API Manual

Certificate Database (Authentication Manager)

DOC221201APIV1.3.0.0EN | Revision V1.3.0.0 | English | Released | Public | 2023-07-07

empowering communication

Table of Contents

Chapter 1 Introduction

1.1 System Requirements

This software package has the following system requirements to its environment:

- netX90 (Use case C) Chip as CPU hardware platform

1.2 Intended audience

This manual is suitable for software developers with the following background:

- Knowledge of the netX DPM Interface ([1])
- Knowledge of the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (RFC-5280) ([4])

1.3 Terms, Abbreviations and Definitions

Term	Description
Default Security Configuration	Artifacts which form a configuration that a Security Component needs to be able to execute its security functionality. It contains no security policy. It can be considered less secure.
CA	Certificate Authority
Certificate	Public Key Certificate which contain identity information of an entity. Certificates are either signed by CA or they are self-signed.
CSR	Certificate Signing Request
DER	Distinguished Encoding Rules is a flexible binary encoding format used to store keys and X.509 certificates.
DPM	Dual Port Memory
EC or ECC	Elliptic Curve Cryptography is an asymmetric cryptographic method based on elliptic curves that is mostly used for key agreement and digital signatures.
EE Certificate	End-Entity Certificate is a digital signed certificate which is issued for a security component. EE certificates do not contain a Basic Constraints extension or contain the extension with the CA flag set to false. Also called Device Identity Certificate or Leaf Certificate.
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
LFW	Loadable Firmware
Operational Security Configuration	The security policy configuration which is deployed in the field by the plant operator.
PEM	Privacy Enhanced Mail is a file format used to store keys and X.509 certificates.
PKI	Public Key Infrastructure
Private Key	The Private Key is a secret key which is kept private by the owning entity (e.g. the netX device). It is used to prove towards other entities, who are presented a certificate, the identity of the owner, i.e. that the presenting entity is actually the entity for which the certificate has been issued.
Public Key	A Public Key is the counterpart to the Private Key. It is publicly presented by the entity and can thus be copied freely. The Public Key allows encryption of ciphertext which is decrypted with the corresponding private key. Public keys are typically embedded into and presented with Public Key Certificates.
RSA	Rivest-Shamir-Adleman is an asymmetric cryptographