 31 79 18 4E 38 A0 24 D9 28 5F BE A8 1C 29 F5 F8 FE 11 72 30 75 97 92 E3 6C 15 C2 28 32 BD 7C C0 39 68 34 23 B3 D8 2B 2E AC 4E E9 22 DF C7 FF 03 96 EF 62 EF E4 F5 B0 37 FD 94 21 9F C3 01 E2 B8 51 B7 82 3B CD 2E B2 1E A9 4F D9 C1 D8 8C 85 E5 05 06 E7 8B 2C 0A 1A 57 51 1C C6 3B D9 3E 65 A8 B8 63
15	$MAC(K_{SesAuthMAC}$; CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData])	=	56 38 79 37 8B 46 64 3B C7 40 A0 2F 1F 99 25 58
16	MACt	=	38 37 46 3B 40 2F 99 58
17	C-APDU	>	90 48 00 00 BF 00 0C FF 3F 30 3F 00 AA 03 B7 09 3B D7 16 C2 5F 9A 92 98 97 DF 4A 90 8D 2D AA 37 C0 7D AD 23 07 A4 9D C3 97 CF 54 9A B5 F4 87 25 7D F6 49 D7 38 45 B1 8C 35 8F 2C E3 E2 36 0F 3F 0B 3D E6 C8 72 37 97 1F A0 17 FE DF C8 84 6F 5C B6 99 3A 11 53 31 25 A4 32 EE 31 79 18 4E 38 A0 24 D9 28 5F BE A8 1C 29 F5 F8 FE 11 72 30 75 97 92 E3 6C 15 C2 28 32 BD 7C C0 39 68 34 23 B3 D8 2B 2E AC 4E E9 22 DF C7 FF 03 96 EF 62 EF E4 F5 B0 37 FD 94 21 9F C3 01 E2 B8 51 B7 82 3B CD 2E B2 1E A9 4F D9 C1 D8 8C 85 E5 05 06 E7 8B 2C 0A 1A 57 51 1C C6 3B D9 3E 65 A8 B8 63 38 37 46 3B 40 2F 99 58 00
18	R-APDU (MACt + SW1 SW2)	<	C1 0C 73 DE EE 41 49 8B 91 00
19	Verify Response MAC RC (SW2) || CmdCtr++ || TI [|| RespData]	=	00 02 00 70 5E 70 FB
20	$MAC(K_{SesAuthMAC}$; RC || CmdCtr || TI [|| RespData])	=	A1 C1 16 0C AB 73 66 DE F5 EE 0C 41 04 49 1F 8B
21	MACt	=	C1 0C 73 DE EE 41 49 8B
22	Compare MACt-s (from step 18 and step 20)	=	C1 0C 73 DE EE 41 49 8B == C1 0C 73 DE EE 41 49 8B

3.8.4 GetKeySettings of ECCPrivateKeyList

Retrieve the meta-data of certain key type (ECCPrivateKeys)

Table 13. GetKeySettings - CARootKeyList

Step	Command		Data
1	Cmd (INS) of Cmd.GetKeySettings	=	45
2	CommMode	=	CommMode.MAC
3	CmdHeader (Option for ECCPrivateKey meta-data)	=	01
4	$K_{SesAuthMAC}$	=	7D 97 4D 25 C6 07 81 88 92 EE 9E 7C 39 C9 07 16
5	CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData]	=	45 02 00 70 5E 70 FB 01
6	MAC($K_{SesAuthMAC}$; CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData])	=	73 6E 5A 82 A2 F8 F6 10 4D C2 32 90 F1 6E 23 00
7	MACt	=	6E 82 F8 10 C2 90 6E 00
8	C-APDU	>	90 45 00 00 09 01 6E 82 F8 10 C2 90 6E 00 00
9	R-APDU	<	02 00 0C 04 00 30 00 00 00 00 00 00 00 00 02 0C 30 00 30 00 00 00 00 00 00 00 00 16 E7 64 23 FB 45 7B 26 91 00
10	PICC's MAC	=	16 E7 64 23 FB 45 7B 26
11	Response	=	02 00 0C 04 00 30 00 00 00 00 00 00 00 00 02 0C 30 00 30 00 00 00 00 00 00 00 00
12	SW1 SW2	=	91 00
13	Verify Response MAC RC (SW2) || CmdCtr++ || TI [|| RespData]	=	00 03 00 70 5E 70 FB 02 00 0C 04 00 30 00 00 00 00 00 00 00 00 02 0C 30 00 30 00 00 00 00 00 00 00 00 00
14	MAC($K_{SesAuthMAC}$; RC || CmdCtr || TI [|| RespData])	=	D1 16 E5 E7 74 64 EF 23 7D FB DB 45 62 7B 49 26
15	MACt	=	16 E7 64 23 FB 45 7B 26
16	Compare MACt-s (from step 10 and step 15)	=	16 E7 64 23 FB 45 7B 26 == 16 E7 64 23 FB 45 7B 26

Table 14. CARootKey's meta-data (from step 11)

Name	Length	Value	Description
EntryCount	1	02	Number of entries
CARootKeyList	[n * 13]		First list (out of 2)
List with meta-data			
- Entry[0] KeyNo	1	00	Key slot number

Table 14. CARootKey's meta-data (from step 11)*...continued*

Name	Length	Value	Description
- Entry[1] CurveID	1	0C (nist-p256)	CurveID
- Entry[2..3] KeyPolicy	2	04 00	Defines the allowed crypto operations with the targeted key. Bit3 is set - this key can be used for SIGMA-I Mutual Authentication.
- Entry[4] WriteAccess	1	30 (CommMode.FULL, WriteAR = 0x00)	Defines the CommMode and access rights required to update the key with Manage CARootKey.
- Entry[5...8] KeyUsageCtr Limit	1	00 00 00 00	-
- Entry[9...13] KeyUsageCtr	1	00 00 00 00	-
CARootKeyList	[n * 13]		Second list (out of 2)
List with meta-data			
- Entry[0] KeyNo	1	02	Key slot number
- Entry[1] CurveID	1	0C (nist-p256)	CurveID
- Entry[2..3] KeyPolicy	2	30 00	Defines the allowed crypto operations with the targeted key. Bit3 is set - this key can be used for SIGMA-I Mutual Authentication.
- Entry[4] WriteAccess	1	30 (CommMode.FULL, WriteAR = 0x00)	Defines the CommMode and access rights required to update the key with Manage CARootKey.
- Entry[5...8] KeyUsageCtr Limit	1	00 00 00 00	-
- Entry[9...13] KeyUsageCtr	1	00 00 00 00	-

3.8.5 Program the "Device leaf private key" to the IC - ECC Key Management

This step inserts Private Key from a pre-generated certificate. It is highly advised that Personalization process is designed that NTAG X DNA generates a ECC key pair. This way, the private key stays in the IC and cannot be retrieved anymore.

Table 15. ManageKeyPair - Write Device leaf private key into ECC key slot

Step	Command		Data
1	Cmd (INS) of GetKeySettings	=	46
2	CommMode	=	CommMode.Full (default)
3	KeyNo	=	00
4	Option	=	01
5	CurveID	=	0C
6	KeyPolicy(LSB)	=	04 00

Table 15. ManageKeyPair - Write Device leaf private key into ECC key slot*...continued*

Step	Command		Data
7	writeAccessCond	=	30
8	KUCLimit	=	00
9	PrivateKey	=	23 09 F1 6F D6 F2 3F 63 73 89 B1 60 7D B0 E0 3E 5D 50 14 ED EC 09 B0 BF E8 78 DA B2 76 72 01 39
10	CmdHeader (KeyNo || Option || CurveID || KeyPolicy || WriteAccess || KUCLimit)	=	00 01 0C 04 00 30 00 00 00 00
11	CmdData (PrivateKey || Padding (80 .. ..))	=	23 09 F1 6F D6 F2 3F 63 73 89 B1 60 7D B0 E0 3E 5D 50 14 ED EC 09 B0 BF E8 78 DA B2 76 72 01 39 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
12	A55A || TI || CmdCtr || 0000000000000000	=	A5 5A 70 5E 70 FB 03 00 00 00 00 00 00 00 00 00
13	IVc = E($K_{SesAuthENC}$, A55A || TI || CmdCtr || 0000000000000000)	=	20 6F B6 20 48 59 15 61 6D AC 0B C0 36 44 74 37
14	E($K_{SesAuthENC}$, IVc, CmdData)	=	56 71 BC EA 1E D1 38 E9 B8 7F 00 7D 71 6C 81 93 EF 5B 07 A0 81 08 03 53 8D 44 1F 9A 98 1A 95 35 6D 93 A2 A6 67 00 67 EA 34 7C 26 64 DE 8B 21 F4
15	CMD || CmdCtr || TI [|| CmdHeader] [|| E($K_{SesAuthENC}$, IVc, CmdData)]	=	46 03 00 70 5E 70 FB 00 01 0C 04 00 30 00 00 00 00 56 71 BC EA 1E D1 38 E9 B8 7F 00 7D 71 6C 81 93 EF 5B 07 A0 81 08 03 53 8D 44 1F 9A 98 1A 95 35 6D 93 A2 A6 67 00 67 EA 34 7C 26 64 DE 8B 21 F4
16	MAC($K_{SesAuthMAC}$; CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData])	=	73 C2 A2 E9 8A 9D F5 A0 CA 84 20 37 C0 A8 43 82
17	MACt	=	C2 E9 9D A0 84 37 A8 82
18	C-APDU	>	90 46 00 00 42 00 01 0C 04 00 30 00 00 00 00 56 71 BC EA 1E D1 38 E9 B8 7F 00 7D 71 6C 81 93 EF 5B 07 A0 81 08 03 53 8D 44 1F 9A 98 1A 95 35 6D 93 A2 A6 67 00 67 EA 34 7C 26 64 DE 8B 21 F4 C2 E9 9D A0 84 37 A8 82 00
19	R-APDU (MACt + SW1 SW2)	<	7F CA 53 2F BD 7E A3 60 91 00
20	Verify Response MAC RC (SW2) || CmdCtr++ || TI [|| RespData]	=	00 04 00 70 5E 70 FB
21	MAC($K_{SesAuthMAC}$; RC || CmdCtr || TI [|| RespData])	=	D2 7F 2B CA 48 53 AE 2F E7 BD 95 7E AC A3 FF 60
22	MACt	=	7F CA 53 2F BD 7E A3 60
23	Compare MACt-s (from step 19 and step 22)	=	7F CA 53 2F BD 7E A3 60 == 7F CA 53 2F BD 7E A3 60

3.9 Certificate Management

3.9.1 GetConfiguration - Cmd.CertificateManagement

Get info about certificates cache sizes, cert. slots.

Table 16. GetConfiguration - Cmd.CertificateManagement

Step	Command		Data
1	Cmd (INS) of CertificateManagement	=	65
2	CommMode	=	CommMode.Full (default)
3	CmdHeader (Option)	=	13
4	CMD || CmdCtr || TI [|| CmdHeader] [|| $E(K_{SesAuthENC}$, IVc, CmdData)]	=	65 04 00 70 5E 70 FB 13
5	MAC($K_{SesAuthMAC}$; CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData])	=	AD 54 F4 61 2E AA 71 54 3A C4 C2 7C 96 B2 DC E9
6	MACt	=	54 61 AA 54 C4 7C B2 E9
7	C-APDU	>	90 65 00 00 09 13 54 61 AA 54 C4 7C B2 E9 00
8	R-APDU (Data + MACt + SW1 SW2)	<	B4 CB A3 D3 AF F6 7B FA 02 52 09 35 C8 6C 2C F9 6A 61 0A DD 55 73 C6 20 91 00
9	Verify Response MAC RC (SW2) || CmdCtr++ || TI [|| RespData]	=	00 05 00 70 5E 70 FB B4 CB A3 D3 AF F6 7B FA 02 52 09 35 C8 6C 2C F9
10	MAC($K_{SesAuthMAC}$; RC || CmdCtr || TI [|| RespData])	=	E4 6A F2 61 EF 0A F7 DD 87 55 A0 73 DB C6 69 20
11	MACt	=	6A 61 0A DD 55 73 C6 20
12	Compare MACt-s (from step 8 and step 11)	=	6A 61 0A DD 55 73 C6 20 == 6A 61 0A DD 55 73 C6 20
13	5AA5 || TI || CmdCtr || 0000000000000000	=	5A A5 70 5E 70 FB 05 00 00 00 00 00 00 00 00 00
14	IVr = E($K_{SesAuthENC}$, 5AA5 || TI || CmdCtr || 0000000000000000)	=	3F 24 2C E7 9C 26 A7 56 E8 EA 0F 42 95 8C 30 CD
15	D($K_{SesAuthENC}$, IVr, RspData)	=	05 04 08 10 80 00 00 00 00 00 00 00 00 00 00 00
16	Remove padding	=	05 04 08 10

Table 17. CertificateManagement meta-data (from step 16)

Name	Length	Value	Description
LeafCacheSize	1	05	5 slots available for end leaf certificates cache
IntermCacheSize	1	04	4 slots available for intermediate certificates cache
FeatureSelection	1	08 (0000 1000b)	
- Bit 4-3: HostCertificate Support	1	01b	support full host certificates
- Bit 2-1: InternalCertificate Support	1	00b	repository default (use certs as stored in the certificate repo)
- Bit 0: EnableSIGMA-I cache	2	0b	disabled (default)
ManageCert Repo- Access Condition	1	10	

Table 17. CertificateManagement meta-data (from step 16)*...continued*

Name	Length	Value	Description
- Bit 5-4: CommMode	1	0001b	CommMode.MAC
- Bit 3-0: AccessCondition	1	0000b	Access conditions for key 0x00

3.9.2 Create certificate repository ManageCertRepo - CreateCertRepo

Table 18. ManageCertRepo - Create certificate repository

Step	Command		Data
1	Cmd (INS) of GetKeySettings	=	49
2	CommMode	=	CommMode.MAC (as from step Table 17)
3	action (Create certificate repository)	=	00
4	repoID	=	01
5	Private Key Id	=	00
6	Repository Size (LSB) (3072 Bytes)	=	00 0C
7	Certificate Repository Write/Reset Access	=	30
8	Certificate Repository Read Command Access	=	30
9	CmdHeader (action || repoID || Private Key Id || Repository Size (LSB) || Write/ Reset Access || Read Command Access)	=	00 01 00 00 0C 30 30
10	CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData]	=	49 05 00 70 5E 70 FB 00 01 00 00 0C 30 30
11	MAC($K_{SesAuthMAC}$; CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData])	=	C9 7A F0 A6 07 B9 E4 33 81 F7 EF D7 C8 9F FE 3E
12	MACt	=	7A A6 B9 33 F7 D7 9F 3E
13	C-APDU	>	90 49 00 00 0F 00 01 00 00 0C 30 30 7A A6 B9 33 F7 D7 9F 3E 00
14	R-APDU (MACt + SW1 SW2)	<	46 A8 C2 8F AF 71 09 B1 91 00
15	Verify Response MAC RC (SW2) || CmdCtr++ || TI [|| RespData]		00 06 00 70 5E 70 FB
16	MAC($K_{SesAuthMAC}$; RC || CmdCtr || TI [|| RespData])	=	C4 46 58 A8 72 C2 1F 8F D2 AF DC 71 80 09 1A B1
17	MACt	=	46 A8 C2 8F AF 71 09 B1
18	Compare MACt-s (from step 14 and step 17)		46 A8 C2 8F AF 71 09 B1 == 46 A8 C2 8F AF 71 09 B1

3.9.3 Read certificate repository from the IC - ReadCertRepo Metadata

Table 19. Cmd.ReadCertRepo

Step	Command		Data
1	Cmd (INS) of ReadCertRepo	=	4A
2	CommMode	=	CommMode.MAC (if reading meta data only. If reading certificate repository directly, access right defined at creation of repository are needed to be set)
3	CmdHeader (RepoID || Data Item)	=	01 FF
4	CMD || CmdCtr || TI [|| CmdHeader] [|| E($K_{SesAuthENC}$, IVc, CmdData)]	=	4A 06 00 70 5E 70 FB 01 FF
5	MAC($K_{SesAuthMAC}$; CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData])	=	2F BC DC 52 20 7A 8A 61 08 1A 13 D0 CA 80 AB 53
6	MACt	=	BC 52 7A 61 1A D0 80 53
7	C-APDU	>	90 4A 00 00 0A 01 FF BC 52 7A 61 1A D0 80 53 00
8	R-APDU (Data + MACt + SW1 SW2)	<	00 00 0C 30 30 22 C4 A2 B4 4C 3A D8 07 91 00
9	Verify Response MAC RC (SW2) || CmdCtr++ || TI [|| RespData]	=	00 07 00 70 5E 70 FB 00 00 0C 30 30
10	MAC($K_{SesAuthMAC}$; RC || CmdCtr || TI [|| RespData])	=	83 22 8F C4 BB A2 01 B4 4B 4C F6 3A BB D8 CB 07
11	MACt	=	22 C4 A2 B4 4C 3A D8 07
12	Compare MACt-s (from step 8 and step 11)	=	22 C4 A2 B4 4C 3A D8 07 == 22 C4 A2 B4 4C 3A D8 07

Table 20. Cmd.ReadCertRepo - Response Data Format for Metadata

Name	Length	Value	Description
Private Key Id	1	00	Id of ECC private key associated with this repository (key must have been created using Manage KeyPair).
Repository Size (LSB) (0C 00h = 3072d Bytes)	2	00 0C	Number of bytes of NVM memory to reserve for the certificate repository.
Certificate Repository Write/ Reset Access	1	30	Access required to write or reset this repository using the ManageCertRepo command
Certificate Repository Read Command Access	1	30	Access required to read from this repository using the ReadCertRepo command

3.9.4 Program the "Device leaf" certificate - ManageCertRepo - LoadCert

Following certificate (comes with Middleware - cert_depth2_x509_rev3\cert_and_key\nist_p\device_leaf_certificate.der) will be programmed into certificate slot:

```
X509 Certificate:
```

```
Version: 3
Serial Number: 35fc393834811506dd59cfb993036ffc981bc918
Signature Algorithm:
    Algorithm ObjectId: 1.2.840.10045.4.3.2 sha256ECDSA
    Algorithm Parameters: NULL
Issuer:
    CN=NXP P1 IntermCAvE201
    O=NXP
    L=Eindhoven
    S=Eindhoven
    C=NL
  Name Hash(sha1): 39245a13ae7bbcb1ea5c863ad404fd5bcfe0942e
  Name Hash(md5): 92953f7f15eb190b17f13d05f2383273

 NotBefore: 4. 06. 2024 07:25
 NotAfter: 2. 06. 2034 07:25

Subject:
    CN=NXP LEAF CERTIFICATE
    O=NXP
    L=Eindhoven
    S=Eindhoven
    C=NL
  Name Hash(sha1): eca28d8a49dadf6b51f85544cfc219c8c270f156
  Name Hash(md5): ff40f47a7b2e394cc5f51a6d0b4e9522

Public Key Algorithm:
    Algorithm ObjectId: 1.2.840.10045.2.1 ECC
    Algorithm Parameters:
    06 08 2a 86 48 ce 3d 03  01 07
        1.2.840.10045.3.1.7 ECDSA_P256 (x962P256v1)
Public Key Length: 256 bits
Public Key: UnusedBits = 0
    0000  04 eb ff 86 3e 9d 53 32  6b bf e8 54 26 ec 53 21
    0010  8d 6b 49 66 0b b9 23 be  9f 6b c4 4d a5 74 21 31
    0020  5a 4f 2b c2 84 1c c2 f3  01 ef 97 8f 15 19 90 59
    0030  d9 98 19 42 2b 4c 66 8a  8f c0 64 04 af 73 ab 00
    0040  e6
Certificate Extensions: 1
    2.5.29.19: Flags = 0, Length = 2
    Basic Constraints
        Subject Type=End Entity
        Path Length Constraint=None

Signature Algorithm:
    Algorithm ObjectId: 1.2.840.10045.4.3.2 sha256ECDSA
    Algorithm Parameters: NULL
Signature: UnusedBits=0
    0000  35 ad 82 12 bd ee fd 88  1d e3 01 d2 c6 ba cf 72
    0010  fd 13 19 3c 07 fc b8 45  7f a6 3e 5c 99 15 31 fb
    0020  00 21 02 4c 2c cf 20 0e  a2 d9 9a 41 00 74 b7 9b
    0030  c6 83 ca 84 79 6c 7f 11  a8 08 93 ab c4 a8 17 33
    0040  50 3c 66 20 02 45 30
Non-root Certificate
Key Id Hash(rfc-sha1): bb84519a71f9a4fc5dc5b9e1bf7bcdb1f0ce90aa
Key Id Hash(sha1): 28d0e8cb43dd43c05de5cf50db7b0acc4faaa0bd
Key Id Hash(bcrypt-sha1): dc13322290249e331013dbe3dfce588c6e681678
Key Id Hash(bcrypt-sha256):
 8a83ee6b26652e04d8633e5ffe98a9408415230c065192b8b0cb614c2c0e471b
Key Id Hash(md5): eeaeff522e8b142dc8e3b685e6cbdf90
```

```
Key Id Hash(sha256):
 60cd1816fc380dd075a00596faefc8b6ea65823e5819956856bd4ee893a81b53
Key Id Hash(pin-sha256): 4FUtz7xz2/P1GQ2trF2pIAHekvSTrPzlGWDxwDfRp8I=
Key Id Hash(pin-sha256-hex):
 e0552dcfbc73dbf3f5190dadac5da92001de92f493acfce51960f1c037d1a7c2
Cert Hash(md5): 27e9f90e3db742d7db73cb56afdc998d
Cert Hash(sha1): 7453fb4fa473d6ef8c43c5e33cf06528c84c41f9
Cert Hash(sha256):
 00691cd9286cab92eb3e29566b22847c739f98e7b1107b2d7c282da3b3bf8167
Signature Hash: b5a449bedfb0d49432fbc710a1df040d599998b5333de94f4aac03fe618074a9
CertUtil: -dump command completed successfully.
```

Uncompressed ASN1 formatting:

```
30 82 01 D3 30 82 01 79 A0 03 02 01 02 02 14 35 FC 39 38 34 81 15 06 DD 59 CF B9
 93 03 6F FC 98 1B C9 18 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 62 31 0B 30 09
 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F
 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30
 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20
 50 31 20 49 6E 74 65 72 6D 43 41 76 45 32 30 31 30 1E 17 0D 32 34 30 36 30 34
 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30 32 30 36 32 35 32 39 5A 30 62 31 0B
 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64
 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31
 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58
 50 20 4C 45 41 46 20 43 45 52 54 49 46 49 43 41 54 45 30 59 30 13 06 07 2A 86
 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 EB FF 86 3E 9D 53 32
 6B BF E8 54 26 EC 53 21 8D 6B 49 66 0B B9 23 BE 9F 6B C4 4D A5 74 21 31 5A 4F
 2B C2 84 1C C2 F3 01 EF 97 8F 15 19 90 59 D9 98 19 42 2B 4C 66 8A 8F C0 64 04
 AF 73 AB 00 E6 A3 0D 30 0B 30 09 06 03 55 1D 13 04 02 30 00 30 0A 06 08 2A 86
 48 CE 3D 04 03 02 03 48 00 30 45 02 20 66 3C 50 33 17 A8 C4 AB 93 08 A8 11 7F
 6C 79 84 CA 83 C6 9B B7 74 00 41 9A D9 A2 0E 20 CF 2C 4C 02 21 00 FB 31 15 99
 5C 3E A6 7F 45 B8 FC 07 3C 19 13 FD 72 CF BA C6 D2 01 E3 1D 88 FD EE BD 12 82
 AD 35
```

Table 21. ManageCertRepo - Write Device Leaf certificate into certificate repository ID 01

Step	Command		Data
1	Cmd (INS) of Cmd.ManageCertRepo	=	49
2	CommMode	=	CommMode.MAC (as set at certificate repository creation Table 18, step7)
3	Command Action	=	01
4	Certificate Repository Id	=	01
5	Certificate Level	=	00
6	Certificate (as per , 7F 21 and TLV of 82 and Length needs to be added)	=	7F 21 82 01 D7 30 82 01 D3 30 82 01 79 A0 03 02 01 02 02 14 35 FC 39 38 34 81 15 06 DD 59 CF B9 93 03 6F FC 98 1B C9 18 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 50 31 20 49 6E 74 65 72 6D 43 41 76 45 32 30 31 30 1E 17 0D 32 34 30 36 30 34 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30

Table 21. ManageCertRepo - Write Device Leaf certificate into certificate repository ID 01*...continued*

Step	Command		Data
			32 30 36 32 35 32 39 5A 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 4C 45 41 46 20 43 45 52 54 49 46 49 43 41 54 45 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 EB FF 86 3E 9D 53 32 6B BF E8 54 26 EC 53 21 8D 6B 49 66 0B B9 23 BE 9F 6B C4 4D A5 74 21 31 5A 4F 2B C2 84 1C C2 F3 01 EF 97 8F 15 19 90 59 D9 98 19 42 2B 4C 66 8A 8F C0 64 04 AF 73 AB 00 E6 A3 0D 30 0B 30 09 06 03 55 1D 13 04 02 30 00 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 48 00 30 45 02 20 66 3C 50 33 17 A8 C4 AB 93 08 A8 11 7F 6C 79 84 CA 83 C6 9B B7 74 00 41 9A D9 A2 0E 20 CF 2C 4C 02 21 00 FB 31 15 99 5C 3E A6 7F 45 B8 FC 07 3C 19 13 FD 72 CF BA C6 D2 01 E3 1D 88 FD EE BD 12 82 AD 35
7	CmdHeader (Command Action || Certificate Repository Id ||Certificate Level || Certificate)	=	01 01 00 7F 21 … AD 35
8	CMD || CmdCtr || TI [|| CmdHeader] [|| E($K_{SesAuthENC}$, CmdData)]	=	49 07 00 70 5E 70 FB 01 01 00 7F 21 … AD 35
9	MAC($K_{SesAuthMAC}$; CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData])	=	05 D1 51 E4 12 9C 3C 74 D3 32 96 78 E5 FC EE 96
10	MACt	=	D1 E4 9C 74 32 78 FC 96
11	C-APDU (An extended Lc field is 00 01 E7h = 487d, Le is 2-byte, as per ref.[4])	>	90 49 00 00 00 01 E7 01 01 00 7F 21 82 01 D7 30 … EE BD 12 82 AD 35 D1 E4 9C 74 32 78 FC 96 00 00
12	R-APDU (MACt + SW1 SW2)	<	0E C0 A4 90 DD 46 09 09 91 00
13	Verify Response MAC RC (SW2) || CmdCtr++ || TI [|| RespData]		00 08 00 70 5E 70 FB
14	MAC($K_{SesAuthMAC}$; RC || CmdCtr || TI [|| RespData])	=	B5 0E 31 C0 FE A4 56 90 B5 DD E2 46 A2 09 77 09
15	MACt	=	0E C0 A4 90 DD 46 09 09
16	Compare MACt-s (from step 12 and step 15)	=	0E C0 A4 90 DD 46 09 09 == 0E C0 A4 90 DD 46 09 09

3.9.5 (optional) Read certificate repository from the IC - ReadCertRepo Metadata

Step is optional as data was already obtained in 2 steps back.

Table 22. ReadCertRepo

Step	Command		Data
1	Cmd (INS) of Cmd.ReadCertRepo	=	4A
2	CommMode	=	CommMode.MAC (if reading meta data only. If reading certificate repository directly, access right defined at creation of repository are needed to be set)
3	CmdHeader (RepoID || Data Item)	=	01 FF
4	CMD || CmdCtr || TI [|| CmdHeader] [|| E($K_{SesAuthENC}$, IVc, CmdData)]	=	4A 08 00 70 5E 70 FB 01 FF
5	MAC($K_{SesAuthMAC}$; CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData])	=	D2 16 DD A9 AA 8D B9 14 C4 BF E3 F5 64 E6 7D B2
6	MACt	=	16 A9 8D 14 BF F5 E6 B2
7	C-APDU	>	90 4A 00 00 00 00 0A 01 FF 16 A9 8D 14 BF F5 E6 B2 00 00
8	R-APDU (Data + MACt + SW1 SW2)	<	00 00 0C 30 30 F2 CA 8B D5 A4 C8 14 F3 91 00
9	Verify Response MAC RC (SW2) || CmdCtr++ || TI [|| RespData]	=	00 09 00 70 5E 70 FB 00 00 0C 30 30
10	MAC($K_{SesAuthMAC}$; RC || CmdCtr || TI [|| RespData])	=	56 F2 7B CA C5 8B FB D5 C2 A4 69 C8 8A 14 11 F3
11	MACt	=	F2 CA 8B D5 A4 C8 14 F3
12	Compare MACt-s (from step 8 and step 11)	=	F2 CA 8B D5 A4 C8 14 F3 == F2 CA 8B D5 A4 C8 14 F3

Table 23. ReadCertRepo - Response Data Format for Metadata

Name	Length	Value	Description
Private Key Id	1	00	Id of ECC private key associated with this repository (key must have been created using Manage KeyPair).
Repository Size (LSB) (0C 00h = 3072d Bytes)	2	00 0C	Number of bytes of NVM memory to reserve for the certificate repository.
Certificate Repository Write/ Reset Access	1	30	Access required to write or reset this repository using the ManageCertRepo command
Certificate Repository Read Command Access	1	30	Access required to read from this repository using the ReadCertRepo command

3.9.6 Program the "Device Intermediate (P1)" certificate - ManageCertRepo - LoadCert

Following certificate will be programmed into certificate slot:

```
X509 Certificate:
Version: 3
```

```
Serial Number: 44393990ca2e5b63c889b6a7fcaa722ff67dfb63
Signature Algorithm:
    Algorithm ObjectId: 1.2.840.10045.4.3.2 sha256ECDSA
    Algorithm Parameters: NULL
Issuer:
    CN=NXP Auth RootCAvE201
    O=NXP
    L=Eindhoven
    S=Eindhoven
    C=NL
  Name Hash(sha1): 96722d219aca9e678d7cdbe930852172d33045ec
  Name Hash(md5): fb0369a3aebbb29dfb7aef76764f709e

 NotBefore: 4. 06. 2024 07:25
 NotAfter: 2. 06. 2034 07:25

Subject:
    CN=NXP P1 IntermCAvE201
    O=NXP
    L=Eindhoven
    S=Eindhoven
    C=NL
  Name Hash(sha1): 39245a13ae7bbcb1ea5c863ad404fd5bcfe0942e
  Name Hash(md5): 92953f7f15eb190b17f13d05f2383273

Public Key Algorithm:
    Algorithm ObjectId: 1.2.840.10045.2.1 ECC
    Algorithm Parameters:
    06 08 2a 86 48 ce 3d 03  01 07
        1.2.840.10045.3.1.7 ECDSA_P256 (x962P256v1)
Public Key Length: 256 bits
Public Key: UnusedBits = 0
    0000  04 59 4e 77 df 1c 6b 27  c4 16 6e 1c fe b1 c1 d5
    0010  bb 3f b8 fe 07 6a dc 7a  61 5c d3 7e 08 15 d1 a1
    0020  92 47 df ff c5 76 ee 73  6d 40 14 f3 5e 3d 2e 41
    0030  ce e1 06 2e 58 7a 60 48  2e 19 98 7b 88 8a a6 90
    0040  9d
Certificate Extensions: 1
    2.5.29.19: Flags = 0, Length = 5
    Basic Constraints
        Subject Type=CA
        Path Length Constraint=None

Signature Algorithm:
    Algorithm ObjectId: 1.2.840.10045.4.3.2 sha256ECDSA
    Algorithm Parameters: NULL
Signature: UnusedBits=0
    0000  d7 85 38 96 14 a6 1d 20  df f4 48 cb e5 d6 a0 f4
    0010  01 f4 84 af 86 cf 6d c2  38 9a 5f 9d b7 04 91 54
    0020  20 02 8e b0 c9 ba a5 1d  29 27 8e 69 23 42 4a a0
    0030  c5 66 05 06 d0 6e 0a 8d  13 f0 20 1e da 5d e9 34
    0040  bd 65 20 02 44 30
Non-root Certificate
Key Id Hash(rfc-sha1): d443dad4e98effd493d11d9d74127f9ea3302d5b
Key Id Hash(sha1): 55547bac44b73ca0122166ba2ea6204b1a572f93
Key Id Hash(bcrypt-sha1): 4ce66d1771d1f57623647a99d2ff51b49daa2782
Key Id Hash(bcrypt-sha256):
 46031871e0768b9ffc7639643e28e7f7fe85e290bdae91d6b2717b88b7f404cd
Key Id Hash(md5): 9659219465959ca2840cfe4570728f2f
```

```
Key Id Hash(sha256):
 8cef2b5b4001f43a53c4fb348a8753e343e523179216a19dbdd959f88c582e5a
Key Id Hash(pin-sha256): hYJKaDiWiZ/D3JkoOXMcT6CxInBpZuXf4A9R5F2iiDM=
Key Id Hash(pin-sha256-hex):
 85824a683896899fc3dc992839731c4fa0b122706966e5dfe00f51e45da28833
Cert Hash(md5): 0c8f2344c8af8728d029220d58789c88
Cert Hash(sha1): 8912d826f454d17bd79d430b3f1ae93c95b45f22
Cert Hash(sha256):
 c3231ea905f4042d65c4912fdcd090cfa6cc66d19360fb4e2d186cf3b1b831df
Signature Hash: 4ceecec423d6b7081ac2d71f7482fdb50ec2d28a0be7ed5fad333d998d830190
CertUtil: -dump command completed successfully.
```

Uncompressed ASN1 formatting:

```
30 82 01 D5 30 82 01 7C A0 03 02 01 02 02 14 44 39 39 90 CA 2E 5B 63 C8 89 B6 A7
 FC AA 72 2F F6 7D FB 63 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 62 31 0B 30 09
 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F
 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30
 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20
 41 75 74 68 20 52 6F 6F 74 43 41 76 45 32 30 31 30 1E 17 0D 32 34 30 36 30 34
 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30 32 30 36 32 35 32 39 5A 30 62 31 0B
 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64
 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31
 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58
 50 20 50 31 20 49 6E 74 65 72 6D 43 41 76 45 32 30 31 30 59 30 13 06 07 2A 86
 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 59 4E 77 DF 1C 6B 27
 C4 16 6E 1C FE B1 C1 D5 BB 3F B8 FE 07 6A DC 7A 61 5C D3 7E 08 15 D1 A1 92 47
 DF FF C5 76 EE 73 6D 40 14 F3 5E 3D 2E 41 CE E1 06 2E 58 7A 60 48 2E 19 98 7B
 88 8A A6 90 9D A3 10 30 0E 30 0C 06 03 55 1D 13 04 05 30 03 01 01 FF 30 0A 06
 08 2A 86 48 CE 3D 04 03 02 03 47 00 30 44 02 20 65 BD 34 E9 5D DA 1E 20 F0 13
 8D 0A 6E D0 06 05 66 C5 A0 4A 42 23 69 8E 27 29 1D A5 BA C9 B0 8E 02 20 54 91
 04 B7 9D 5F 9A 38 C2 6D CF 86 AF 84 F4 01 F4 A0 D6 E5 CB 48 F4 DF 20 1D A6 14
 96 38 85 D7
```

Table 24. ManageCertRepo - Write Device P1 (Intermediate) certificate into certificate repository ID 01

Step	Command		Data
1	Cmd (INS) of Cmd.ManageCertRepo	=	49
2	CommMode	=	CommMode.MAC (as set at certificate repository creation Table 18, step7)
3	Command Action	=	01
4	Certificate Repository Id	=	01
5	Certificate Level	=	01
6	Certificate	=	7F 21 82 01 D9 30 82 01 D5 30 82 01 7C A0 03 02 01 02 02 14 44 39 39 90 CA 2E 5B 63 C8 89 B6 A7 FC AA 72 2F F6 7D FB 63 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 41 75 74 68 20 52 6F 6F 74 43 41 76 45 32 30 31 30 1E 17 0D 32 34 30 36

Table 24. ManageCertRepo - Write Device P1 (Intermediate) certificate into certificate repository ID 01*...continued*

Step	Command		Data
			30 34 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30 32 30 36 32 35 32 39 5A 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 50 31 20 49 6E 74 65 72 6D 43 41 76 45 32 30 31 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 59 4E 77 DF 1C 6B 27 C4 16 6E 1C FE B1 C1 D5 BB 3F B8 FE 07 6A DC 7A 61 5C D3 7E 08 15 D1 A1 92 47 DF FF C5 76 EE 73 6D 40 14 F3 5E 3D 2E 41 CE E1 06 2E 58 7A 60 48 2E 19 98 7B 88 8A A6 90 9D A3 10 30 0E 30 0C 06 03 55 1D 13 04 05 30 03 01 01 FF 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 47 00 30 44 02 20 65 BD 34 E9 5D DA 1E 20 F0 13 8D 0A 6E D0 06 05 66 C5 A0 4A 42 23 69 8E 27 29 1D A5 BA C9 B0 8E 02 20 54 91 04 B7 9D 5F 9A 38 C2 6D CF 86 AF 84 F4 01 F4 A0 D6 E5 CB 48 F4 DF 20 1D A6 14 96 38 85 D7
7	CmdHeader (Command Action || Certificate Repository Id ||Certificate Level || Certificate)	=	01 01 01 7F 21 … 85 D7
8	CMD || CmdCtr || TI [|| CmdHeader] [|| E($K_{SesAuthENC}$, CmdData)]	=	49 09 00 70 5E 70 FB 01 01 01 7F 21 … 85 D7
9	MAC($K_{SesAuthMAC}$; CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData])	=	C1 23 9D CA E2 61 87 05 8C 27 48 7F B8 E3 67 05
10	MACt	=	23 CA 61 05 27 7F E3 05
11	C-APDU (An extended Lc field is 00 01 E9h = 489d, Le is 2-byte, as per ref.[4])	>	90 49 00 00 00 01 E9 01 01 01 7F 21 82 01 D9 30 ... 20 1D A6 14 96 38 85 D7 23 CA 61 05 27 7F E3 05 00 00
12	R-APDU (MACt + SW1 SW2)	<	75 0F BF E0 E7 90 9C E0 91 00
13	Verify Response MAC RC (SW2) || CmdCtr++ || TI [|| RespData]		00 0A 00 70 5E 70 FB
14	MAC($K_{SesAuthMAC}$; RC || CmdCtr || TI [|| RespData])	=	9B 75 70 0F D7 BF 17 E0 6A E7 F4 90 95 9C 28 E0
15	MACt	=	75 0F BF E0 E7 90 9C E0
16	Compare MACt-s (from step 12 and step 16)	=	75 0F BF E0 E7 90 9C E0 == 75 0F BF E0 E7 90 9C E0

3.9.7 (optional) Read certificate repository from the IC - ReadCertRepo Metadata

Step is optional as data was already obtained in 2 steps back.

Table 25. ReadCertRepo

Step	Command		Data
1	Cmd (INS) of Cmd.ReadCertRepo	=	4A
2	CommMode	=	CommMode.MAC (if reading meta data only. If reading certificate repository directly, access right defined at creation of repository are needed to be set)
3	CmdHeader (RepoID || Data Item)	=	01 FF
4	CMD || CmdCtr || TI [|| CmdHeader] [|| $E(K_{SesAuthENC}, IVc, CmdData)$]	=	4A 0A 00 70 5E 70 FB 01 FF
5	$MAC(K_{SesAuthMAC}$; CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData])	=	56 71 CF B3 8A DC 23 B9 64 55 9B A1 56 49 CC B6
6	MACt	=	71 B3 DC B9 55 A1 49 B6
7	C-APDU	>	90 4A 00 00 00 00 0A 01 FF 71 B3 DC B9 55 A1 49 B6 00 00
8	R-APDU (Data + MACt + SW1 SW2)	<	00 00 0C 30 30 6C 20 3F 5A B2 19 44 77 91 00
9	Verify Response MAC RC (SW2) || CmdCtr++ || TI [|| RespData]	=	00 0B 00 70 5E 70 FB 00 00 0C 30 30
10	$MAC(K_{SesAuthMAC}$; RC || CmdCtr || TI [|| RespData])	=	7F 6C 12 20 47 3F 14 5A 37 B2 0F 19 7E 44 BC 77
11	MACt	=	6C 20 3F 5A B2 19 44 77
12	Compare MACt-s (from step 8 and step 11)	=	6C 20 3F 5A B2 19 44 77 == 6C 20 3F 5A B2 19 44 77

Table 26. ReadCertRepo - Response Data Format for Metadata

Name	Length	Value	Description
Private Key Id	1	00	Id of ECC private key associated with this repository (key must have been created using Manage KeyPair).
Repository Size (LSB) (0C 00h = 3072d Bytes)	2	00 0C	Number of bytes of NVM memory to reserve for the certificate repository.
Certificate Repository Write/ Reset Access	1	30	Access required to write or reset this repository using the ManageCertRepo command
Certificate Repository Read Command Access	1	30	Access required to read from this repository using the ReadCertRepo command

3.9.8 Activate certificate repository - ManageCertRepo - ActivateRepo

Certificate repository needs to be Activated.

Table 27. ManageCertRepo - Activate certificate repository

Step	Command		Data
1	Cmd (INS) of Cmd.ManageCertRepo	=	49
2	CommMode	=	CommMode.MAC (as set at certificate repository creation Table 18, step7)
3	Command Action	=	04
4	Certificate Repository Id	=	01
5	CmdHeader (Command Action || Certificate Repository Id)	=	04 01
6	CMD || CmdCtr || TI [|| CmdHeader] [|| $E(K_{SesAuthENC}, CmdData)$]	=	49 0B 00 70 5E 70 FB 04 01
7	$MAC(K_{SesAuthMAC}; CMD \|\| CmdCtr \|\| TI [\|\| CmdHeader] [\|\| CmdData])$	=	12 2C 01 D8 65 58 99 4C EC E5 75 2E 08 97 B1 34
8	MACt	=	2C D8 58 4C E5 2E 97 34
9	C-APDU	>	90 49 00 00 0A 04 01 2C D8 58 4C E5 2E 97 34 00
10	R-APDU (MACt + SW1 SW2)	<	4D 78 BC 7E 26 D7 48 B0 91 00
11	Verify Response MAC RC (SW2) || CmdCtr++ || TI [|| RespData]		00 0C 00 70 5E 70 FB
12	$MAC(K_{SesAuthMAC}; RC \|\| CmdCtr \|\| TI [\|\| RespData])$	=	90 4D 4B 78 10 BC FF 7E 5E 26 C0 D7 6B 48 5F B0
13	MACt	=	4D 78 BC 7E 26 D7 48 B0
14	Compare MACt-s (from step 10 and step 13)	=	4D 78 BC 7E 26 D7 48 B0 == 4D 78 BC 7E 26 D7 48 B0

4 Verification of personalization

To verify certificate chain on the host side, in following flow so called SIGMA-I Asymmetric authentication is performed.

4.1 SIGMA-I authentication with ISOGeneralAuthenticate

SIGMA-I stands for Sign, MAC, and protect Identity of authentication initiator using encryption. The SIGMA-I protocol consists of an exchange of three messages between the Initiator (or SIGMA-I Verifier) and the Responder (or SIGMA-I Prover). Since the Initiators' and Responders' identities (certificates) are exchanged encrypted, they are protected against passive attacks. The Initiators' identity is protected against active attackers, because the Initiator has the opportunity of verifying the identity of the Responder before disclosing its own identity. "Man in the middle" attacks are prevented by signing the payloads. Identity misbinding attacks are prevented by MACing the identities.

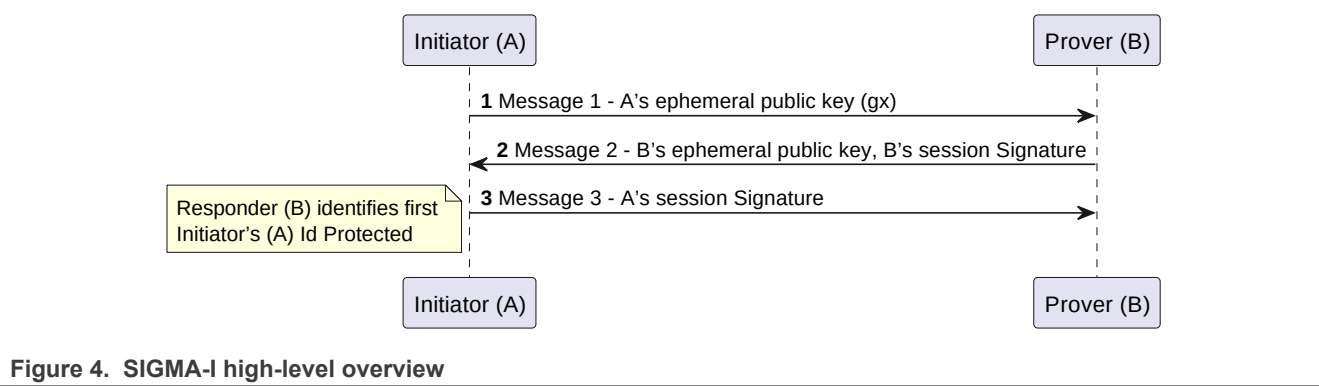

Figure 4. SIGMA-I high-level overview

Notes on the following examples:

- First-time authentication - no cache available on NTAG X DNA. NFC Host can act as SIGMA-I Verifier or Responder.
- After successful authentication, session keys are generated and the Cmd.FreeMem command is sent.

4.1.1 Host as Initiator

Conditions:

- No cache on Host
- First-time authentication (certificates need to be exchanged)
- Knowledge of $S.PrivKey_{HOST}$
- Certificates used in this example can be found in NX Middleware sub-dir "cert_depth2_x509_rev3\cert_and_key\nist_p\":
  - device_leaf_certificate.der
  - device_p1_certificate.der
  - device_root_certificate.der
  - host_leaf_certificate.der
  - host_p1_certificate.der
  - host_root_certificate.der

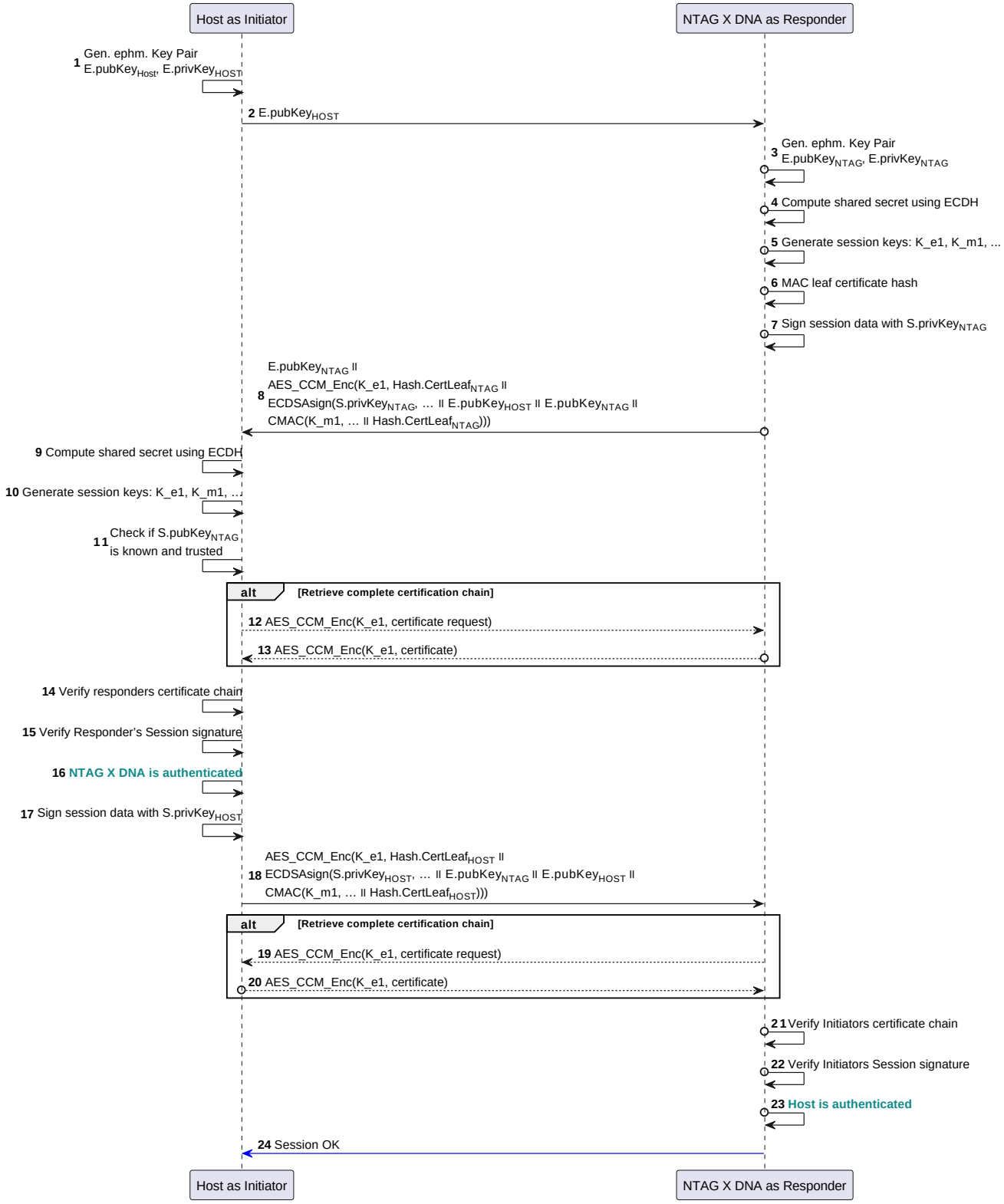

Figure 5. SIGMA-I Host as Initiator

4.1.1.1 ECDH key exchange and Session keys computation

Table 28. Generate Host Ephemeral key pair and exchange it with target. Get "Device Leaf certificate" signature and ECC signature over.

Step	Command		Data
1	Read contents of Device Root Certificate (stored on Host)	=	30 82 01 D7 30 82 01 7C A0 03 02 01 02 02 14 04 31 63 DC B1 BE 85 02 65 3C 9D BD B0 2E C6 33 0E DB EE 2D 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 41 75 74 68 20 52 6F 6F 74 43 41 76 45 32 30 31 30 1E 17 0D 32 34 30 36 30 34 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30 32 30 36 32 35 32 39 5A 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 41 75 74 68 20 52 6F 6F 74 43 41 76 45 32 30 31 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 D3 F0 AE 4A ED 73 31 55 D8 67 D0 2A 82 DE 3F 7B 88 AF BD B1 93 F2 E3 B9 9B B6 AB 8F 08 3A D7 39 13 D9 84 9F E8 CC A3 57 F7 55 F2 6F F6 D9 F1 F1 42 AC F0 2A F6 BE 7E 2C 49 6C F3 CE AA 3E E4 30 A3 10 30 0E 30 0C 06 03 55 1D 13 04 05 30 03 01 01 FF 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 49 00 30 46 02 21 00 D8 FB E3 F3 E0 99 6F 84 40 69 71 5D 3A 51 9A 02 C1 B2 2C 49 A4 9C 9B D0 4E 4E A2 3B 2B DD 61 CA 02 21 00 9C 12 96 05 62 A6 CD 07 46 11 0D AB 2E 04 83 FD 67 B2 67 56 74
2	Generate ECDH key pair - Private Key - E. $privKey_{HOST}$	=	1E 06 AE F7 A2 35 55 8D D7 54 B4 83 ED 7F 2B B9 78 E4 09 D2 A9 57 CE 52 0C 93 82 12 38 2F 68 73
3	Generate ECDH key pair - Public Key - E. $pubKey_{HOST}$	=	04 55 15 64 97 59 4E 96 53 13 D5 0A C2 2C 00 C8 F2 26 D7 54 78 3E 63 5A CA 69 2F 5A 42 37 F7 9D 36 DD 56 4B 07 0C BE BB 6F 33 E0 21 91 42 95 1F C1 C7 1A 23 3C D9 16 BD 04 5C 64 51 A2 8C 47 AF DB
4	Decide on AES session key size (AES-128)	=	01
5	CLA of ISOGeneralAuthenticate	=	00
6	Cmd (INS) of ISOGeneralAuthenticate	=	86
7	P1 (SIGMA-I)	=	01
8	P2 (Certificate repository)	=	01
9	Lc	=	48

Table 28. Generate Host Ephemeral key pair and exchange it with target. Get "Device Leaf certificate" signature and ECC signature over...*continued*

Step	Command		Data
10	(SIGMA-I Message Type) Initiator sends its supported AES key sizes and its ephemeral public key + Length	=	A0 46
11	(Protocols Payload Encodings) Asymmetric authentication Protocols Payload Encoding - AES key size option	=	83 01
12	AES key size option + AES session key size (AES-128)	=	83 01 01
13	Asymmetric authentication Protocols Payload Encoding - Ephemeral ECDH public key, plaintext, uncompressed format		86 41
14	Data (MSGI_PUBLIC_KEY)		83 01 01 86 41 04 55 15 64 97 59 4E 96 53 13 D5 0A C2 2C 00 C8 F2 26 D7 54 78 3E 63 5A CA 69 2F 5A 42 37 F7 9D 36 DD 56 4B 07 0C BE BB 6F 33 E0 21 91 42 95 1F C1 C7 1A 23 3C D9 16 BD 04 5C 64 51 A2 8C 47 AF DB 00
15	Le	=	00
16	C-APDU	>	00 86 01 01 48 A0 46 83 01 01 86 41 04 55 15 64 97 59 4E 96 53 13 D5 0A C2 2C 00 C8 F2 26 D7 54 78 3E 63 5A CA 69 2F 5A 42 37 F7 9D 36 DD 56 4B 07 0C BE BB 6F 33 E0 21 91 42 95 1F C1 C7 1A 23 3C D9 16 BD 04 5C 64 51 A2 8C 47 AF DB 00
17	R-APDU (E.pubKey$_{NTAG}$, AES_CCM_ Enc(Certificate hash and signature)) + SW1 SW2	<	B1 81 B0 83 01 01 86 41 04 9B 1E D1 AA DC 69 B1 3C 4B 33 A3 79 4A E4 0B 4F 8F 76 D5 E5 09 FC A6 E5 58 C0 61 20 A1 51 4D 72 13 3C 1C 0B 7C 1D B6 0F 5C 59 A9 12 54 EA 92 0F C8 D9 CE 08 3D 08 1F 9A 12 71 EC E5 F6 85 1D 19 87 68 2B 86 87 A5 C8 B5 B9 F4 08 54 BB 97 D0 87 04 AA 04 43 2B 13 F0 E6 BB 61 52 7F F4 AA B5 08 5D 13 1E 45 0B AE 2C 29 2F 2B B3 1F 57 40 FC 09 E9 5F A9 84 5F AC D9 B8 D9 CA 88 93 8A 7B 74 A1 9E E7 79 16 D7 B1 9B 05 03 10 79 C2 13 B0 F1 25 BA 62 43 0E CD 27 2E B3 6F 8F 06 6D F6 EA 3C 3B B1 6A 09 8B FD B6 27 C9 BB EF 90 00
18	(SIGMA-I Message Type) MSGR_HASH_ AND_SIG (Responder sends the session AES key size and its ephemeral public key, certificate hash and signature)	=	B1 81 B0
19	(Protocols Payload Encodings) AES key size options (83 01)	=	01
20	(Protocols Payload Encodings) NTAG's Ephemeral ECDH public key - E.pub Key$_{NTAG}$ uncompressed format (86 41)	=	04 9B 1E D1 AA DC 69 B1 3C 4B 33 A3 79 4A E4 0B 4F 8F 76 D5 E5 09 FC A6 E5 58 C0 61 20 A1 51 4D 72 13 3C 1C 0B 7C 1D B6 0F 5C 59 A9 12 54 EA 92 0F

Table 28. Generate Host Ephemeral key pair and exchange it with target. Get "Device Leaf certificate" signature and ECC signature over...*continued*

Step	Command		Data
			C8 D9 CE 08 3D 08 1F 9A 12 71 EC E5 F6 85 1D 19
21	(Protocols Payload Encodings) Encrypted payload (87 68) - C_k_r (Encrypted hash and signature)	=	2B 86 87 A5 C8 B5 B9 F4 08 54 BB 97 D0 87 04 AA 04 43 2B 13 F0 E6 BB 61 52 7F F4 AA B5 08 5D 13 1E 45 0B AE 2C 29 2F 2B B3 1F 57 40 FC 09 E9 5F A9 84 5F AC D9 B8 D9 CA 88 93 8A 7B 74 A1 9E E7 79 16 D7 B1 9B 05 03 10 79 C2 13 B0 F1 25 BA 62 43 0E CD 27 2E B3 6F 8F 06 6D F6 EA 3C 3B B1 6A 09 8B FD B6 27 C9 BB EF
22	**Compute shared secret ShS** ShS = ECDH(E.privKey$_{HOST}$, E.pub Key$_{NTAG}$) - will be always random	=	32 6F C5 91 4C BE 27 44 6D 6F F7 B3 52 B8 3A 05 C9 D2 04 F1 F9 E0 60 B7 67 CB 48 9A CA C1 60 F7
	x-coordinate of shared secret | initators's public key | responder's public key		32 6F C5 91 4C BE 27 44 6D 6F F7 B3 52 B8 3A 05 C9 D2 04 F1 F9 E0 60 B7 67 CB 48 9A CA C1 60 F7 55 15 64 97 59 4E 96 53 13 D5 0A C2 2C 00 C8 F2 26 D7 54 78 3E 63 5A CA 69 2F 5A 42 37 F7 9D 36 DD 56 4B 07 0C BE BB 6F 33 E0 21 91 42 95 1F C1 C7 1A 23 3C D9 16 BD 04 5C 64 51 A2 8C 47 AF DB 9B 1E D1 AA DC 69 B1 3C 4B 33 A3 79 4A E4 0B 4F 8F 76 D5 E5 09 FC A6 E5 58 C0 61 20 A1 51 4D 72 13 3C 1C 0B 7C 1D B6 0F 5C 59 A9 12 54 EA 92 0F C8 D9 CE 08 3D 08 1F 9A 12 71 EC E5 F6 85 1D 19
23	Derive the base key Ki (take 8d MSB) Ki = SHA-256(x-coordinate of shared secret | initators's public key | responder's public key)	=	4A 61 88 32 60 D9 98 37 36 CE A3 97 0F 60 68 15
24	K_e1:= AESCMAC(Ki, [i]2 || “K_e1“ (into HEX) || 0x00 || "SIGMA-I" (into HEX) || [L]2)	=	0001 4B5F6531 00 5349474D412D49 0080
25	K_e1	=	80 AE A0 65 AE 8B 0F 2D F0 A2 5D F7 9E 53 24 2F
26	K_m1:= AESCMAC(Ki, [i]2 || “K_m1“ (into HEX) || 0x00 || "SIGMA-I" (into HEX) || [L]2)	=	0001 4B5F6D31 00 5349474D412D49 0080
27	K_m1	=	D3 C6 BC A2 A3 4C 55 3F 4F C6 29 3C C6 0F FB 3B
28	IV_e1:= AESCMAC(Ki, [i]2 || “IV_e1“ (into HEX) || 0x00 || "IVs" (into HEX) || [L]2 || 0x 000000)	=	00 01 49 56 5F 65 31 00 49 56 73 00 68 00 00 00
29	IV_e1	=	CF 39 FD B8 7C 9A 5A 50 AF F4 28 5C C1 7C A9 F0
30	13 leftmost IV_e1 bytes	=	CF 39 FD B8 7C 9A 5A 50 AF F4 28 5C C1
31	K_e2:= AESCMAC(Ki, [i]2 || “K_e2“ (into HEX) || 0x00 || "EV2" (into HEX) || [L]2 || 0x00000000)	=	00 01 4B 5F 65 32 00 45 56 32 00 80 00 00 00 00
32	K_e2	=	41 CC A9 73 57 F4 C4 1F 12 E8 EE 68 B2 DF 8F 95
33	K_m2:= AESCMAC(Ki, [i]2 || “K_m2“ (into HEX) || 0x00 || "EV2" (into HEX) || [L]2 || 0x00000000)	=	00 01 4B 5F 6D 32 00 45 56 32 00 80 00 00 00 00

Table 28. Generate Host Ephemeral key pair and exchange it with target. Get "Device Leaf certificate" signature and ECC signature over...*continued*

Step	Command		Data
34	K_m2	=	B5 6E A9 F7 AC 25 07 18 B5 DA 68 B9 8D 45 83 8F
35	Encrypted (Cert hash + Signature) (C_k_r)	=	2B 86 87 A5 C8 B5 B9 F4 08 54 BB 97 D0 87 04 AA 04 43 2B 13 F0 E6 BB 61 52 7F F4 AA B5 08 5D 13 1E 45 0B AE 2C 29 2F 2B B3 1F 57 40 FC 09 E9 5F A9 84 5F AC D9 B8 D9 CA 88 93 8A 7B 74 A1 9E E7 79 16 D7 B1 9B 05 03 10 79 C2 13 B0 F1 25 BA 62 43 0E CD 27 2E B3 6F 8F 06 6D F6 EA 3C 3B B1 6A 09 8B FD B6 27 C9 BB EF
36	AES CCM Decrypt Auth. field length	=	8d (09 8B FD B6 27 C9 BB EF)
37	AES_CCM_Dec(K_e1, Iv_e1, C_K_r) == Decrypted (Cert hash + Signature) (which coresponds to the hash of device_leaf_ certificate.der)	=	00 69 1C D9 28 6C AB 92 EB 3E 29 56 6B 22 84 7C 73 9F 98 E7 B1 10 7B 2D 7C 28 2D A3 B3 BF 81 67 73 94 0B 69 FA CC F3 EC AF 06 41 5A 7A CF C7 AF AA E0 83 2D 68 F8 FF CB 5D 1B 38 F6 61 B1 60 98 37 FE 6D 39 41 89 55 3B 27 FA 50 37 1C 18 9B FE AE 83 7E 7E B8 F3 D1 25 C4 89 49 B6 92 EA 25 14
38	Device leaf cert hash (leaf_cert_hash)	=	00 69 1C D9 28 6C AB 92 EB 3E 29 56 6B 22 84 7C 73 9F 98 E7 B1 10 7B 2D 7C 28 2D A3 B3 BF 81 67
39	Device ECC signature (Resp_ECC_Sig)	=	73 94 0B 69 FA CC F3 EC AF 06 41 5A 7A CF C7 AF AA E0 83 2D 68 F8 FF CB 5D 1B 38 F6 61 B1 60 98 37 FE 6D 39 41 89 55 3B 27 FA 50 37 1C 18 9B FE AE 83 7E 7E B8 F3 D1 25 C4 89 49 B6 92 EA 25 14
40	Check if E.pubKey$_{HOST}$ is known and trusted	=	false (proceed to request Leaf certificate from Responder (NTAG X DNA))

4.1.1.2 Host requests Responders cert. chain

Table 29. Host requests device's Leaf certificate

Step	Command		Data
1	MSGI_CERT_REQUEST (Initiator requests a certificate from responder)	=	A2
2	Length of the whole MSGI_CERT_ REQUEST payload	=	0C
3	Tag for Encrypted payload	=	87
4	Length of the Encrypted payload	=	0A
5	Tag and Length for Certificate request (leaf, level = 0)	=	80 00
6	++IV_e1	=	CF 39 FD B8 7C 9A 5A 50 AF F4 28 5C C2
7	AES_CCM_Enc(K_e1, ++IV_e1, A=NULL, Tag and Length for Certificate request)	=	67 8E 89 A0 CA 78 E6 F2 04 E5
8	C-APDU	>	00 86 01 01 00 00 0E A2 0C 87 0A 67 8E 89 A0 CA 78 E6 F2 04 E5 00 00
9	R-APDU + SW1 SW2	<	B3 82 01 E4 C1 D7 E4 8F 12 20 4C 58 40 DD A5 E7 16 68 D5 1F 28 05 4E 6D 92 6A 08 D6 45 E8 1B E6

Table 29. Host requests device's Leaf certificate*...continued*

Step	Command		Data
			8F 56 D1 D7 C9 71 70 96 67 B4 BC 59 C1 32 B2 5C BA 4E E4 2A 24 96 78 5F 8B 00 97 A6 48 77 95 A2 03 C9 1E EB EB 70 3B 6A 20 46 F3 1B 5F 78 A0 ED FB 1B 1D 99 C1 23 35 AD 47 34 9C FD 98 98 0E 7E BF A6 5E E7 6B 4F 75 57 47 19 17 8E 83 C4 8A 33 AC 35 C8 75 35 8B A7 F6 75 CC B7 34 EE 32 BD 2F 01 28 55 56 13 E2 4A 9B 6A 1F BA A9 58 49 CD 57 D5 A2 B9 A2 28 BD E3 07 9A 44 DF 99 B3 57 5B 03 7A 1C 6E 5F 89 D8 F8 CA 36 BA 9A 19 AD 2F D2 90 DF 5B 32 66 14 54 40 23 50 A5 A6 7A DE 21 CC 5F 5F A0 8F E8 20 65 77 21 D0 08 AE 97 14 29 56 F1 C3 AD A2 A8 03 7C 44 45 36 14 40 99 3E 42 ED 1E 17 CC A2 55 FE A9 47 AD FE 62 D5 92 C7 AA 94 A5 69 9B 78 9E B6 34 8E B6 DD D9 5C 6E 36 34 83 06 10 3F C1 D4 CE 12 CC B1 6A 30 50 88 75 A7 F2 4B A8 76 0E 36 CC 79 29 49 B3 24 C5 A0 DD 29 1A 30 AD 5F C2 95 A7 8C C2 5F F2 28 B9 00 DD 55 E8 F1 A2 65 C7 83 40 59 39 B8 6D 19 09 A7 21 35 52 68 1F 76 B6 1F A7 B4 8D 37 D3 E0 35 CD 18 AD 36 A2 17 F5 83 07 EA 43 57 BD 80 86 C6 FB C9 C4 14 8F 5F 36 BB 52 DC 67 AB 96 07 AE CB 46 C3 8F BD 33 83 25 D5 CD 55 20 A3 5B DB 37 6F E1 EC 5C 28 54 0A 76 DD EA D8 17 44 C7 98 D1 11 50 C0 95 D7 9F 2C C7 40 98 96 81 AA 03 3F 78 05 BF AE 05 73 9F 24 83 05 8D 67 35 AC 6F 45 C0 42 F5 41 55 91 A5 A0 FA FA AF B1 2F 36 51 79 E7 C1 63 C6 17 5E 62 1C 48 54 3A 06 FD 43 48 82 7F 52 46 A3 7E B1 48 96 5F 3C 39 31 AC FC 0A 17 9D 07 0F 08 0A 6D EB 7A D9 AD 64 C8 0F 10 CF 90 00
10	SIGMA-I message type, Tag and Length	=	B3 82 01 E4
11	++IV_e1	=	CF 39 FD B8 7C 9A 5A 50 AF F4 28 5C C3
12	AES_CCM_Dec(K_e1, Iv_e1, R-APDU (without B3 82 01 E4))		7F 21 82 01 D7 30 82 01 D3 30 82 01 79 A0 03 02 01 02 02 14 35 FC 39 38 34 81 15 06 DD 59 CF B9 93 03 6F FC 98 1B C9 18 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 50 31 20 49 6E 74 65 72 6D 43 41 76 45 32 30 31 30 1E 17 0D 32 34 30 36 30 34 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30 32 30 36 32 35 32 39 5A 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30

Table 29. Host requests device's Leaf certificate*...continued*

Step	Command		Data
			1B 06 03 55 04 03 0C 14 4E 58 50 20 4C 45 41 46 20 43 45 52 54 49 46 49 43 41 54 45 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 EB FF 86 3E 9D 53 32 6B BF E8 54 26 EC 53 21 8D 6B 49 66 0B B9 23 BE 9F 6B C4 4D A5 74 21 31 5A 4F 2B C2 84 1C C2 F3 01 EF 97 8F 15 19 90 59 D9 98 19 42 2B 4C 66 8A 8F C0 64 04 AF 73 AB 00 E6 A3 0D 30 0B 30 09 06 03 55 1D 13 04 02 30 00 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 48 00 30 45 02 20 66 3C 50 33 17 A8 C4 AB 93 08 A8 11 7F 6C 79 84 CA 83 C6 9B B7 74 00 41 9A D9 A2 0E 20 CF 2C 4C 02 21 00 FB 31 15 99 5C 3E A6 7F 45 B8 FC 07 3C 19 13 FD 72 CF BA C6 D2 01 E3 1D 88 FD EE BD 12 82 AD 35
13	Tag (Uncompressed certificate)	=	7F 21
14	Tag and Length	=	82 01 D7
15	Device Leaf Certificate (ASN1)	=	30 82 01 D3 30 82 01 79 A0 03 02 01 02 02 14 35 FC 39 38 34 81 15 06 DD 59 CF B9 93 03 6F FC 98 1B C9 18 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 50 31 20 49 6E 74 65 72 6D 43 41 76 45 32 30 31 30 1E 17 0D 32 34 30 36 30 34 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30 32 30 36 32 35 32 39 5A 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 4C 45 41 46 20 43 45 52 54 49 46 49 43 41 54 45 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 EB FF 86 3E 9D 53 32 6B BF E8 54 26 EC 53 21 8D 6B 49 66 0B B9 23 BE 9F 6B C4 4D A5 74 21 31 5A 4F 2B C2 84 1C C2 F3 01 EF 97 8F 15 19 90 59 D9 98 19 42 2B 4C 66 8A 8F C0 64 04 AF 73 AB 00 E6 A3 0D 30 0B 30 09 06 03 55 1D 13 04 02 30 00 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 48 00 30 45 02 20 66 3C 50 33 17 A8 C4 AB 93 08 A8 11 7F 6C 79 84 CA 83 C6 9B B7 74 00 41 9A D9 A2 0E 20 CF 2C 4C 02 21 00 FB 31 15 99 5C 3E A6 7F 45 B8 FC 07 3C 19 13 FD 72 CF BA C6 D2 01 E3 1D 88 FD EE BD 12 82 AD 35
16	Check Host cache for P1 device certificate	=	false

Table 29. Host requests device's Leaf certificate*...continued*

Step	Command		Data
17	Request for Level 1 device certificate (Intermediate certificate P1)	=	true

Table 30. Host requests device's Intermediate (P1) certificate

Step	Command		Data
1	MSGI_CERT_REQUEST (Initiator requests a certificate from responder)	=	A2
2	Length of the whole MSGI_CERT_ REQUEST payload	=	0C
3	Tag for Encrypted payload	=	87
4	Length of the Encrypted payload	=	0A
5	Tag and Length for Certificate request (P1, level = 1)	=	81 00
6	++IV_e1	=	CF 39 FD B8 7C 9A 5A 50 AF F4 28 5C C4
7	AES_CCM_Enc(K_e1, ++IV_e1, A=NULL, Tag and Length for Certificate request)	=	F1 C7 D7 76 82 97 DB CD 5B FC
8	C-APDU	>	00 86 01 01 00 00 0E A2 0C 87 0A F1 C7 D7 76 82 97 DB CD 5B FC 00 00
9	R-APDU + SW1 SW2	<	B3 82 01 E6 69 32 76 7B 08 10 9F 88 A3 C3 68 41 70 32 84 4F 49 FC 7D 56 73 C2 D5 8D 7A 6D 46 63 63 E5 D1 C8 1D C4 13 3E 91 E0 1C 1F 4F A4 78 D0 E7 5A CC 2C 80 CE 7B 12 3F F3 17 55 A9 02 19 C2 BD AB B9 EC C3 00 2E 6E 9E 4E AB 99 BC 5D 8C 1E CF 80 E3 84 58 88 13 C6 9A F6 66 C2 0D FA E4 F6 5F 67 6E 99 9D 62 FE C3 04 67 64 83 F6 22 0A DE AA 6A 4B 68 C3 26 ED 28 77 2C 6E E9 07 4E A7 AB 77 68 E0 5D 8D 9B DD 08 9C 70 A8 4B C5 C7 4E C1 79 34 A9 8F DB 6B CB 8D DC 6D CA E1 AC 7F 66 5C 14 5E 46 BF 93 BD 9F A5 37 85 9E 49 B6 9A 2E 9A A2 65 AD 15 40 7F DA 5A 21 B3 C3 CC EB 11 FD AB BE CA CB 0A 33 34 CC 7C 52 78 53 8E 4A 35 7E E0 91 8D B7 0C 8C 43 BA F0 30 A0 DD 83 AE 67 79 AC BA A5 F8 65 6D 7E A4 D7 AC 21 12 2E 1D 7C 85 2B D5 63 B9 88 17 4E C7 68 73 E1 20 27 A6 C4 13 51 16 A8 54 ED 8C A1 46 CA 62 2B F0 DD 30 EC C3 93 FE F9 E9 6F C5 43 5B 3A AA 42 0C 8D 65 52 DF 6B C5 DD 06 78 2E 31 61 65 38 28 EB D8 E4 3E D8 1A 9D 16 92 5F A8 A9 32 C1 A1 19 60 84 78 7F 9A 06 56 52 A8 B7 25 9D 70 87 DD 42 55 C7 03 71 1F E2 C6 A1 22 4F EE D2 38 57 DE 61 FD 00 B9 41 E4 3C 06 97 D9 7F B6 93 A9 9B FD 76 03 87 A3 C9 B0 91 46 8B D8 B3 E6 F9 82 94 16 B3 28 1E 40 DF D4 63 8E E1 6E 17 8D 40 29 66 24 19 F3 72 88 B4 61 3A 59 B7 43 18 06 62 53 A2 AD AD A1 A7 B6 51 0E 51 D9 7F 36 6A EB 49 0F 28 EF B1 86 66 D2 1D DF 3A 4B 93 87 AE 26 1C 20 0D 4D 01 D4 57 71 10 EF 7D

Table 30. Host requests device's Intermediate (P1) certificate*...continued*

Step	Command		Data
			3D EB 13 40 00 88 36 A1 BD 9B FA 72 B9 6D 5F 9C F6 6A F7 35 9F DB 1B 07 6F E3 AB EC 3E 36 B0 14 9F A3 5F 7D B6 4C F9 00 0B 21 90 00
10	SIGMA-I message type, Tag and Length	=	B3 82 01 E6
11	++IV_e1	=	CF 39 FD B8 7C 9A 5A 50 AF F4 28 5C C5
12	AES_CCM_Dec(K_e1, Iv_e1, R-APDU (without B3 82 01 E6))		7F 21 82 01 D9 30 82 01 D5 30 82 01 7C A0 03 02 01 02 02 14 44 39 39 90 CA 2E 5B 63 C8 89 B6 A7 FC AA 72 2F F6 7D FB 63 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 41 75 74 68 20 52 6F 6F 74 43 41 76 45 32 30 31 30 1E 17 0D 32 34 30 36 30 34 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30 32 30 36 32 35 32 39 5A 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 50 31 20 49 6E 74 65 72 6D 43 41 76 45 32 30 31 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 59 4E 77 DF 1C 6B 27 C4 16 6E 1C FE B1 C1 D5 BB 3F B8 FE 07 6A DC 7A 61 5C D3 7E 08 15 D1 A1 92 47 DF FF C5 76 EE 73 6D 40 14 F3 5E 3D 2E 41 CE E1 06 2E 58 7A 60 48 2E 19 98 7B 88 8A A6 90 9D A3 10 30 0E 30 0C 06 03 55 1D 13 04 05 30 03 01 01 FF 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 47 00 30 44 02 20 65 BD 34 E9 5D DA 1E 20 F0 13 8D 0A 6E D0 06 05 66 C5 A0 4A 42 23 69 8E 27 29 1D A5 BA C9 B0 8E 02 20 54 91 04 B7 9D 5F 9A 38 C2 6D CF 86 AF 84 F4 01 F4 A0 D6 E5 CB 48 F4 DF 20 1D A6 14 96 38 85 D7
13	Tag (Uncompressed certificate)	=	7F 21
14	Tag and Length	=	82 01 D9
15	Device P1 Certificate (ASN1)	=	30 82 01 D5 30 82 01 7C A0 03 02 01 02 02 14 44 39 39 90 CA 2E 5B 63 C8 89 B6 A7 FC AA 72 2F F6 7D FB 63 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 41 75 74 68 20 52 6F 6F 74 43 41 76 45

Table 30. Host requests device's Intermediate (P1) certificate*...continued*

Step	Command		Data
			32 30 31 30 1E 17 0D 32 34 30 36 30 34 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30 32 30 36 32 35 32 39 5A 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 50 31 20 49 6E 74 65 72 6D 43 41 76 45 32 30 31 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 59 4E 77 DF 1C 6B 27 C4 16 6E 1C FE B1 C1 D5 BB 3F B8 FE 07 6A DC 7A 61 5C D3 7E 08 15 D1 A1 92 47 DF FF C5 76 EE 73 6D 40 14 F3 5E 3D 2E 41 CE E1 06 2E 58 7A 60 48 2E 19 98 7B 88 8A A6 90 9D A3 10 30 0E 30 0C 06 03 55 1D 13 04 05 30 03 01 01 FF 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 47 00 30 44 02 20 65 BD 34 E9 5D DA 1E 20 F0 13 8D 0A 6E D0 06 05 66 C5 A0 4A 42 23 69 8E 27 29 1D A5 BA C9 B0 8E 02 20 54 91 04 B7 9D 5F 9A 38 C2 6D CF 86 AF 84 F4 01 F4 A0 D6 E5 CB 48 F4 DF 20 1D A6 14 96 38 85 D7

4.1.1.3 Verify Responder certificate chain

Table 31. Verify Device's certificate chain

Step	Command		Data
	Device Leaf cert. verification with Device P1 cert.		
1	Extract the ECC Public key from P1 Device Certificate	=	04 59 4E 77 DF 1C 6B 27 C4 16 6E 1C FE B1 C1 D5 BB 3F B8 FE 07 6A DC 7A 61 5C D3 7E 08 15 D1 A1 92 47 DF FF C5 76 EE 73 6D 40 14 F3 5E 3D 2E 41 CE E1 06 2E 58 7A 60 48 2E 19 98 7B 88 8A A6 90 9D
2	Relevant TBSCertificate SEQUENCE of Leaf device certificate	=	30 82 01 79 A0 03 02 01 02 02 14 35 FC 39 38 34 81 15 06 DD 59 CF B9 93 03 6F FC 98 1B C9 18 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 50 31 20 49 6E 74 65 72 6D 43 41 76 45 32 30 31 30 1E 17 0D 32 34 30 36 30 34 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30 32 30 36 32 35 32 39 5A 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E

Table 31. Verify Device's certificate chain*...continued*

Step	Command		Data
			64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 4C 45 41 46 20 43 45 52 54 49 46 49 43 41 54 45 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 EB FF 86 3E 9D 53 32 6B BF E8 54 26 EC 53 21 8D 6B 49 66 0B B9 23 BE 9F 6B C4 4D A5 74 21 31 5A 4F 2B C2 84 1C C2 F3 01 EF 97 8F 15 19 90 59 D9 98 19 42 2B 4C 66 8A 8F C0 64 04 AF 73 AB 00 E6 A3 0D 30 0B 30 09 06 03 55 1D 13 04 02 30 00
3	Leaf certificate hash (Hash.CertLeaf$_{NTAG}$) - SHA256(TBSCertificate SEQUENCE)	=	B5 A4 49 BE DF B0 D4 94 32 FB C7 10 A1 DF 04 0D 59 99 98 B5 33 3D E9 4F 4A AC 03 FE 61 80 74 A9
4	Leaf certificate signature (Sig.Cert Leaf$_{NTAG}$)	=	66 3C 50 33 17 A8 C4 AB 93 08 A8 11 7F 6C 79 84 CA 83 C6 9B B7 74 00 41 9A D9 A2 0E 20 CF 2C 4C FB 31 15 99 5C 3E A6 7F 45 B8 FC 07 3C 19 13 FD 72 CF BA C6 D2 01 E3 1D 88 FD EE BD 12 82 AD 35
5	ECDSAVerify(Pub.DeviceP1Key, Hash. CertLeaf$_{NTAG}$, Sig.CertLeaf$_{NTAG}$)	=	Signature valid. Device Leaf certificate's trust is granted by Device P1 certificate.
	Device P1 cert. verification with Device Root cert.		
6	Extract the ECC Public key from Device Root Certificate	=	04 D3 F0 AE 4A ED 73 31 55 D8 67 D0 2A 82 DE 3F 7B 88 AF BD B1 93 F2 E3 B9 9B B6 AB 8F 08 3A D7 39 13 D9 84 9F E8 CC A3 57 F7 55 F2 6F F6 D9 F1 F1 42 AC F0 2A F6 BE 7E 2C 49 6C F3 CE AA 3E E4 30
7	Relevant TBSCertificate SEQUENCE of P1 device certificate	=	30 82 01 7C A0 03 02 01 02 02 14 44 39 39 90 CA 2E 5B 63 C8 89 B6 A7 FC AA 72 2F F6 7D FB 63 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 41 75 74 68 20 52 6F 6F 74 43 41 76 45 32 30 31 30 1E 17 0D 32 34 30 36 30 34 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30 32 30 36 32 35 32 39 5A 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 50 31 20 49 6E 74 65 72 6D 43 41 76 45 32 30 31 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 59 4E 77 DF 1C 6B 27 C4 16 6E 1C FE B1 C1 D5 BB 3F B8 FE 07 6A DC 7A 61 5C D3 7E 08 15 D1 A1 92 47 DF FF C5 76 EE 73 6D 40 14 F3 5E 3D 2E 41 CE E1 06

Table 31. Verify Device's certificate chain*...continued*

Step	Command		Data
			2E 58 7A 60 48 2E 19 98 7B 88 8A A6 90 9D A3 10 30 0E 30 0C 06 03 55 1D 13 04 05 30 03 01 01 FF
8	P1 certificate hash (Hash.DeviceP1) - SHA256(TBSCertificate SEQUENCE)	=	4C EE CE C4 23 D6 B7 08 1A C2 D7 1F 74 82 FD B5 0E C2 D2 8A 0B E7 ED 5F AD 33 3D 99 8D 83 01 90
9	P1 certificate signature (Sig.DeviceP1)	=	65 BD 34 E9 5D DA 1E 20 F0 13 8D 0A 6E D0 06 05 66 C5 A0 4A 42 23 69 8E 27 29 1D A5 BA C9 B0 8E 54 91 04 B7 9D 5F 9A 38 C2 6D CF 86 AF 84 F4 01 F4 A0 D6 E5 CB 48 F4 DF 20 1D A6 14 96 38 85 D7
10	ECDSAVerify(Pub.DeviceRootKey, Hash. DeviceP1, Sig.DeviceP1)	=	Signature valid. Device Leaf certificate's trust is granted by Device Root certificate.

In this example certificate depth of 2 was used. Whole chain (P1, Leaf) of certificates was verified with Device Root certificate (it's Public key).

Hint: If in the final application there is probability that Host will perform another SIGMA-I authentication to the same NTAG X DNA (UID) again, Host shall store Hash.CertLeaf$_{NTAG}$, Hash.DeviceP1, Sig.CertLeaf$_{NTAG}$, Sig.DeviceP1 and Pub.DeviceP1Key to it's cache. This way whole SIGMA-I authentication speed can be improved significantly as certificates retrival over NFC can be skipped.

4.1.1.4 Verify Responder’s Session signature

Table 32. Verify Responder’s (NTAG X DNA) Session signature

Step	Command		Data
1	Hash.CertLeaf$_{NTAG}$ - "Device Leaf certificate" hash (store it on Host for future auth.)	=	00 69 1C D9 28 6C AB 92 EB 3E 29 56 6B 22 84 7C 73 9F 98 E7 B1 10 7B 2D 7C 28 2D A3 B3 BF 81 67
2	AES-CMAC(K_m1, 0x01 || leaf_cert_hash)	=	73 44 89 45 5A BE 48 EF 54 FD 65 89 84 A3 7E D9
3	Public Key of "Device Leaf certificate" (ASN1) (pk_resp)	=	04 EB FF 86 3E 9D 53 32 6B BF E8 54 26 EC 53 21 8D 6B 49 66 0B B9 23 BE 9F 6B C4 4D A5 74 21 31 5A 4F 2B C2 84 1C C2 F3 01 EF 97 8F 15 19 90 59 D9 98 19 42 2B 4C 66 8A 8F C0 64 04 AF 73 AB 00 E6
4	Sig.CertLeaf$_{NTAG}$ - Signature of received "Device Leaf certificate" (ASN1) (sig_resp)	=	73 94 0B 69 FA CC F3 EC AF 06 41 5A 7A CF C7 AF AA E0 83 2D 68 F8 FF CB 5D 1B 38 F6 61 B1 60 98 37 FE 6D 39 41 89 55 3B 27 FA 50 37 1C 18 9B FE AE 83 7E 7E B8 F3 D1 25 C4 89 49 B6 92 EA 25 14
5	keySize_init	=	01
6	keySize_resp	=	01
7	xP (Ephemeral Initiator's Public key) - E. pubKey$_{HOST}$	=	04 55 15 64 97 59 4E 96 53 13 D5 0A C2 2C 00 C8 F2 26 D7 54 78 3E 63 5A CA 69 2F 5A 42 37 F7 9D 36 DD 56 4B 07 0C BE BB 6F 33 E0 21 91 42 95 1F C1 C7 1A 23 3C D9 16 BD 04 5C 64 51 A2 8C 47 AF DB
8	yP (Ephemeral Responder's Public key) - E.pubKey$_{NTAG}$	=	04 9B 1E D1 AA DC 69 B1 3C 4B 33 A3 79 4A E4 0B 4F 8F 76 D5 E5 09 FC A6 E5 58 C0 61 20 A1 51 4D 72

Table 32. Verify Responder's (NTAG X DNA) Session signature *...continued*

Step	Command		Data
			13 3C 1C 0B 7C 1D B6 0F 5C 59 A9 12 54 EA 92 0F C8 D9 CE 08 3D 08 1F 9A 12 71 EC E5 F6 85 1D 19
9	0x01 || keySize_init || keySize_resp || xP (E.pubKey$_{HOST}$) || yP (E.pubKey$_{NTAG}$) || AES-CMAC(K_m1, 0x01 || Hash.Cert Leaf$_{NTAG}$) - keys without compression point		01 01 01 55 15 64 97 59 4E 96 53 13 D5 0A C2 2C 00 C8 F2 26 D7 54 78 3E 63 5A CA 69 2F 5A 42 37 F7 9D 36 DD 56 4B 07 0C BE BB 6F 33 E0 21 91 42 95 1F C1 C7 1A 23 3C D9 16 BD 04 5C 64 51 A2 8C 47 AF DB 9B 1E D1 AA DC 69 B1 3C 4B 33 A3 79 4A E4 0B 4F 8F 76 D5 E5 09 FC A6 E5 58 C0 61 20 A1 51 4D 72 13 3C 1C 0B 7C 1D B6 0F 5C 59 A9 12 54 EA 92 0F C8 D9 CE 08 3D 08 1F 9A 12 71 EC E5 F6 85 1D 19 73 44 89 45 5A BE 48 EF 54 FD 65 89 84 A3 7E D9
10	SHA256(0x01 || keySize_init || keySize_ resp || xP (E.pubKey$_{HOST}$) || yP (E.pub Key$_{NTAG}$) || AES-CMAC(K_m1, 0x01 || leaf_cert_hash))	=	C0 2F 89 E9 F7 A3 7B 82 51 DF 55 23 5C D6 22 16 DD 2E 29 F4 F2 56 E2 62 70 96 86 05 D5 3D 36 B3
11	ECDSA-Verify(pk_resp, sig_resp, 0x01 || keySize_init || keySize_resp || xP (E.pub Key$_{HOST}$) || yP (E.pubKey$_{NTAG}$) || AES-CMAC(K_m1, 0x01 || leaf_cert_hash))	=	Signature verified. NTAG X DNA is authenticated.

4.1.1.5 Host sends Host's Leaf certificate Hash and Signature

Table 33. Sign session data with S.privKeyHost

Step	Command		Data
1	Host leaf cert hash (Hash.CertLeaf$_{HOST}$)	=	3D FA 2A 7F 69 90 40 E6 25 60 81 59 F6 C0 BE 68 62 86 94 54 CE BD D5 F8 5F 8F 04 3E AC 05 EF 8A
2	Host leaf cert hash ECC signature	=	94 06 66 FB FB 5B 4B BE 62 28 E3 09 BE 3E 9D 6B 31 5C 6B 6A 4E 2E 85 3A E1 6B 9D 23 01 32 09 1A DE 40 F0 63 6E 9F 4F 71 FC 38 EB 65 10 86 EA 3F 9B 1F 9D 78 16 05 03 48 88 1E 92 E3 61 33 DF 81
3	AES_CMAC(K_m1, 0x02 || Hash.Cert Leaf$_{HOST}$)	=	99 03 F8 64 7A 24 70 18 FB 0B CE C4 03 CB 68 45
4	0x02 || keysize_I || keysize_resp || E. pubKey$_{NTAG}$ || E.pubKey$_{HOST}$ || AES_ CMAC(K_m1, 0x02 || Hash.CertLeaf$_{HOST}$)	=	02 01 01 9B 1E D1 AA DC 69 B1 3C 4B 33 A3 79 4A E4 0B 4F 8F 76 D5 E5 09 FC A6 E5 58 C0 61 20 A1 51 4D 72 13 3C 1C 0B 7C 1D B6 0F 5C 59 A9 12 54 EA 92 0F C8 D9 CE 08 3D 08 1F 9A 12 71 EC E5 F6 85 1D 19 55 15 64 97 59 4E 96 53 13 D5 0A C2 2C 00 C8 F2 26 D7 54 78 3E 63 5A CA 69 2F 5A 42 37 F7 9D 36 DD 56 4B 07 0C BE BB 6F 33 E0 21 91 42 95 1F C1 C7 1A 23 3C D9 16 BD 04 5C 64 51 A2 8C 47 AF DB 99 03 F8 64 7A 24 70 18 FB 0B CE C4 03 CB 68 45
5	SHA256(0x02 || keysize_I || keysize_ resp || E.pubKey$_{NTAG}$ || E.pubKey$_{HOST}$ || AES_CMAC(K_m1, 0x02 || Hash.Cert Leaf$_{HOST}$))	=	4B B3 A0 0A 07 C0 66 6B 38 53 A8 30 53 C2 2D 58 60 1D B8 64 F0 F0 94 AE 2F AD 0A 3B 34 40 1C 85

Table 33. Sign session data with S.privKeyHost*...continued*

Step	Command		Data
6	**S.privKeyHost (must be kept in secure environment)**	=	9A 84 2A E1 A5 17 DB F9 47 A4 1C 3A 92 20 A0 12 DD 59 A6 D7 2C 2F C9 82 78 D6 3E F7 79 0F 74 46
7	ECDSA Signature = ECDSASign(**S.priv KeyHost**, SHA256(0x02 || keysize_I || keysize_resp || E.pubKey$_{NTAG}$ || E.pub Key$_{HOST}$ || AES_CMAC(K_m1, 0x02 || Hash.CertLeaf$_{HOST}$)) - will be always random, due to ECDSA algorithm	=	94 06 66 FB FB 5B 4B BE 62 28 E3 09 BE 3E 9D 6B 31 5C 6B 6A 4E 2E 85 3A E1 6B 9D 23 01 32 09 1A DE 40 F0 63 6E 9F 4F 71 FC 38 EB 65 10 86 EA 3F 9B 1F 9D 78 16 05 03 48 88 1E 92 E3 61 33 DF 81
8	Data = Hash.CertLeaf$_{HOST}$ || ECDSA Signature	=	3D FA 2A 7F 69 90 40 E6 25 60 81 59 F6 C0 BE 68 62 86 94 54 CE BD D5 F8 5F 8F 04 3E AC 05 EF 8A 94 06 66 FB FB 5B 4B BE 62 28 E3 09 BE 3E 9D 6B 31 5C 6B 6A 4E 2E 85 3A E1 6B 9D 23 01 32 09 1A DE 40 F0 63 6E 9F 4F 71 FC 38 EB 65 10 86 EA 3F 9B 1F 9D 78 16 05 03 48 88 1E 92 E3 61 33 DF 81
9	++IV_e1	=	CF 39 FD B8 7C 9A 5A 50 AF F4 28 5C C6
10	AES_CCM_Enc(K_e1, Iv_e1, Data) = C_k_i	=	9F 0C B7 EA FA 43 58 8B EB B7 79 15 CA F9 90 01 77 3A 40 3E 66 56 2B 16 34 4A E0 6E FE 30 FB 54 A2 D7 44 C1 43 DA A1 05 B8 8A B9 75 4E C5 86 AD 8D F3 72 0B 2C FF AD 63 3E 39 05 88 D4 F2 D1 D3 1A C2 9A AC C9 76 69 12 34 DB 58 FC E5 7D 40 F2 06 6D 65 78 02 89 7D 5D 54 C4 4F 30 61 32 87 68 BD A0 5C 61 5E F5 4E 01

Table 34. Host sends Host's Leaf certificate Hash and Signature

Step	Command		Data
1	SIGMA-I Message Type - MSGI_HASH_ AND_SIG (Initiator sends certificate hash (or full certificate) and signature)	=	A1
2	Length of the whole MSGI_HASH_AND_ SIG payload	=	68
3	CLA of ISOGeneralAuthenticate	=	00
4	Cmd (INS) of ISOGeneralAuthenticate	=	86
5	P1 (SIGMA-I)	=	01
6	P2 (Certificate repository)	=	01
7	Lc	=	6A
8	Le	=	00
9	C-APDU	>	00 86 01 01 6A A1 68 9F 0C B7 EA FA 43 58 8B EB B7 79 15 CA F9 90 01 77 3A 40 3E 66 56 2B 16 34 4A E0 6E FE 30 FB 54 A2 D7 44 C1 43 DA A1 05 B8 8A B9 75 4E C5 86 AD 8D F3 72 0B 2C FF AD 63 3E 39 05 88 D4 F2 D1 D3 1A C2 9A AC C9 76 69 12 34 DB 58 FC E5 7D 40 F2 06 6D 65 78 02 89 7D 5D 54 C4 4F 30 61 32 87 68 BD A0 5C 61 5E F5 4E 01 00
10	R-APDU + SW1 SW2	<	B2 0C 87 0A D3 BF 8D 87 A1 ED A1 54 26 6A 90 00

Table 34. Host sends Host's Leaf certificate Hash and Signature*...continued*

Step	Command		Data
11	encrypted cert request	=	D3 BF 8D 87 A1 ED A1 54 26 6A
12	++IV_e1	=	CF 39 FD B8 7C 9A 5A 50 AF F4 28 5C C7
13	AES_CCM_Dec(K_e1, Iv_e1, R-APDU (without B2 0C 87 0A)) - Leaf certificate request	=	80 00

4.1.1.6 Responder requests Hosts cert. chain

Table 35. Host sends Host's Leaf certificate

Step	Command		Data
1	Read Cert.Leaf$_{HOST}$	=	30 82 01 D3 30 82 01 79 A0 03 02 01 02 02 14 2E 9F 8B 0B 73 76 81 53 F6 10 0E 8E E5 24 06 49 C5 0C AA E1 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 50 31 20 49 6E 74 65 72 6D 43 41 76 45 32 30 31 30 1E 17 0D 32 34 30 36 30 34 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30 32 30 36 32 35 32 39 5A 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 4C 45 41 46 20 43 45 52 54 49 46 49 43 41 54 45 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 B9 65 B5 80 D5 2E 76 91 E6 88 01 E2 97 CD 40 77 3C B8 14 DA 20 87 38 88 7D 04 78 A2 28 F7 98 CE AE CC 47 DC 6D 46 C3 FE CA 76 36 B9 1B 6F 07 8E 22 74 88 3B C4 82 0A 40 A8 50 D8 BA 0A 0C FB 2E A3 0D 30 0B 30 09 06 03 55 1D 13 04 02 30 00 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 48 00 30 45 02 21 00 93 37 37 A3 4C 2D CA AC 5E F7 3A A6 10 22 A0 78 97 C9 7A 40 65 34 5F 66 B2 F4 85 A5 98 28 AA A4 02 20 6F A9 58 E6 B1 96 A0 12 DD 47 C1 79 FA 93 77 17 29 B3 EB 03 1D CD 12 32 28 FB 5C C0 E7 4A F7 B3
2	Asymmetric authentication Protocol Payload Encoding (uncompressed certificate)	=	7F 21
3	Tag and Length	=	82 01 D7
4	Payload for encryption	=	7F 21 82 01 D7 30 82 01 D3 30 82 01 79 A0 03 02 01 02 02 14 2E 9F 8B 0B 73 76 81 53 F6 10 0E 8E E5 24 06 49 C5 0C AA E1 30 0A 06 08 2A 86 48 CE

Table 35. Host sends Host's Leaf certificate*...continued*

Step	Command		Data
			3D 04 03 02 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 50 31 20 49 6E 74 65 72 6D 43 41 76 45 32 30 31 30 1E 17 0D 32 34 30 36 30 34 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30 32 30 36 32 35 32 39 5A 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 4C 45 41 46 20 43 45 52 54 49 46 49 43 41 54 45 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 B9 65 B5 80 D5 2E 76 91 E6 88 01 E2 97 CD 40 77 3C B8 14 DA 20 87 38 88 7D 04 78 A2 28 F7 98 CE AE CC 47 DC 6D 46 C3 FE CA 76 36 B9 1B 6F 07 8E 22 74 88 3B C4 82 0A 40 A8 50 D8 BA 0A 0C FB 2E A3 0D 30 0B 30 09 06 03 55 1D 13 04 02 30 00 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 48 00 30 45 02 21 00 93 37 37 A3 4C 2D CA AC 5E F7 3A A6 10 22 A0 78 97 C9 7A 40 65 34 5F 66 B2 F4 85 A5 98 28 AA A4 02 20 6F A9 58 E6 B1 96 A0 12 DD 47 C1 79 FA 93 77 17 29 B3 EB 03 1D CD 12 32 28 FB 5C C0 E7 4A F7 B3
5	++IV_e1	=	CF 39 FD B8 7C 9A 5A 50 AF F4 28 5C C8
6	AES_CCM_Enc(K_e1, ++IV_e1, A=NULL, Payload)	=	59 A7 40 92 E1 8A 9B 32 5C E6 9F 7D 15 13 C6 68 7A 31 61 DF 6F D6 76 FE F1 2D 6D 5C 19 B8 86 29 15 43 BB D2 FC 1D 98 41 D4 70 93 20 72 53 C2 2E 8C 51 1E 0D E5 5A 13 4A 57 80 C6 F1 46 D4 6F CB 0A 49 CD CD B7 AF 79 69 91 E9 34 0D 3E AD F6 6C 7D 88 38 48 4F C5 A5 C5 EB 3E 3F 0F 32 B1 C0 26 EA BD BD 60 83 64 A4 C1 AB 1D 72 44 6D B9 D7 B0 B9 A4 EA 4A B2 5D 15 44 19 3D 70 FA 17 93 62 DB 56 56 E4 19 39 3C 2F 1E 46 A5 16 07 C9 40 3D 9A 53 A8 E3 96 8A 7A 3E 8A AC B3 F8 27 C7 F1 4E AD 2A D4 72 00 A6 7C 0C 3A FC B4 73 6F 9C 63 3A D7 E2 12 2E FE 3A 0A 98 F9 54 3F F7 2B D5 87 85 2F D1 D3 B9 EF FF F9 26 E9 7C E8 CC 4F 91 F1 B4 A2 41 B3 D9 4C CA 23 93 96 FB 30 6D BE EA 35 83 A3 69 1C AA 58 4B 09 43 45 1F A6 E0 49 AA 96 5F 21 79 5C CC B0 47 46 27 8D 20 41 AB 9E 5E 22 4F FC C6 C1 E5 FE 1B A3 48 A1 CB B7 B7 79 2F 74 8C D5 A1 BA D8 7E FF E5 50 18 03 24 4B 01 F7 BB A8 EC 3A 49 15 CA 92 DC 0B 9E 35 DF 83 51 40 D5 06 95 85 16 11 27 7F 65 A8 5E E0 2C 3F 47 C6 60 B8 0B

Table 35. Host sends Host's Leaf certificate*...continued*

Step	Command		Data
			AA 11 43 C7 78 DD 15 35 47 A3 84 05 EA 1D CE 40 8A 18 B4 BA 98 C1 6F 57 C7 1B 96 4D AB C3 FD FD BE 43 3E AD EB 3A 88 00 DB 7E 4C 45 18 EC 48 7E 31 EE 16 2B B1 E7 E0 B5 EB 3F 5A 93 9A 40 6C 9D B4 AF 83 88 82 51 BD 0D 87 95 FC 86 F6 A1 A8 1B A9 D4 26 B6 6B F9 51 63 B0 35 F2 67 45 92 27 43 73 6E B0 84 8E 2A 71 51 3D FB B2 EC 70 C3 A3 10 BC 88 9A A8 18 82 5F C7 74 06 7E B1 25 B8 23 6C CA A6 47 0F 00 9F CA C4 CC 03 4D 9C FD FF 21 E1 A2 BB C9 43 34 62 F4 B7 32 8E B1 40 ED F8 A7 28 FF D9 2B A9
7	SIGMA-I Message Type (MSGI_CERT_REPLY)	=	A3
8	Tag	=	82
	Length	=	01 E4
9	CLA of ISOGeneralAuthenticate	=	00
10	Cmd (INS) of ISOGeneralAuthenticate	=	86
11	P1 (SIGMA-I)	=	01
12	P2 (Certificate repository)	=	01
13	Lc	=	00 01 E8
14	Le	=	00 00
15	C-APDU	>	00 86 01 01 00 01 E8 A3 82 01 E4 59 A7 40 92 E1 8A 9B 32 5C E6 9F 7D 15 13 C6 68 7A 31 61 DF 6F D6 76 FE F1 2D 6D 5C 19 B8 86 29 15 43 BB D2 FC 1D 98 41 D4 70 93 20 72 53 C2 2E 8C 51 1E 0D E5 5A 13 4A 57 80 C6 F1 46 D4 6F CB 0A 49 CD CD B7 AF 79 69 91 E9 34 0D 3E AD F6 6C 7D 88 38 48 4F C5 A5 C5 EB 3E 3F 0F 32 B1 C0 26 EA BD BD 60 83 64 A4 C1 AB 1D 72 44 6D B9 D7 B0 B9 A4 EA 4A B2 5D 15 44 19 3D 70 FA 17 93 62 DB 56 56 E4 19 39 3C 2F 1E 46 A5 16 07 C9 40 3D 9A 53 A8 E3 96 8A 7A 3E 8A AC B3 F8 27 C7 F1 4E AD 2A D4 72 00 A6 7C 0C 3A FC B4 73 6F 9C 63 3A D7 E2 12 2E FE 3A 0A 98 F9 54 3F F7 2B D5 87 85 2F D1 D3 B9 EF FF F9 26 E9 7C E8 CC 4F 91 F1 B4 A2 41 B3 D9 4C CA 23 93 96 FB 30 6D BE EA 35 83 A3 69 1C AA 58 4B 09 43 45 1F A6 E0 49 AA 96 5F 21 79 5C CC B0 47 46 27 8D 20 41 AB 9E 5E 22 4F FC C6 C1 E5 FE 1B A3 48 A1 CB B7 B7 79 2F 74 8C D5 A1 BA D8 7E FF E5 50 18 03 24 4B 01 F7 BB A8 EC 3A 49 15 CA 92 DC 0B 9E 35 DF 83 51 40 D5 06 95 85 16 11 27 7F 65 A8 5E E0 2C 3F 47 C6 60 B8 0B AA 11 43 C7 78 DD 15 35 47 A3 84 05 EA 1D CE 40 8A 18 B4 BA 98 C1 6F 57 C7 1B 96 4D AB C3 FD FD BE 43 3E AD EB 3A 88 00 DB 7E 4C 45 18 EC 48 7E 31 EE 16 2B B1

Table 35. Host sends Host's Leaf certificate*...continued*

Step	Command		Data
			E7 E0 B5 EB 3F 5A 93 9A 40 6C 9D B4 AF 83 88 82 51 BD 0D 87 95 FC 86 F6 A1 A8 1B A9 D4 26 B6 6B F9 51 63 B0 35 F2 67 45 92 27 43 73 6E B0 84 8E 2A 71 51 3D FB B2 EC 70 C3 A3 10 BC 88 9A A8 18 82 5F C7 74 06 7E B1 25 B8 23 6C CA A6 47 0F 00 9F CA C4 CC 03 4D 9C FD FF 21 E1 A2 BB C9 43 34 62 F4 B7 32 8E B1 40 ED F8 A7 28 FF D9 2B A9 00 00
16	R-APDU + SW1 SW2	<	B2 0C 87 0A B9 CC D8 6D 0B 29 07 83 78 05 90 00
17	encrypted cert request (without B2 0C 87 0A)	=	B9 CC D8 6D 0B 29 07 83 78 05
14	++IV_e1	=	CF 39 FD B8 7C 9A 5A 50 AF F4 28 5C C9
15	AES_CCM_Dec(K_e1, Iv_e1, encrypted cert request) - P1 certificate request	=	81 00

Table 36. Host sends Host's P1 certificate

Step	Command		Data
1	Read Cert.$P1_{HOST}$	=	30 82 01 D5 30 82 01 7C A0 03 02 01 02 02 14 0D 27 71 80 74 4F 98 13 2A AE 96 C1 A4 25 D4 5B 1B B1 94 E6 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 41 75 74 68 20 52 6F 6F 74 43 41 76 45 32 30 31 30 1E 17 0D 32 34 30 36 30 34 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30 32 30 36 32 35 32 39 5A 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 50 31 20 49 6E 74 65 72 6D 43 41 76 45 32 30 31 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 80 DC 77 18 53 AD 79 E7 41 5E DF 26 82 9E 6F 5A BF BE 50 90 14 7A F8 7B 46 4D 6B D6 30 A6 E5 0B 4B CF 13 B6 04 D7 62 AA E5 3E 3B 86 C1 5F DF 32 1E 1A 7B 3D 05 BE 9C A6 D8 77 7B 20 DC BA 06 7C A3 10 30 0E 30 0C 06 03 55 1D 13 04 05 30 03 01 01 FF 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 47 00 30 44 02 20 6C F3 6A 0D 16 9E 5A C7 6A 0D 7D EB 26 F3 2C 90 C4 CF 3C CF 0D 0B 48 41 AE 37 C8 3B 20 49 85 3E 02 20 69 BE 1F 18 37 C0 52 2A D9 B6 07 84 6D 2B E2 AB D1 A4 98 CA DF 45 98 48 A7 F1 24 46 7C 3C D5 03

Table 36. Host sends Host's P1 certificate*...continued*

Step	Command		Data
2	Asymmetric authentication Protocol Payload Encoding (uncompressed certificate)	=	7F 21
3	Tag and Length	=	82 01 D9
4	Payload for encryption	=	7F 21 82 01 D9 30 82 01 D5 30 82 01 7C A0 03 02 01 02 02 14 0D 27 71 80 74 4F 98 13 2A AE 96 C1 A4 25 D4 5B 1B B1 94 E6 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 41 75 74 68 20 52 6F 6F 74 43 41 76 45 32 30 31 30 1E 17 0D 32 34 30 36 30 34 30 36 32 35 32 39 5A 17 0D 33 34 30 36 30 32 30 36 32 35 32 39 5A 30 62 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 12 30 10 06 03 55 04 08 0C 09 45 69 6E 64 68 6F 76 65 6E 31 12 30 10 06 03 55 04 07 0C 09 45 69 6E 64 68 6F 76 65 6E 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 50 31 20 49 6E 74 65 72 6D 43 41 76 45 32 30 31 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 80 DC 77 18 53 AD 79 E7 41 5E DF 26 82 9E 6F 5A BF BE 50 90 14 7A F8 7B 46 4D 6B D6 30 A6 E5 0B 4B CF 13 B6 04 D7 62 AA E5 3E 3B 86 C1 5F DF 32 1E 1A 7B 3D 05 BE 9C A6 D8 77 7B 20 DC BA 06 7C A3 10 30 0E 30 0C 06 03 55 1D 13 04 05 30 03 01 01 FF 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 47 00 30 44 02 20 6C F3 6A 0D 16 9E 5A C7 6A 0D 7D EB 26 F3 2C 90 C4 CF 3C CF 0D 0B 48 41 AE 37 C8 3B 20 49 85 3E 02 20 69 BE 1F 18 37 C0 52 2A D9 B6 07 84 6D 2B E2 AB D1 A4 98 CA DF 45 98 48 A7 F1 24 46 7C 3C D5 03
5	++IV_e1	=	CF 39 FD B8 7C 9A 5A 50 AF F4 28 5C CA
6	AES_CCM_Enc(K_e1, ++IV_e1, A=NULL, Payload)	=	20 08 35 EB 19 CB 1F A8 71 C7 BE DA DF 89 7C 7A F0 1C EE 63 3C 0F D4 32 87 38 9B 3E C3 DA 8E 8A 1C C3 E3 B2 3D A5 16 AE 3E 02 74 2E 93 8F 11 FE C1 F5 F3 2C 3B 00 3F 10 BB 55 E3 DB 1B D5 60 39 43 42 C0 7D A8 F3 C1 0B 16 D4 72 5D FB D2 6C 8E C5 12 75 86 32 4F 79 18 22 E4 47 B8 48 C2 09 68 61 F9 DE 09 02 31 8D 4C 35 54 D4 58 AE 41 D4 D6 77 30 5B 1A 84 A6 37 67 29 40 0B 43 72 62 EE 33 A3 73 23 D2 E9 63 B8 A8 C1 A2 2F F1 28 50 D9 73 F3 3A 3A B6 EA 7C 93 E2 94 69 14 14 A6 21 70 9C 31 48 DC F1 54 DA A3 C2 8B 98 38 BA 13 C8 BF 4E

Table 36. Host sends Host's P1 certificate*...continued*

Step	Command		Data
			27 03 EC 89 B3 D1 14 D0 85 EF 67 A5 E2 61 7D 5A 7B BA E2 8E FE B5 C3 47 6A 80 C9 CB 58 29 2A 5D F4 85 75 47 62 BE 3C 2F 20 B4 F1 AB 01 32 E2 9D 55 0E 0A 60 BC FD F8 BE BC 24 17 74 C4 73 B6 80 D5 40 17 EE 48 42 BF 45 F1 D8 E4 77 EA 57 A5 8A 6C 8E 98 ED 31 88 48 E6 5B 62 10 AB 57 BA 23 1C F2 84 20 5B C2 B7 2A 17 70 24 30 47 B6 F0 E7 22 16 EE 26 AC 84 66 4E 78 90 AB 37 CF DA 5D 88 47 3F 60 7F 27 7E D0 12 1B 7D 7F D7 23 38 D2 92 16 C9 07 05 99 ED 94 80 C3 E6 56 52 E6 6D C4 5C 1B 6D 54 09 B8 08 E5 DE BF 9E 0B 4B 92 75 D5 B1 5C 3F 56 81 E6 97 D9 A1 D4 F9 4C AD 4E FF FF 02 57 49 63 A2 62 74 3F 59 99 B8 6C 2C A1 69 0A 74 99 86 6E B5 A2 E6 6B C5 54 92 6A 8E AB 9E A1 61 35 32 C3 FA B9 04 22 04 C3 82 F5 CA 62 F2 62 AF BC B8 8B D5 40 E9 6E B5 7E B7 A5 D0 4C 96 33 59 DF 6F 56 AE 51 3E AE FC 77 40 4B E3 1C 79 41 AA 42 C5 70 CF AB 44 C3 F6 2C 7F 35 17 D8 84 25 2C 45 69 AF D9 E5 68 11 2D 6E D6 DC 43 06 36 54 9D A1 07 44 B6 45 9A 5B
7	SIGMA-I Message Type (MSGI_CERT_REPLY)	=	A3
8	Tag	=	82
	Length	=	01 E6
9	CLA of ISOGeneralAuthenticate	=	00
10	Cmd (INS) of ISOGeneralAuthenticate	=	86
11	P1 (SIGMA-I)	=	01
12	P2 (Certificate repository)	=	01
13	Lc	=	00 01 EA
14	Le	=	00 00
15	C-APDU	>	00 86 01 01 00 01 EA A3 82 01 E6 20 08 35 EB 19 CB 1F A8 71 C7 BE DA DF 89 7C 7A F0 1C EE 63 3C 0F D4 32 87 38 9B 3E C3 DA 8E 8A 1C C3 E3 B2 3D A5 16 AE 3E 02 74 2E 93 8F 11 FE C1 F5 F3 2C 3B 00 3F 10 BB 55 E3 DB 1B D5 60 39 43 42 C0 7D A8 F3 C1 0B 16 D4 72 5D FB D2 6C 8E C5 12 75 86 32 4F 79 18 22 E4 47 B8 48 C2 09 68 61 F9 DE 09 02 31 8D 4C 35 54 D4 58 AE 41 D4 D6 77 30 5B 1A 84 A6 37 67 29 40 0B 43 72 62 EE 33 A3 73 23 D2 E9 63 B8 A8 C1 A2 2F F1 28 50 D9 73 F3 3A 3A B6 EA 7C 93 E2 94 69 14 14 A6 21 70 9C 31 48 DC F1 54 DA A3 C2 8B 98 38 BA 13 C8 BF 4E 27 03 EC 89 B3 D1 14 D0 85 EF 67 A5 E2 61 7D 5A 7B BA E2 8E FE B5 C3 47 6A 80 C9 CB 58 29 2A 5D F4 85 75 47 62 BE 3C 2F 20 B4 F1 AB 01 32 E2 9D 55 0E 0A 60 BC

Table 36. Host sends Host's P1 certificate*...continued*

Step	Command		Data
			FD F8 BE BC 24 17 74 C4 73 B6 80 D5 40 17 EE 48 42 BF 45 F1 D8 E4 77 EA 57 A5 8A 6C 8E 98 ED 31 88 48 E6 5B 62 10 AB 57 BA 23 1C F2 84 20 5B C2 B7 2A 17 70 24 30 47 B6 F0 E7 22 16 EE 26 AC 84 66 4E 78 90 AB 37 CF DA 5D 88 47 3F 60 7F 27 7E D0 12 1B 7D 7F D7 23 38 D2 92 16 C9 07 05 99 ED 94 80 C3 E6 56 52 E6 6D C4 5C 1B 6D 54 09 B8 08 E5 DE BF 9E 0B 4B 92 75 D5 B1 5C 3F 56 81 E6 97 D9 A1 D4 F9 4C AD 4E FF FF 02 57 49 63 A2 62 74 3F 59 99 B8 6C 2C A1 69 0A 74 99 86 6E B5 A2 E6 6B C5 54 92 6A 8E AB 9E A1 61 35 32 C3 FA B9 04 22 04 C3 82 F5 CA 62 F2 62 AF BC B8 8B D5 40 E9 6E B5 7E B7 A5 D0 4C 96 33 59 DF 6F 56 AE 51 3E AE FC 77 40 4B E3 1C 79 41 AA 42 C5 70 CF AB 44 C3 F6 2C 7F 35 17 D8 84 25 2C 45 69 AF D9 E5 68 11 2D 6E D6 DC 43 06 36 54 9D A1 07 44 B6 45 9A 5B 00 00
16	R-APDU + SW1 SW2	<	B4 00 90 00
17	**Mutual Authentication is complete.**	=	Great success!
14	Session keys K_e2, K_m2 can be used to send messages in established secure tunnel		

4.1.1.7 Cmd.FreeMem

Get available free memory in NTAG X DNA with command Cmd.FreeMem, using the session keys produced after successful SIGMA-I authentication.

Table 37. Get available free memory with command Cmd.FreeMem

Step	Command		Data
1	K_e2	=	41 CC A9 73 57 F4 C4 1F 12 E8 EE 68 B2 DF 8F 95
2	K_m2	=	B5 6E A9 F7 AC 25 07 18 B5 DA 68 B9 8D 45 83 8F
3	Cmd (INS) of Cmd.FreeMem	=	6E
4	CommMode	=	CommMode.MAC
5	Cmd || CmdCtr || TI [|| CmdHeader] [|| CmdData]	=	6E 00 00 00 00 00 00
6	MAC(K_m2; Cmd || CmdCtr || TI)	=	BC 7F F9 C5 3B 47 F9 0C D9 88 ED 8B C8 20 73 61
7	MACt	=	7F C5 47 0C 88 8B 20 61
8	C-APDU	>	90 6E 00 00 08 7F C5 47 0C 88 8B 20 61 00
9	R-APDU + SW1 SW2	<	60 05 00 F2 BF C2 A2 F5 FA D1 EC 91 00
10	PICC's MAC	=	F2 BF C2 A2 F5 FA D1 EC
11	RespData (LSB: 00 05 06h = 1376d Bytes of free memory)	=	60 05 00

Table 37. Get available free memory with command Cmd.FreeMem*...continued*

Step	Command		Data
12	SW1 SW2	=	91 00
13	Verify Response MAC RC (SW2) || CmdCtr++ || TI [|| RespData]	=	00 01 00 00 00 00 00 60 05 00
14	MAC($K_{SesAuthMAC}$; RC || CmdCtr || TI [|| RespData])	=	A1 F2 9A BF B7 C2 82 A2 35 F5 B3 FA 02 D1 A6 EC
15	MACt	=	F2 BF C2 A2 F5 FA D1 EC
16	Compare MACt-s (from step 10 and step 15)	=	F2 BF C2 A2 F5 FA D1 EC == F2 BF C2 A2 F5 FA D1 EC

5 Secure Dynamic Messaging (SDM)

SDM also known as Secure Unique NFC message (SUN), enables user experience in a secure and convenient way:

- A unique IC individual data is generated on each tap, e.g. an NDEF message holding a "tap unique" URL (link to a website/service)
- **Allows confidential and integrity protected data exchange, without requiring a preceding authentication**
- Attaching confidentiality and integrity protected meta-data to the URI (e.g. UID, SDM counter, custom data, CMAC)
- Reading with standard NDEF readers (standard NFC enabled mobile device), no specific mobile app is needed (just NFC tap from Home screen)

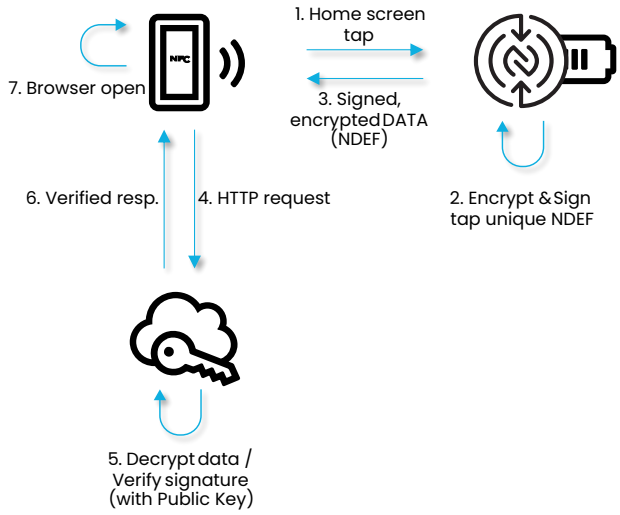

Figure 6. Secure Dynamic Messaging system example

NTAG X DNA is delivered/pre-formatted as "NFC Forum Type 4 Tag" ref.[9] already.

SDM can be activated for NDEF File inside NDEF Application only. Configured static or dynamic values are mirrored as text (ASCII encoded) into the NDEF File (0xE102), like e.g. URL NDEF message record, on each NFC tap.

NTAG X DNA creates the SDM at NFC/VCC boot-up sequence, within ISO/IEC 14443 specification ref.[2] time and H_{MIN} limits.

SDM (except SDMSIG - ECC signature) is the same as on NTAG 424 DNA. ref.[8] can be used for reference. SDMSIG Section 1.7.1 is described in this document.

5.1 Mirroring commons

1. Content is mirrored within the NDEF file (0xE102), only in non-authenticated state
2. Mirrored content (dynamic data) overlays below "place holding" content (static data)
3. Fully configurable what to mirror (UID, Counter, Part of static data, Tag Tamper status, CMAC)
4. Fully configurable where in the NDEF file to mirror
5. ASCII encoded (to represent 1 byte – 2 characters are needed)

6. Any separator of any length between mirrors can be set
7. For each mirror following has to be defined:
   - starting offset
   - length
8. Mirror starting position (**offset**) + mirror **length** must not overlap with any other enabled mirror

5.2 SDM Parameters

5.2.1 SDM Options

What can be automatically mirrored by NTAG X DNA into it's NDEF File (0xE102)?

- UID
- SDM Read Counter
- SDM Encrypted File Data
- SDM CMAC or SDM ECC Signature
- GPIO status

5.2.2 Output Mapping Examples

5.2.2.1 PICCData

- Metadata of the targeted PICC and file
- Consists of the UID (UID of the PICC) and / or the SDMReadCtr
- PICCData can be mirrored in plain or encrypted, this depends on the configuration of the access right: SDMMetaRead
- In case SDMMetaRead is set to 0xE, plain (not encrypted) mirroring is activated

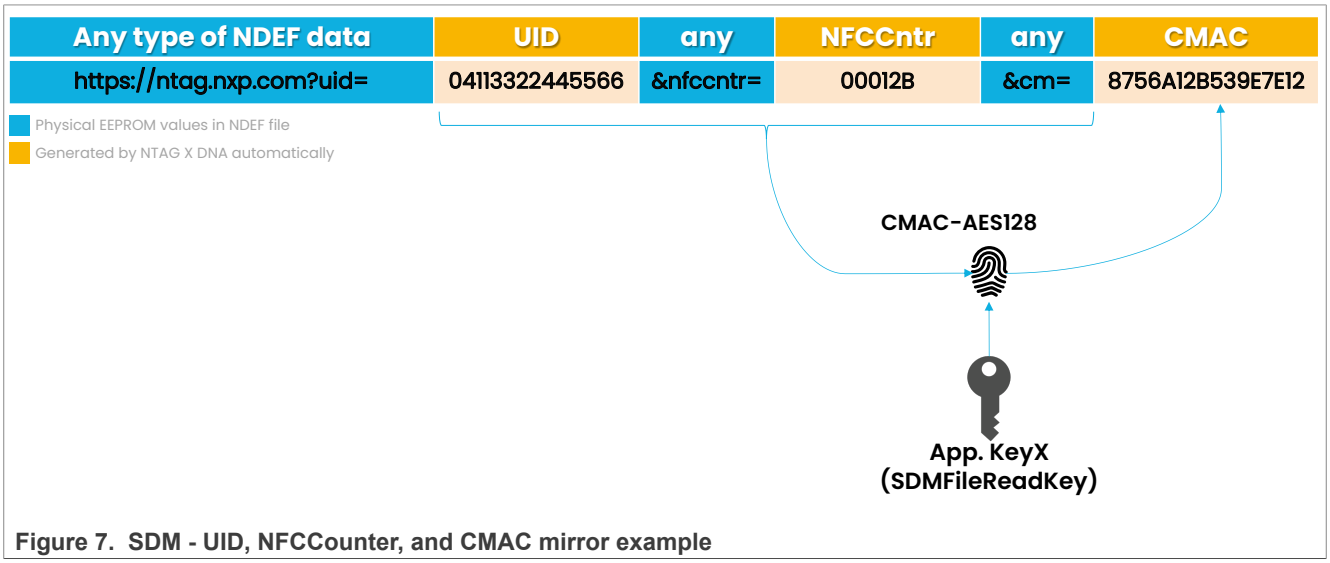

Figure 7. SDM - UID, NFCCounter, and CMAC mirror example

5.2.2.1.1 Encryption of PICCData

- Enabled when one of the Application keys is enabled for access right SDMMetaRead
- In case PICCData is morrered encrypted, the format of the plain input is:
  - PICCData = PICCDataTag || [UID] || [SDM Read Counter] || Random Padding
- Encryption and decryption of PICCData are calculated using the underlying block cipher (AES) in CBC mode as per ref.[5] using the zero byte IV

Calculation function:

PICCData = Enc($SDM_{MetaReadKey}$; PICCDataTag || [VCUID] || [SDM Read Counter] || Random Padding)

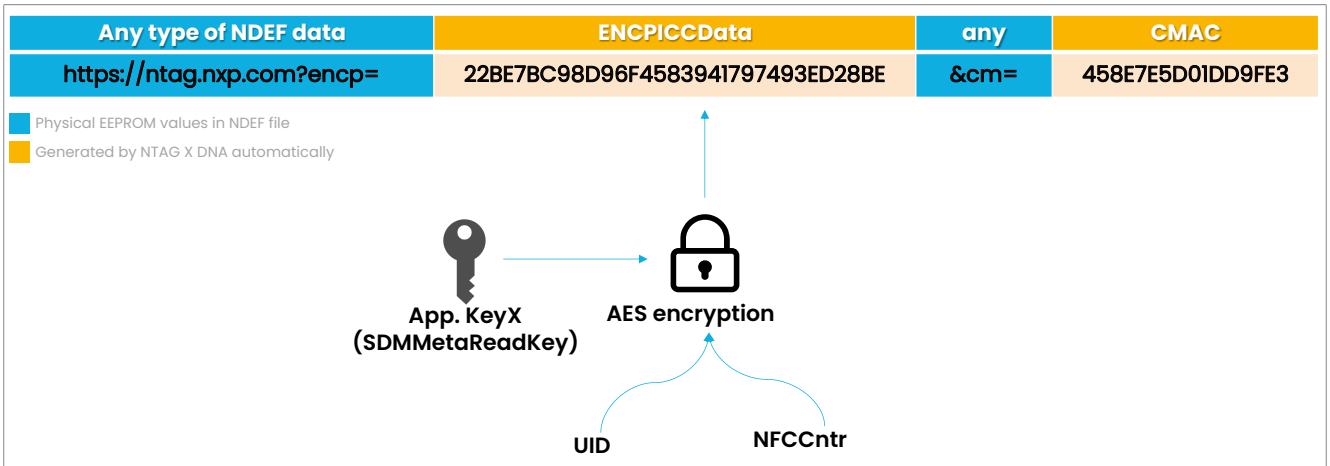

Figure 8. SDM - EncPICCData and CMAC example

5.2.2.2 SDM Read Counter

To allow replay detection by the validating party, the SDM Read Counter is associated with the NDEF file (Secure Dynamic Messaging must be enabled).

- 3-byte command (read) counter
- Allows replay detection by the backend that validates the data that has been read out already
- Enabled on the file level (NDEF File)
- SDM Counter is incremented by 1 (one) on the tag before calculating response :
  - Upon receiving Cmd.ReadData
    - If not preceded by Cmd.ISOReadBinary targeting the same file
    - Only once in case of command chaining
  - Upon receiving Cmd.ISOReadBinary
    - Same rules as for Cmd.ReadData (above)
- In cryptographic calculations represented **LSB** first
- When read out as part of NDEF File, it is returned in **ASCII** representation (**MSB** first)
- When enabled, SDMReadCtr is maintained, even if not mirrored in the PICCData
- Alternatively value of it can be retrieved with Cmd.GetFileCounters command
- Counter is reset to 0x000000 when enabling SDM with Cmd.ChangeFileSettings
- Error returned if counter value 0xFFFFFF
- Increasing counter to it's maximum value does not influence on maximum number of EEPROM programming cycles

5.2.2.2.1 SDM Read Counter Limit

- Main use case it to limit the number of reads from the tag side
- Unsigned integer of 3 bytes, related to the SDM Read Counter
- Once the SDM Read Counter equals the SDM Read Counter Limit, reading of the file with ReadData or ISOReadBinary cannot be executed without authentication
- Reading of the limit is always allowed in authenticated state

5.2.2.3 SDMMAC

- Is a MAC ref.[6] calculated over the response data
- MAC calculation is configured for a selectable application key (KeyNr00 – 04), access rights name FileAR.SDMFileRead
- SDMMAC is **mirrored** within the NDEF file when:
  - configured with Cmd.ChangeFileSettings
  - Only in non-authenticated state
- Length: 8 bytes (truncated from 16-bytes as per NIST ref.[6])
- SDM Mirror configured with:
  - SDMMACOffset: defines where calculated MAC will be mirrored to
  - SDMMACInputOffset: defines from which position in the NDEF file the MAC calculation starts

5.2.2.3.1 MAC Calculation

SDMMAC = MACt (SesSDMFileReadMACKey; DynamicFileData[SDMMACInputOffset ... SDMMACOffset - 1])

5.2.2.4 SDMENCFileData

- SDM supports mirroring full or parts of the NDEF file contents in an encrypted way
- SDMEncFileData mirroring is only done in a non-authenticated state (If authenticated, no mirroring is done)
- Encryption is done using the SDMFileRead Key, if specified

5.2.2.4.1 Encryption of SDMENCFileData

- Encryption is done using a session key (SesSDMFileReadEncKey) that is derived from the SDMFileReadKey
- Encryption and decryption of SDMEncFileData are calculated using the underlying block cipher in CBC mode of NIST SP800-38A ref.[5]

Calculation:

IV = Enc(SesSDMFileReadEncKey; SDMReadCounter || 0x00000000000000000000000000)

*SDMEncFileData = Enc(KSesSDMFileReadEncKey; StaticFileData[SDMEncOffset … SDMEncOffset + SDMEncLenght*2 -1])*

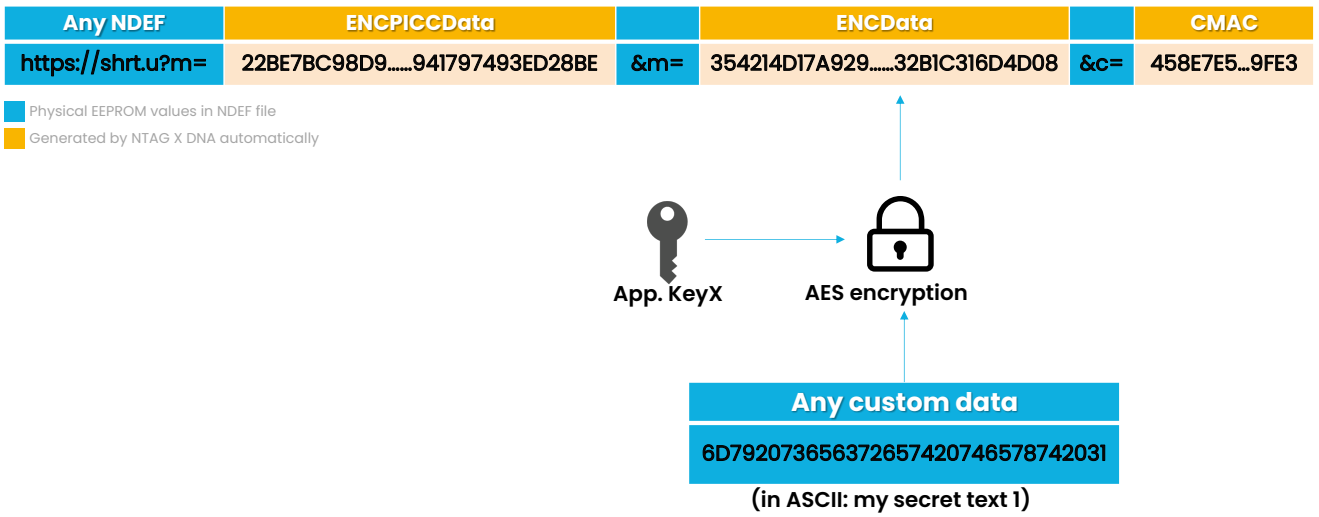

Figure 9. SDM - EncPICCData, SMDENCFileData and CMAC example

5.2.2.5 GPIOStatus

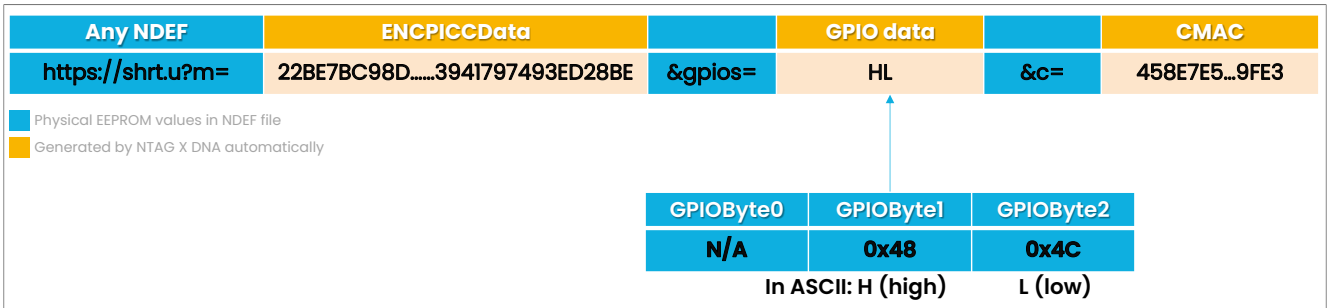

Figure 10. SDM - EncPICCData, GPIO Status and CMAC example

5.2.2.6 SDM ECC Signature (SDMSIG)

- Mirroring of ECDSA (ECC Signature) to the NDEF (SDMSIG mirroring) is only done in a non-authenticated state
- Signature is calculated using the specified ECCPrivateKey
- One of the SDMMAC or SDMSIG can be chosen (cannot co-exist)

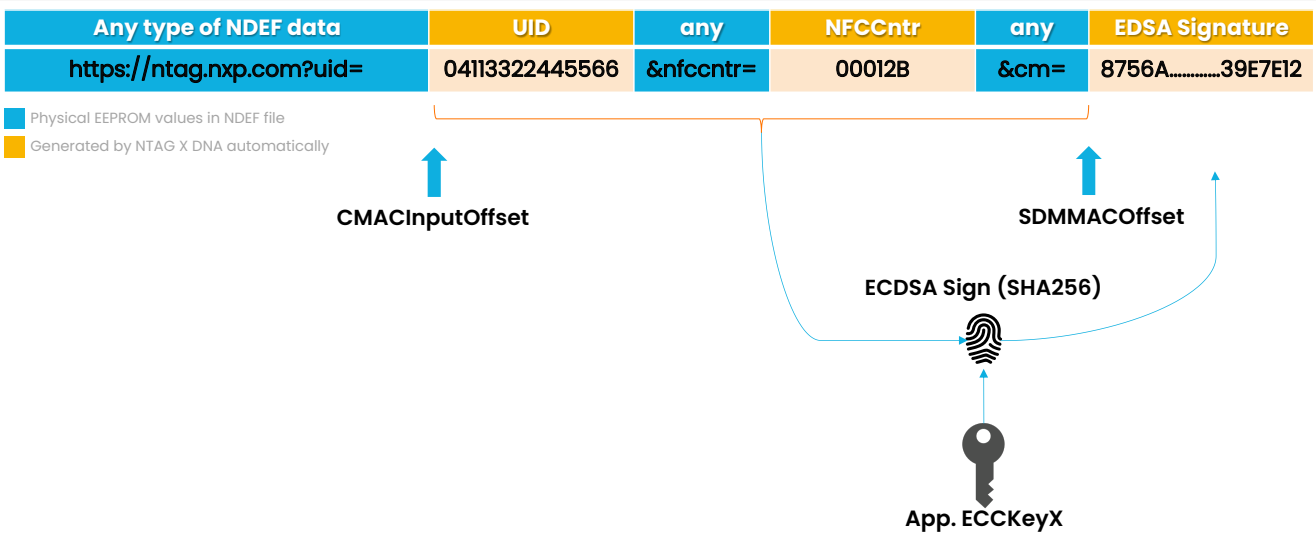

Figure 11. PICCData and SDMSIG example

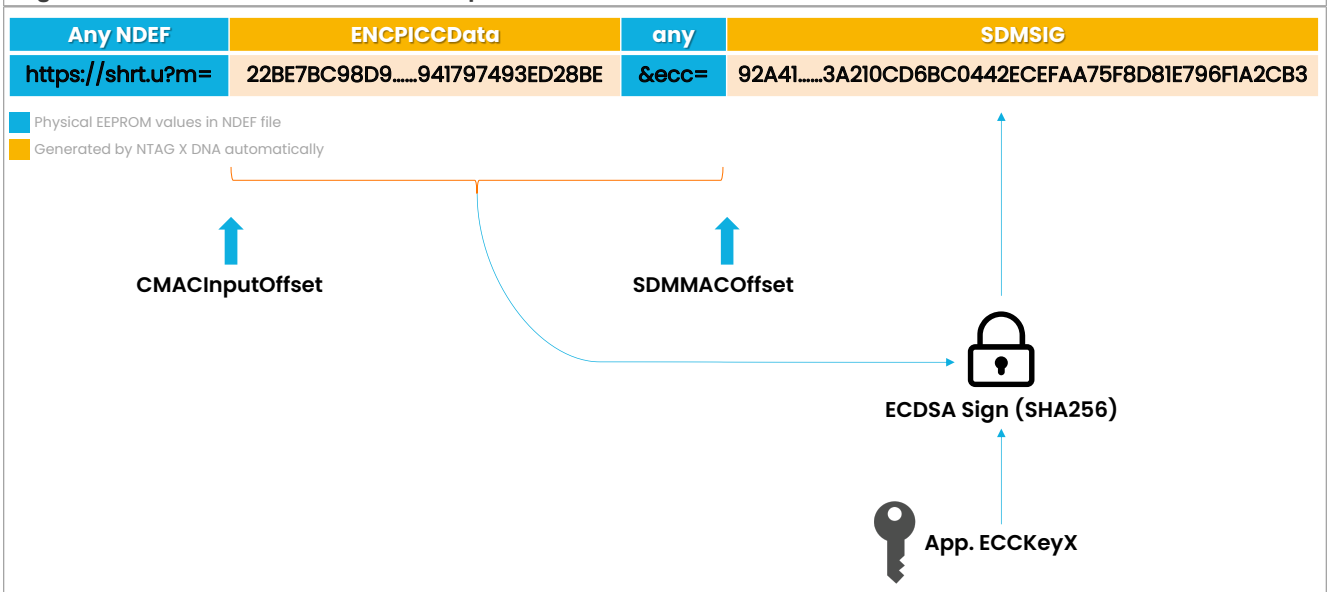

Figure 12. SDM - EncPICCData and SDMSIG example

5.2.2.6.1 Personalization for SDMSIG

An application ECCPrivateKey must be chosen for FileAR.SDMFileRead2 - required for SDMSIG mirror. SDMSIG is calculated as:

SDMSIG= ECDSASign(Priv.x, DynamicFileData[SDMMACInputOffset..SDMMACOffset− 1])

Therefore, Priv.x (ECCPrivateKey in slot x) must be provisioned by host, by NXP via Trust provisioning or generated by the NTAG X DNA itself.

1. ECC key pair is generated on the Host. Also Device Certificate is generated and signed on the Host, then programmed to NTAG X DNA. Provisioning this way, personalization / importing the private key must be done in a secure environment.
2. (highly recommended) ECC key pair is generated on the NTAG X DNA. This way private key is known only to NTAG X DNA and cannot be extracted. Secure hardening granted by CC EAL 6+. Public key part is then

used by Host to create and sign certificate outside NTAG X DNA and later programmed back as the part of Device certificate to the NTAG X DNA.

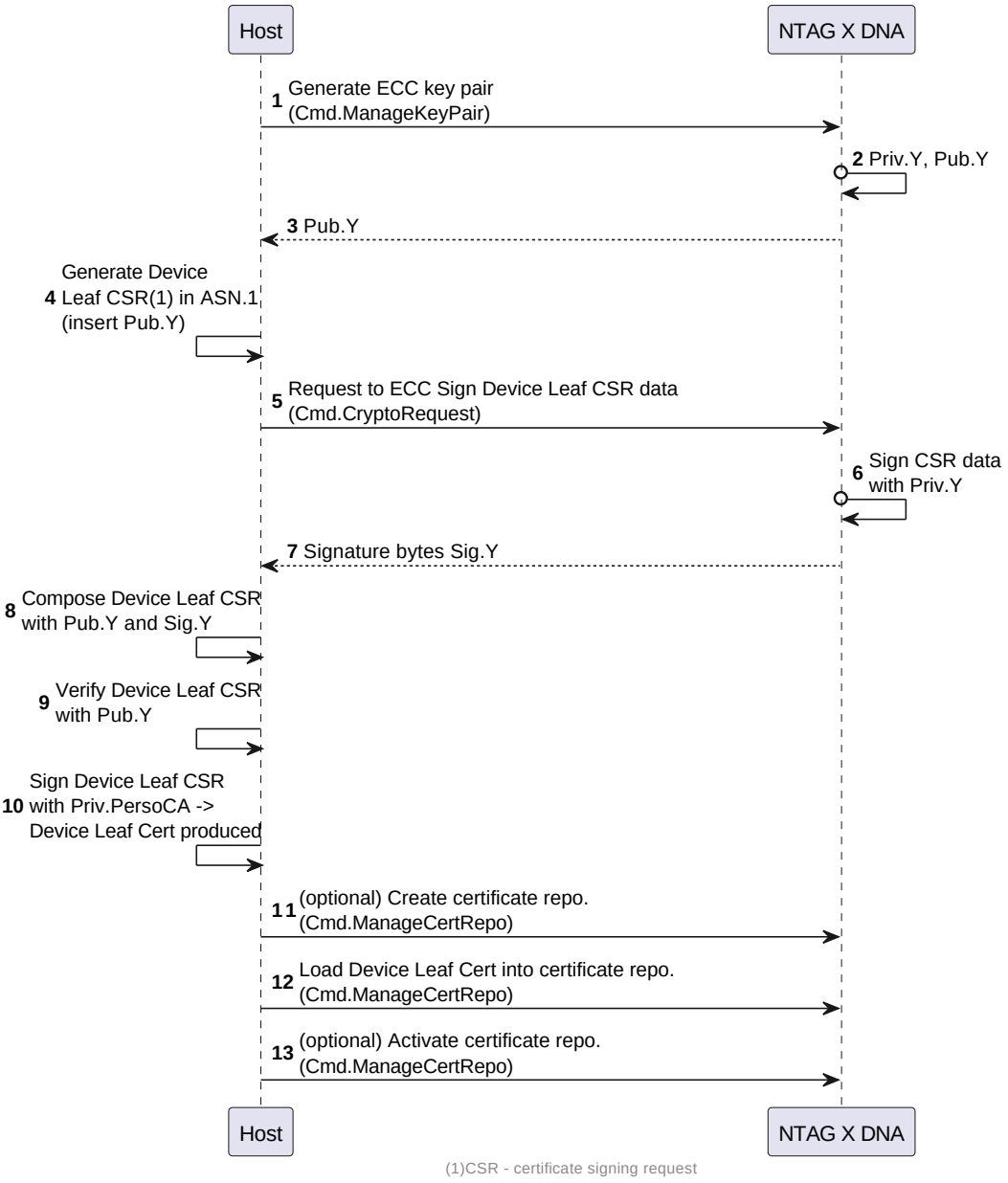

Figure 13. Asymmetric crypto provisioning

Note: Provisioning of certificates must be done in secure environment.

Desired SDM which will be configured and programmed:

https://ntag.nxp.com/xdna?m=00010203040506&c=00013B&s=5E0F5E417050E741D666AA1E4E2E51BE6ACCFF92E7AA0093DC8DDBB0276BB6AA7E1E22C42B66C7C1D0E477916FBCE499F5EE67805960B3035714132B3AD774E3

Prerequisites:

- Active Symmetric or Asymmetric authentication
- Knowledege of SDM Mirrors offsets
  - UIDOffset: 0x1B
  - SDMReadCtrOffset: 0x2C
  - SDMMACOffset: 0x35
  - SDMMACInputOffset: 0x1B (in SDMSIG calculation include everything after "m=", up to SDMMACOffset)
- ECC Key pair generated/stored on NTAG X DNA
- Correct KeyPolicy of targeted ECC Private Key (Bit5 must be set - key can be used for ECC-based Secure Dynamic Messaging. Otherwise Cmd.ChangeFileSettings may return 91 9E - Parameter Error - Targeted ECCPrivateKey for FileAR.SDMFileRead2 is not existing or not enabled for ECC-based SDM)
- Agreement on ECC curve used

1. Authenticate EV2 First (or SIGMA-I Authentication)

Table 38. Cmd.ChangeFileSettings

Step	Command		Data
1	Cmd (INS) of Cmd.ChangeFileSettings	=	5F
2	CommMode	=	CommMode.FULL (but for this example WRITE access rights were set to E (then CommMode.PLAIN can be used) for simplicity reasons).
3	CmdHeader (FileNo)	=	02
4	FileOption	=	40
5	AccessRights		EEEE
6	SDMOptions		C1
7	SDMAccessRights		E120
8	UIDOffset		1B
9	SDMReadCtrOffset		2C
10	SDMMACInputOffset		1B
11	SDMMACOffset		35
12	CmdData		40 EE EE C1 20 E1 1B 00 00 2C 00 00 1B 00 00 35 00 00
13	C-APDU	>	90 5F 00 00 13 02 40 EE EE C1 20 E1 1B 00 00 2C 00 00 1B 00 00 35 00 00 00
14	R-APDU (SW1 SW2)	<	91 00

Table 39. NDEF Message Creation

Step	Command		Data Message
1	NDEF File Content format (in ASCII)	=	https://ntag.nxp.com/xdna?m=00010203040506&c=00013B&s=5E0F5E417050E741D666AA1E4E2E51BE6ACCFF92E7AA0093DC8DDBB0276BB6AA7E1E22C42B66C7C1D0E477916FBCE499F5EE67805960B3035714132B3AD774E3
2	NDEF File Content (in Hex)	=	6E 74 61 67 2E 6E 78 70 2E 63 6F 6D 2F 78 64 6E 61 3F 6D 3D 30 30 30 31 30 32 30 33 30 34 30 35

Table 39. NDEF Message Creation*...continued*

Step	Command		Data Message
			30 36 26 63 3D 30 30 30 31 33 42 26 73 3D 35 45 30 46 35 45 34 31 37 30 35 30 45 37 34 31 44 36 36 36 41 41 31 45 34 45 32 45 35 31 42 45 36 41 43 43 46 46 39 32 45 37 41 41 30 30 39 33 44 43 38 44 44 42 42 30 32 37 36 42 42 36 41 41 37 45 31 45 32 32 43 34 32 42 36 36 43 37 43 31 44 30 45 34 37 37 39 31 36 46 42 43 45 34 39 39 46 35 45 45 36 37 38 30 35 39 36 30 42 33 30 33 35 37 31 34 31 33 32 42 33 41 44 37 37 34 45 33
3	NDEF Length + NDEF header	=	00B5 D101B15504

Table 40. Cmd.WriteData

Step	Command		Data
1	Cmd (INS) of Cmd.WriteData	=	8D
2	CommMode	=	CommMode.PLAIN
3	NDEF Length + NDEF header + NDEF File Content (in Hex)		00 B5 D1 01 B1 55 04 6E 74 61 67 2E 6E 78 70 2E 63 6F 6D 2F 78 64 6E 61 3F 6D 3D 30 30 30 31 30 32 30 33 30 34 30 35 30 36 26 63 3D 30 30 30 31 33 42 26 73 3D 35 45 30 46 35 45 34 31 37 30 35 30 45 37 34 31 44 36 36 36 41 41 31 45 34 45 32 45 35 31 42 45 36 41 43 43 46 46 39 32 45 37 41 41 30 30 39 33 44 43 38 44 44 42 42 30 32 37 36 42 42 36 41 41 37 45 31 45 32 32 43 34 32 42 36 36 43 37 43 31 44 30 45 34 37 37 39 31 36 46 42 43 45 34 39 39 46 35 45 45 36 37 38 30 35 39 36 30 42 33 30 33 35 37 31 34 31 33 32 42 33 41 44 37 37 34 45 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
4	C-APDU	>	90 8D 00 00 00 00 D6 02 00 00 00 CF 00 00 00 B5 D1 01 B1 55 04 6E 74 61 67 2E 6E 78 70 2E 63 6F 6D 2F 78 64 6E 61 3F 6D 3D 30 30 30 31 30 32 30 33 30 34 30 35 30 36 26 63 3D 30 30 30 31 33 42 26 73 3D 35 45 30 46 35 45 34 31 37 30 35 30 45 37 34 31 44 36 36 36 41 41 31 45 34 45 32 45 35 31 42 45 36 41 43 43 46 46 39 32 45 37 41 41 30 30 39 33 44 43 38 44 44 42 42 30 32 37 36 42 42 36 41 41 37 45 31 45 32 32 43 34 32 42 36 36 43 37 43 31 44 30 45 34 37 37 39 31 36 46 42 43 45 34 39 39 46 35 45 45 36 37 38 30 35 39 36 30 42 33 30 33 35 37 31 34 31 33 32 42 33 41 44 37 37 34 45 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
5	R-APDU (SW1 SW2)	<	91 00

5.2.2.6.2 Read Device Leaf certificate from NTAG X DNA repository

1. ISO14443-4 Activation (Section 3.3) - UID out of the process is 04310281D08990
2. Select NDEF Application (Section 3.6)

Table 41. Cmd.AuthenticateEV2First using Key No 0x00

Step	Command		Data Message
1	Key No	=	00
2	Key0 value	=	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3	Cmd (INS)	=	71
4	pcdCap2Len	=	06
5	PCDCap2	=	00 00 00 00 00 00
6	Command header (Key No || LenCap)	=	00 00
7	Cmd.AuthenticateFirst C-APDU	>	90 71 00 00 02 00 00 00
8	R-APDU (E(K_0, RndB) || Response Code)	<	C5 CD B7 49 1E C0 07 0C D1 01 A8 61 3B 2A E6 93 91 AF
9	E(K_0, RndB)	=	C5 CD B7 49 1E C0 07 0C D1 01 A8 61 3B 2A E6 93
10	Response Code (SW1 SW2)	=	91 AF (AF means additional frame)
11	D(K_0, RndB)	=	50 4A F2 1F 90 00 5C 29 83 F4 B1 CC AA 3F E1 CB
12	PCD generates RndA	=	E7 48 4A E2 23 1F 47 07 21 69 1D 51 40 6B 9A 3F
13	PCD prepares RndB' (rotate left by 1 byte)	=	4A F2 1F 90 00 5C 29 83 F4 B1 CC AA 3F E1 CB 50
14	RndA || RndB'	=	E7 48 4A E2 23 1F 47 07 21 69 1D 51 40 6B 9A 3F 4A F2 1F 90 00 5C 29 83 F4 B1 CC AA 3F E1 CB 50
15	E(K_0, RndA || RndB')	=	56 16 F4 AE 4F 66 54 26 5C A3 A5 B4 38 DF 97 06 2F C0 2F 40 EB 3F A1 68 80 0A 3F A6 3F 0E D4 29
16	Cmd.AuthenticatePart2 C-APDU (INS = AF)	>	90 AF 00 00 20 56 16 F4 AE 4F 66 54 26 5C A3 A5 B4 38 DF 97 06 2F C0 2F 40 EB 3F A1 68 80 0A 3F A6 3F 0E D4 29 00
17	R-APDU E(Kx, TI || RndA' || PDcap2 || PCDcap2)	<	75 D7 D0 A6 AC B6 A5 56 01 D8 93 3E 87 FC 26 A4 E9 00 8D AC 7F C9 C6 53 AF BA 86 6B A6 2B 22 13 91 00
18	E(Kx, TI || RndA' || PDcap2 || PCDcap2)	=	75 D7 D0 A6 AC B6 A5 56 01 D8 93 3E 87 FC 26 A4 E9 00 8D AC 7F C9 C6 53 AF BA 86 6B A6 2B 22 13
19	Response Code (SW1 SW2)		91 00
20	D(K_0 , TI || RndA´ || PDcap2 || PCDcap2)	=	99 C1 04 2A 48 4A E2 23 1F 47 07 21 69 1D 51 40 6B 9A 3F E7 00 00 00 00 00 00 00 00 00 00 00
21	TI (4 byte)	=	99 C1 04 2A
22	RndA´ (16 byte)	=	48 4A E2 23 1F 47 07 21 69 1D 51 40 6B 9A 3F E7
23	PDcap2 (6 byte)	=	00 00 00 00 00 00
24	PCDcap2 (6 byte)	=	00 00 00 00 00 00
25	RndA (rotate right for 1 byte)	=	E7 48 4A E2 23 1F 47 07 21 69 1D 51 40 6B 9A 3F
26	PCD compares sent RndA (from step 12) and received RndA (from step 25)	=	E7 48 4A E2 23 1F 47 07 21 69 1D 51 40 6B 9A 3F == E7 48 4A E2 23 1F 47 07 21 69 1D 51 40 6B 9A 3F

Table 41. Cmd.AuthenticateEV2First using Key No 0x00*...continued*

Step	Command		Data Message
27	SV 1 = [0xA5][0x5A][0x00][0x01][0x00] [0x80][RndA[15:14] || [(RndA[13:8] ⊕ Rnd B[15:10])] || [RndB[9:0] || RndA[7:0]	=	A5 5A 00 01 00 80 E7 48 1A A8 D1 00 D7 07 5C 29 83 F4 B1 CC AA 3F E1 CB 21 69 1D 51 40 6B 9A 3F
28	SV 2 = [0x5A][0xA5][0x00][0x01][0x00] [0x80][RndA[15:14] || [(RndA[13:8] ⊕ Rnd B[15:10])] || [RndB[9:0] || RndA[7:0]	=	A5 5A 00 01 00 80 E7 48 1A A8 D1 00 D7 07 5C 29 83 F4 B1 CC AA 3F E1 CB 21 69 1D 51 40 6B 9A 3F
29	Encryption Session Key ($K_{SesAuthENC}$) = CMAC (K_0, SV1)	=	36 42 0D 40 70 FF 7E 44 D3 51 94 90 70 25 F4 09
30	MAC Session Key ($K_{SesAuthMAC}$) = CMAC(K_0, SV2)	=	D2 A0 21 E4 6B 21 42 85 7C 7E 24 68 96 DB 59 49

3. Read by NXP pre-provisioned Application certificate (can be used for SDMSIG as well, but it needs to be enabled in KeyPolicy key meta-data):

Table 42. Cmd.ReadCertRepo

Step	Command		Data
1	Cmd (INS) of ReadCertRepo	=	4A
2	CommMode	=	CommMode.MAC (if reading meta data only. If reading certificate repository directly, access right defined at creation of repository are needed to be set)
3	CmdHeader (RepoID || Data Item)	=	00 00
4	CMD || CmdCtr || TI [|| CmdHeader] [|| E($K_{SesAuthENC}$, IVc, CmdData)]	=	4A 00 00 99 C1 04 2A 00 00
5	MAC($K_{SesAuthMAC}$; CMD || CmdCtr || TI [|| CmdHeader] [|| CmdData])	=	3D B6 B2 9A 08 8B 9D 65 8E 47 6E B3 69 8F 5E 0F
6	MACt	=	B6 9A 8B 65 47 B3 8F 0F
7	C-APDU	>	90 4A 00 00 00 00 0A 00 00 B6 9A 8B 65 47 B3 8F 0F 00 00
8	R-APDU (Data + MACt + SW1 SW2)	<	0C 8E 49 89 41 C2 E8 07 D1 D2 F3 0C F7 DE 1C CF 93 40 75 02 E2 68 2D D8 62 7D A8 B9 46 F4 BB F5 9C 32 C2 09 5E 77 CD 90 37 6A 81 8E 84 72 52 75 E1 E8 33 E8 5A 6F BE 85 52 CC CD 94 29 71 E3 32 F7 42 96 80 15 89 2D AB 3E FC C8 7F 9F 78 DD 5B 8C 42 C4 F9 32 8D 16 65 EA F6 F0 DF 7D D1 04 87 93 4E 2D 38 35 33 A3 B6 B3 C4 A5 92 29 31 8D 51 80 3D 6F 6D 0F A5 47 BB 98 10 24 91 B0 61 7E 78 26 04 B8 3D A1 08 F2 71 3B E4 CA A8 55 07 5D E8 C7 CE 62 B5 0E BD 42 BC 08 2A 1C 5A 19 42 00 D5 08 4F C1 1C 2E 0D 1E 24 5A 6D 3F F0 BE E7 39 F5 F7 57 B5 5F A1 84 1D D6 62 B6 51 39 C7 19 74 88 40 C6 37 BF 19 24 51 83 8D FB 27 53 AB AF 92 56 30 80 5A A7 6D 82 EE 08 85 5A 9D 62 5B B4 11 48 27 C0 4D B1 32 93 40 2F 3C 66 EC CF B8 3F 37 1D 7C B9 C4 8B 0D 03 67 0C 17 E6 66 10 17 52 25 66 F1 47 C1 BB 10 A6 29 3C D8 AD 90 04 C0 58 AC 08 3D EE 87 D1 D0 AC 24 2A D5 F2 88 25 5B 2D 4E 9C 4F 5E D7 2A 5E 22 1A B3 7E 47 5D 41 46 1A 85 C1

Table 42. Cmd.ReadCertRepo*...continued*

Step	Command		Data
			A7 B7 ED B3 47 21 7E 98 04 E2 BB 3E 9D 93 D3 CA EC AF A0 BC EF 85 89 41 C6 97 5B D1 D7 CC EF 28 BD EC 2B 7D 07 12 9D 56 66 24 EC A1 89 0D 8D 00 61 C7 4F AE 71 FA D2 78 5B 57 61 CE 2F B8 E2 D9 91 C3 DE 9A E1 AE D8 08 91 00
9	Verify Response MAC RC (SW2) || CmdCtr++ || TI [|| RespData]	=	00 01 00 99 C1 04 2A 0C 8E 49 89 41 C2 E8 07 D1 D2 F3 0C F7 DE 1C CF 93 40 75 02 E2 68 2D D8 62 7D A8 B9 46 F4 BB F5 9C 32 C2 09 5E 77 CD 90 37 6A 81 8E 84 72 52 75 E1 E8 33 E8 5A 6F BE 85 52 CC CD 94 29 71 E3 32 F7 42 96 80 15 89 2D AB 3E FC C8 7F 9F 78 DD 5B 8C 42 C4 F9 32 8D 16 65 EA F6 F0 DF 7D D1 04 87 93 4E 2D 38 35 33 A3 B6 B3 C4 A5 92 29 31 8D 51 80 3D 6F 6D 0F A5 47 BB 98 10 24 91 B0 61 7E 78 26 04 B8 3D A1 08 F2 71 3B E4 CA A8 55 07 5D E8 C7 CE 62 B5 0E BD 42 BC 08 2A 1C 5A 19 42 00 D5 08 4F C1 1C 2E 0D 1E 24 5A 6D 3F F0 BE E7 39 F5 F7 57 B5 5F A1 84 1D D6 62 B6 51 39 C7 19 74 88 40 C6 37 BF 19 24 51 83 8D FB 27 53 AB AF 92 56 30 80 5A A7 6D 82 EE 08 85 5A 9D 62 5B B4 11 48 27 C0 4D B1 32 93 40 2F 3C 66 EC CF B8 3F 37 1D 7C B9 C4 8B 0D 03 67 0C 17 E6 66 10 17 52 25 66 F1 47 C1 BB 10 A6 29 3C D8 AD 90 04 C0 58 AC 08 3D EE 87 D1 D0 AC 24 2A D5 F2 88 25 5B 2D 4E 9C 4F 5E D7 2A 5E 22 1A B3 7E 47 5D 41 46 1A 85 C1 A7 B7 ED B3 47 21 7E 98 04 E2 BB 3E 9D 93 D3 CA EC AF A0 BC EF 85 89 41 C6 97 5B D1 D7 CC EF 28 BD EC 2B 7D 07 12 9D 56 66 24 EC A1 89 0D 8D 00 61 C7 4F AE 71 FA D2 78 5B 57 61 CE 2F B8 E2 D9
10	MAC($K_{SesAuthMAC}$; RC || CmdCtr || TI [|| RespData])	=	FD 91 A9 C3 0A DE C5 9A 77 E1 6B AE 1C D8 5C 08
11	MACt	=	91 C3 DE 9A E1 AE D8 08
12	Compare MACt-s (from step 8 and step 11)	=	91 C3 DE 9A E1 AE D8 08 == 91 C3 DE 9A E1 AE D8 08
13	RespData (Encrypted Certificate)	=	0C 8E 49 89 41 C2 E8 07 D1 D2 F3 0C F7 DE 1C CF 93 40 75 02 E2 68 2D D8 62 7D A8 B9 46 F4 BB F5 9C 32 C2 09 5E 77 CD 90 37 6A 81 8E 84 72 52 75 E1 E8 33 E8 5A 6F BE 85 52 CC CD 94 29 71 E3 32 F7 42 96 80 15 89 2D AB 3E FC C8 7F 9F 78 DD 5B 8C 42 C4 F9 32 8D 16 65 EA F6 F0 DF 7D D1 04 87 93 4E 2D 38 35 33 A3 B6 B3 C4 A5 92 29 31 8D 51 80 3D 6F 6D 0F A5 47 BB 98 10 24 91 B0 61 7E 78 26 04 B8 3D A1 08 F2 71 3B E4 CA A8 55 07 5D E8 C7 CE 62 B5 0E BD 42 BC 08 2A 1C 5A 19 42 00 D5 08 4F C1 1C 2E 0D 1E 24 5A 6D 3F F0 BE E7 39 F5

Table 42. Cmd.ReadCertRepo*...continued*

Step	Command		Data
			F7 57 B5 5F A1 84 1D D6 62 B6 51 39 C7 19 74 88 40 C6 37 BF 19 24 51 83 8D FB 27 53 AB AF 92 56 30 80 5A A7 6D 82 EE 08 85 5A 9D 62 5B B4 11 48 27 C0 4D B1 32 93 40 2F 3C 66 EC CF B8 3F 37 1D 7C B9 C4 8B 0D 03 67 0C 17 E6 66 10 17 52 25 66 F1 47 C1 BB 10 A6 29 3C D8 AD 90 04 C0 58 AC 08 3D EE 87 D1 D0 AC 24 2A D5 F2 88 25 5B 2D 4E 9C 4F 5E D7 2A 5E 22 1A B3 7E 47 5D 41 46 1A 85 C1 A7 B7 ED B3 47 21 7E 98 04 E2 BB 3E 9D 93 D3 CA EC AF A0 BC EF 85 89 41 C6 97 5B D1 D7 CC EF 28 BD EC 2B 7D 07 12 9D 56 66 24 EC A1 89 0D 8D 00 61 C7 4F AE 71 FA D2 78 5B 57 61 CE 2F B8 E2 D9
14	5AA5 || TI || CmdCtr || 0000000000000000	=	5AA5 99C1042A 01 00 0000000000000000
15	IVr = E($K_{SesAuthENC}$, 5AA5 || TI || CmdCtr || 0000000000000000)	=	2E C6 0C BB 8A BC 04 71 84 DD BE 26 73 E4 58 70
16	D($K_{SesAuthENC}$, IVr, RespData)	=	7F 21 82 01 69 30 82 01 65 30 82 01 0B A0 03 02 01 02 02 07 04 31 02 81 D0 89 90 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 46 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 41 75 74 68 20 52 6F 6F 74 43 41 76 45 32 30 31 31 17 30 15 06 03 55 04 05 13 0E 36 33 37 30 39 33 32 30 31 30 31 30 30 31 30 1E 17 0D 32 34 31 32 31 32 30 30 30 30 30 30 5A 17 0D 34 34 31 32 31 32 30 30 30 30 30 30 5A 30 2C 31 17 30 15 06 03 55 04 0D 0C 0E 30 34 41 30 30 30 30 34 30 31 30 30 30 31 31 11 30 0F 06 03 55 04 2D 03 08 00 04 31 02 81 D0 89 90 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 C6 15 03 E8 99 29 F9 E3 43 8A D1 BB 25 AF F1 02 D9 A5 85 11 10 44 40 33 85 E5 33 AB 5F D2 20 7F 57 AB B6 0F 03 F0 94 A8 62 A1 9F 3F 41 6E F5 9D 14 69 A1 D0 95 CB F9 A4 EE B4 EF 71 54 1F F8 36 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 48 00 30 45 02 21 00 99 75 DF F0 3F C2 1D 51 96 32 6D 5A E4 D8 B7 35 B8 25 03 BD 58 F7 C6 00 C6 AB C4 FC 94 94 29 73 02 20 56 83 85 C6 1E D3 F0 45 9A 7E B5 F5 F3 73 CE CD 7D 92 E3 46 46 CC B8 DF 29 24 D6 BF 09 14 65 29 80 00
17	Remove padding and "uncompressed certificate" TLV	=	82 01 69 30 82 01 65 30 82 01 0B A0 03 02 01 02 02 07 04 31 02 81 D0 89 90 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 46 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 41 75 74 68 20 52 6F 6F 74 43 41 76 45 32 30 31 31 17 30 15 06 03 55 04 05 13 0E 36 33 37 30 39 33 32 30 31 30 31 30 30 31 30 1E 17 0D 32 34 31 32 31 32 30 30 30 30 30 30 5A 17 0D 34 34 31 32 31 32 30 30 30 30 30 30 5A 30 2C 31

Table 42. Cmd.ReadCertRepo*...continued*

Step	Command		Data
			17 30 15 06 03 55 04 0D 0C 0E 30 34 41 30 30 30 30 34 30 31 30 30 30 31 31 11 30 0F 06 03 55 04 2D 03 08 00 04 31 02 81 D0 89 90 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 C6 15 03 E8 99 29 F9 E3 43 8A D1 BB 25 AF F1 02 D9 A5 85 11 10 44 40 33 85 E5 33 AB 5F D2 20 7F 57 AB B6 0F 03 F0 94 A8 62 A1 9F 3F 41 6E F5 9D 14 69 A1 D0 95 CB F9 A4 EE B4 EF 71 54 1F F8 36 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 48 00 30 45 02 21 00 99 75 DF F0 3F C2 1D 51 96 32 6D 5A E4 D8 B7 35 B8 25 03 BD 58 F7 C6 00 C6 AB C4 FC 94 94 29 73 02 20 56 83 85 C6 1E D3 F0 45 9A 7E B5 F5 F3 73 CE CD 7D 92 E3 46 46 CC B8 DF 29 24 D6 BF 09 14 65 29

X509 presentation of programmed certificate data (step 17, Table 42)

```
TAG 82 LEN=0x1 VAL=69
SEQ(30) LEN=0x165{
    SEQ(30) LEN=0x10B{
        SEQ(A0) LEN=0x3{
            INT(02) LEN=0x1 VAL=02
        };
        INT(02) LEN=0x7 VAL=04 31 02 81 D0 89 90
        SEQ(30) LEN=0xA{
            OID(06) LEN=0x8 VAL=UNIVERSAL OID.1.2.840.10045.4.3.2
        };
        SEQ(30) LEN=0x46{
            SET(31) LEN=0xC{
                SEQ(30) LEN=0xA{
                    OID(06) LEN=0x3 VAL=organizationName
                    UTF8(0C) LEN=0x3 VAL=NXP
                };
            };
            SET(31) LEN=0x1D{
                SEQ(30) LEN=0x1B{
                    OID(06) LEN=0x3 VAL=commonName
                    UTF8(0C) LEN=0x14 VAL=NXP Auth RootCAvE201
                };
            };
            SET(31) LEN=0x17{
                SEQ(30) LEN=0x15{
                    OID(06) LEN=0x3 VAL=SerialNumber
                    String(13) LEN=0xE VAL=63709320101001
                };
            };
        };
        SEQ(30) LEN=0x1E{
            Date(17) LEN=0xD VAL=241212000000Z
            Date(17) LEN=0xD VAL=441212000000Z
        };
        SEQ(30) LEN=0x2C{
            SET(31) LEN=0x17{
                SEQ(30) LEN=0x15{
                    OID(06) LEN=0x3 VAL=UNIVERSAL OID.2.5.4.13
```

```
                    UTF8(0C) LEN=0xE VAL=04A00004010001
                };
            };
            SET(31) LEN=0x11{
                SEQ(30) LEN=0xF{
                    OID(06) LEN=0x3 VAL=UNIVERSAL OID.2.5.4.45
                    BIT STRING(03) LEN=0x8 VAL=00 04 31 02 81 D0 89 90
                };
            };
        };
        SEQ(30) LEN=0x59{
            SEQ(30) LEN=0x13{
                OID(06) LEN=0x7 VAL=UNIVERSAL OID.1.2.840.10045.2.1
                OID(06) LEN=0x8 VAL=UNIVERSAL OID.1.2.840.10045.3.1.7
            };
            BIT STRING(03) LEN=0x42 VAL=00 04 C6 15 03 E8 99 29 F9 E3 43 8A D1
 BB 25 AF F1 02 D9 A5 85 11 10 44 40 33 85 E5 33 AB 5F D2 20 7F 57 AB B6 0F 03
 F0 94 A8 62 A1 9F 3F 41 6E F5 9D 14 69 A1 D0 95 CB F9 A4 EE B4 EF 71 54 1F F8
 36
        };
    };
    SEQ(30) LEN=0xA{
        OID(06) LEN=0x8 VAL=UNIVERSAL OID.1.2.840.10045.4.3.2
    };
    BIT STRING(03) LEN=0x48 VAL=00 30 45 02 21 00 99 75 DF F0 3F C2 1D 51 96 32
 6D 5A E4 D8 B7 35 B8 25 03 BD 58 F7 C6 00 C6 AB C4 FC 94 94 29 73 02 20 56 83
 85 C6 1E D3 F0 45 9A 7E B5 F5 F3 73 CE CD 7D 92 E3 46 46 CC B8 DF 29 24 D6 BF
 09 14 65 29
};
```

Extracted data for better readability:

Subject: OID.2.5.4.45=#03080004310281D08990, Description=04A00004010001

Issuer: SERIALNUMBER=63709320101001, CN=NXP Auth RootCAvE201, O=NXP

Valid Date: 12. 12. 2024 01:00:00

Expiry Date: 12. 12. 2044 01:00:00

Serial Number: 04310281D08990

Public Key:

04C61503E89929F9E3438AD1BB25AFF102D9A585111044403385E533AB5FD2207F57ABB60F03F094A862A19F3F416EF59D1469A1D095CBF9A4EEB4EF71541FF836 [1]

Certificate's signature - Sig.Auth:

99 75 DF F0 3F C2 1D 51 96 32 6D 5A E4 D8 B7 35 B8 25 03 BD 58 F7 C6 00 C6 AB C4 FC 94 94 29 73 56 83 85 C6 1E D3 F0 45 9A 7E B5 F5 F3 73 CE CD 7D 92 E3 46 46 CC B8 DF 29 24 D6 BF 09 14 65 29

[1] Needs to be mapped with e.g. UID in the verification backend. Will be used in Section 5.2.2.6.4.

5.2.2.6.3 Verify pre-provisioned NXP Device Auth. certificate

There are two different repositories to fetch NXP Signing certificates:

Table 43. Table Caption

	Common Name	Purpose	Issuer Serial Number	
1	**NXP Auth RootCAv E201**	Pre-provisioned Application Certificate verification	**63709320101001**	**https://www.gp-ca.nxp.com/CA/getCA?caid=63709320101001**
2	NXP Orig RootCAv E201	ECC Originality Check	63709320110001	https://www.gp-ca.nxp.com/CA/getCA?caid=63709320110001

Table 44. Verify pre-provisioned

Step	Command		Data
1	Fetch NXP Host signing certificate (NXP Auth RootCAvE201)	=	curl https://www.gp-ca.nxp.com/CA/getCA?caid=63709320101001 -o NXPAuthRootCAvE201.crt
2	Extract the ECC Public key from it	=	openssl x509 -in NXPAuthRootCAvE201.crt -text -noout
3	Authentication CA Public key - Pub.Cert AuthKey (without compression point 04)	=	D4 56 5E AB FF F2 01 C2 B7 85 13 A9 4A 27 AE 2D 1D 5A 16 13 7B 89 2A D1 6C B0 73 14 4B 22 83 08 0B A4 00 B3 2E 25 D7 CB D1 FE 33 DA DF 62 FF 1D 01 B8 77 08 62 1A A6 2E 63 C5 42 41 E1 B9 33 61
4	Get TBSCertificate SEQUENCE from Cert.Auth (Table 42)	=	30 82 01 0B A0 03 02 01 02 02 07 04 31 02 81 D0 89 90 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 46 31 0C 30 0A 06 03 55 04 0A 0C 03 4E 58 50 31 1D 30 1B 06 03 55 04 03 0C 14 4E 58 50 20 41 75 74 68 20 52 6F 6F 74 43 41 76 45 32 30 31 31 17 30 15 06 03 55 04 05 13 0E 36 33 37 30 39 33 32 30 31 30 31 30 30 31 30 1E 17 0D 32 34 31 32 31 32 30 30 30 30 30 30 5A 17 0D 34 34 31 32 31 32 30 30 30 30 30 30 5A 30 2C 31 17 30 15 06 03 55 04 0D 0C 0E 30 34 41 30 30 30 30 34 30 31 30 30 30 31 31 11 30 0F 06 03 55 04 2D 03 08 00 04 31 02 81 D0 89 90 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 C6 15 03 E8 99 29 F9 E3 43 8A D1 BB 25 AF F1 02 D9 A5 85 11 10 44 40 33 85 E5 33 AB 5F D2 20 7F 57 AB B6 0F 03 F0 94 A8 62 A1 9F 3F 41 6E F5 9D 14 69 A1 D0 95 CB F9 A4 EE B4 EF 71 54 1F F8 36
5	SHA256(TBSCertificate SEQUENCE from Cert.Auth) - part which needs to be signed	=	90 17 92 C6 84 4A E1 B8 9A 67 4F 25 DF 83 BF 64 4C CE DA 84 92 29 92 DF 1A CB FE 2C 3D 82 BC B6
6	Get signature from Cert.Auth - Sig.Auth Paragraph	=	99 75 DF F0 3F C2 1D 51 96 32 6D 5A E4 D8 B7 35 B8 25 03 BD 58 F7 C6 00 C6 AB C4 FC 94 94 29 73 56 83 85 C6 1E D3 F0 45 9A 7E B5 F5 F3 73 CE CD 7D 92 E3 46 46 CC B8 DF 29 24 D6 BF 09 14 65 29
	Verify pre-provisioned certificate Cert. Auth		
7	$ECDSA_{Verify}$(Pub.CertOrigKey (step 3), SHA256(Cert. Orig) (step 5), Sig.CertOrig (step 6))	=	Signature valid.

5.2.2.6.4 Signature Verification - SDMSIG verification

- Hash function applied is SHA256

- ECDSA Signature calculation is used
- ECDSA Signature is verified using Public Key (Pub.X)

NTAG X DNA calculates SDMSIG as following:

SDMSIG = ECDSASign(Priv.x, SHA256(DynamicFileData[SDMMACInputOffset ... SDMMACOffset − 1]))

Prerequisites:
- ECDSA Digital Signature verification algorithm
- SHA256 hashing algorithm
- Knowledege of SDM Mirrors offsets
  - SDMMACOffset
  - SDMMACInputOffset
- Public key (it is stored in NTAG's certificate repository)

Key used: Public Key of provisioned application certificate Cert.App

Length [bytes]: 64

Algorithm: *ECDSAVerify(Pub.x, SHA256(DynamicFileData[SDMMACInputOffset ... SDMMACOffset − 1]))*

Output:

1. Signature Verified

Few SDM readouts:

1. http://ntag.nxp.com/xdna?m=04310281D08990&c=000007&s=89F3A92FBA72C77F76285CB0FC08E91A3FD6114B3C005B07D86A7DCC547D0E6FE6C02CBD7ABE66DAC0D2238C050DD5B2E376C296045C7161723696F226221FF0
2. http://ntag.nxp.com/xdna?m=04310281D08990&c=000009&s=C7BD625862EA8EF9E7C14AC35D6BABBFDD12B321C67F487A43E661F6D31158079D297E484B2ED8D262D94460024440DE885EB64B18868FF9A715C9416EA4886C
3. http://ntag.nxp.com/xdna?m=04310281D08990&c=00000B&s=E8F8F138BA23F8ABEA326B2231C3F23C69AAA2571B80495CB432129166AE14D566419ABEFAD2FC40A3A2246EFFC4F5E57A1C21FA705F28DFA96624647422099F

Table 45. SDMSIG verification of readout 1

Step	Command		Data
1	SDMMACInputOffset	=	1B
2	SDMMACOffset	=	35
3	*DynamicFileData[SDMMACInputOffset ... SDMMACOffset − 1]* (ASCII)	=	04310281D08990&c=000007&s=
4	DynamicFileData (hex)	=	30 34 33 31 30 32 38 31 44 30 38 39 39 30 26 63 3D 30 30 30 30 30 37 26 73 3D
5	SHA256(DynamicFileData (hex))	=	FB 22 10 19 2B 41 81 4E E3 79 FC 65 A1 03 AE EF 05 7E 8B 80 7D 47 BB A9 B5 97 DA EA 3D AA 27 62
6	Pub.X.xD	=	C6 15 03 E8 99 29 F9 E3 43 8A D1 BB 25 AF F1 02 D9 A5 85 11 10 44 40 33 85 E5 33 AB 5F D2 20 7F
7	Pub.X.yD	=	57 AB B6 0F 03 F0 94 A8 62 A1 9F 3F 41 6E F5 9D 14 69 A1 D0 95 CB F9 A4 EE B4 EF 71 54 1F F8 36
8	Signature part 1, r	=	89 F3 A9 2F BA 72 C7 7F 76 28 5C B0 FC 08 E9 1A 3F D6 11 4B 3C 00 5B 07 D8 6A 7D CC 54 7D 0E 6F

Table 45. SDMSIG verification of readout 1*...continued*

Step	Command		Data
9	Signature part 2, s	=	E6 C0 2C BD 7A BE 66 DA C0 D2 23 8C 05 0D D5 B2 E3 76 C2 96 04 5C 71 61 72 36 96 F2 26 22 1F F0
10	Curve	=	secp256r1
11	*ECDSAVerify(Pub.X,* SHA256(DynamicFile Data, Signature*)*	=	Signature VALID

5.2.3 SDM Access Rights

- **SDM Meta Read**: Encryption of the PICCData (VCUID + SDM Read Counter) using the specified key (0xE means plain PICCData mirroring, 0xF means no PICCData mirroring)
- **SDM File Read**: Encryption of the File Data using the specified key (0xE is not possible, 0xF means no File Data mirroring)
- **SDM Counter Ret**: Retrieval of the SDM Read Counter using the specified key (0xE means free retrieval, 0xF means no access to the counter)

5.2.4 SDM Session Key Generation

These keys are used only for Secure Dynamic Messaging. Session keys are discarded after the session.

Note: *These are not the same session keys as used for Standard Secure Messaging (which are generated during AuthenticateFirst or AuthenticateNonFirst).*

Prerequisites: CMAC with AES-128 cipher core

Key used: SDMFileReadKey

Length [bytes]: 16

Algorithm:

1. SV 1 = 0xC3 || 0x3C || 0x00 || 0x01 || 0x00 || 0x80 || VCUID || SDMReadCtr [|| Zero Padding]
2. SV 2 = 0x3C || 0xC3 || 0x00 || 0x01 || 0x00 || 0x80 [|| VCUID] [|| SDMReadCtr] [|| Zero Padding]
3. $K_{SesSDMFileReadENC}$ = MAC($K_{SDMFileRead}$; SV1)
4. $K_{SesSDMFileReadMAC}$ = MAC($K_{SDMFileRead}$; SV2)

Output:

1. $K_{SesSDMFileReadENC}$ (is used for calculating CMAC Section 5.2.2.3)
2. $K_{SesSDMFileReadMAC}$(is used for Encryption of SDMFileData Section 5.2.2.4)

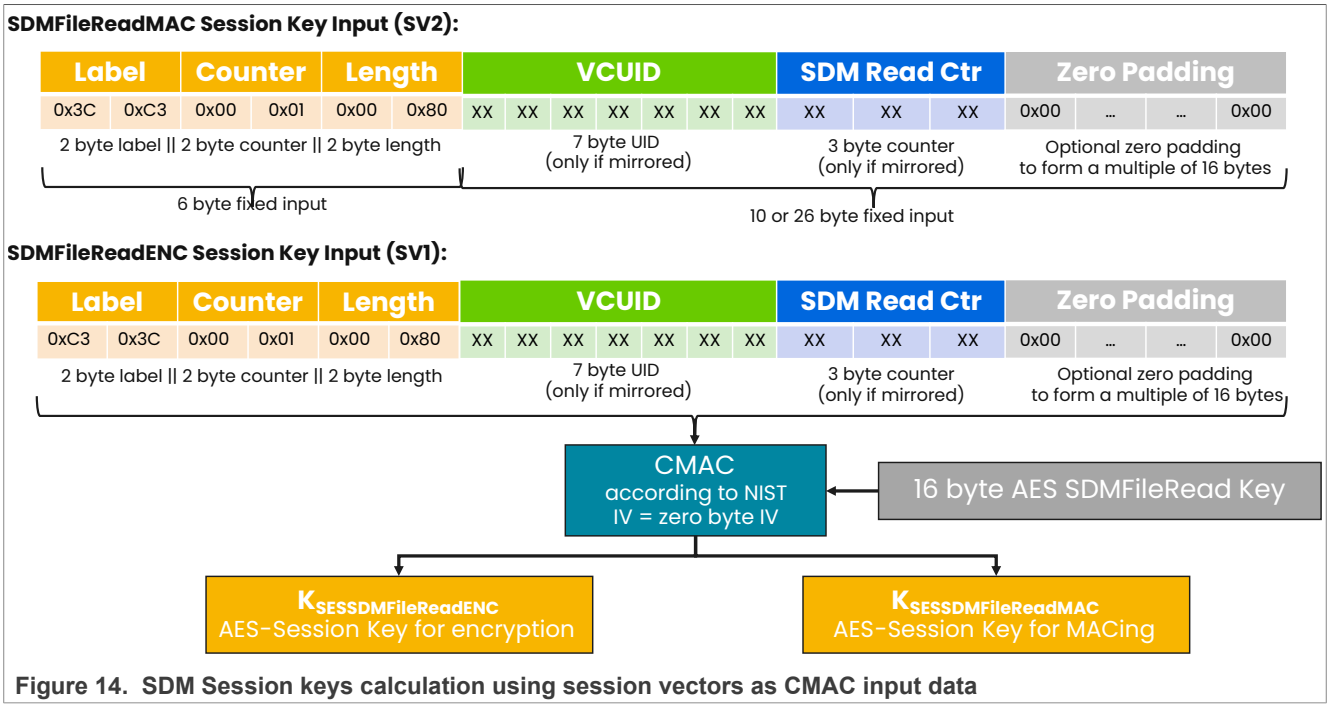

Figure 14. SDM Session keys calculation using session vectors as CMAC input data

Table 46. SDM Session Key Generation

Step	Command		Data
1	Is UID mirrored?	=	If YES, it must be included in SV calculation
2	Is SDMReadCtr mirrored?	=	If YES, it must be included in SV calculation
3	UID	=	04C767F2066180

Table 46. SDM Session Key Generation*...continued*

Step	Command		Data
4	SDMReadCtr	=	010000 (LSB first as per [Section 1.7])
5	$K_{SDMFileRead}$	=	5ACE7E50AB65D5D51FD5BF5A16B8205B
6	SV1 = C33C 0001 0080 [UID] [SDMReadCtr] [ZeroPadding] [1]	=	C33C0001008004C767F2066180010000
7	SV2 = 3CC3 0001 0080 [UID] [SDMReadCtr] [ZeroPadding]	=	3CC30001008004C767F2066180010000
8	$K_{SesSDMFileReadENC}$ = MAC($K_{SDMFileRead}$; SV1)	=	66DA61797E23DECA5D8ECA13BBADF7A9
9	$K_{SesSDMFileReadMAC}$ = MAC($K_{SDMFileRead}$; SV2)	=	3A3E8110E05311F7A3FCF0D969BF2B48

[1] In case of encrypting file data - PICCENCData, mirroring of UID and SDMReadCtr is mandatory. Therefore, both are always included in SV1 calculation.

6 Abbreviations

Table 47. Abbreviations

Acronym	Description
AES	Advanced Encryption Standard
AID	Application IDentifier
APDU	Application Protocol Data Unit
DF-Name	ISO7816 Dedicated File Name
C-APDU	Command APDU
CMAC	Cipher-based Message Authentication Code ref.[6]
CRC	Cyclic Redundancy Check
CIP	Communication Interface Parameters
DPP	Digital Passport
HD	Host device
IC	Integrated Circuit
KDF	Key derivation function
LRP	Leakage resilient primitive
LSB	Lowest Significant Byte
LSb	Lowest Significant bit
MAC	Message Authentication Code
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
NVM	Non-volatile memory
PCD	Proximity Coupling Device
PICC	Proximity Integrated Circuit Card
PRF	Pseudo Random Function
R-APDU	Response APDU (received from PICC)
SDM	Secure Dynamic Messaging (also abbreviated as SUN)
SSM	Standard Secure Messaging
SUN	Secure Unique NFC Messaging (also abbreviated as SDM)
UID	Unique IDentifier
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

7 References

[1] Specification - ISO/IEC 14443-2:2016 - Identification cards -- Contactless integrated circuit cards -- Proximity cards -- Part 2: Radio frequency power and signal interface

[2] Specification - ISO/IEC 14443-3:2018 - Identification cards -- Contactless integrated circuit cards -- Proximity cards -- Part 3: Initialization and anti-collision

[3] Specification - ISO/IEC 14443-4:2018 - Identification cards -- Contactless integrated circuit cards -- Proximity cards -- Part 4: Transmission protocol

[4] Specification - ISO/IEC 7816-4:2020 - Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange

[5] Recommendation - NIST Special Publication 800-38A - National Institute of Standards and Technology (NIST). Recommendation for BlockCipher Modes of Operation (link)

[6] Recommendation - NIST Special Publication 800-38B - National Institute of Standards and Technology (NIST). Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication (link)

[7] Document - Certicom Research. Sec 1 - Elliptic curve cryptography. Version 2.0, May 2009.

[8] Application Note - NTAG 42x DNA Features and Hints, Document number 5072**, 5244**

[9] Specification - NFC Forum: Type 4 Tag, Version 1.0 - [T4T] - 26 July 2016.

[10] Document - Globalplatform technology - apdu transport over spi / i2c - version 1.0. Version 1.0, January 2020

[11] Data sheet - N4Txxx - Secure NFC Forum T4T compliant tag

[12] Data sheet - NTAG X DNA - Secure NFC Forum T4T compliant IC with PKI (Public Interface Structure) (link)

[13] Application note - AN14137 - NTAG X DNA - Features and hints (link)

[14] Application note - AN14362 - NTAG X DNA - Energy harvesting (link)

[15] Application note - AN14513 - NTAG X DNA - Dual Interface (link)

[16] User guide - UG10083 - NTAG X DNA - Quick start guide with support package (link)

8 Note about the source code in the document

The example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

9 Revision history

Table 48. Revision history

Document rev.	Release date	Description
AN14137 v.1.0	27 May 2025	• Initial version

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Matter, Zigbee — are developed by the Connectivity Standards Alliance. The Alliance's Brands and all goodwill associated therewith, are the exclusive property of the Alliance.

Tables

Figures

Contents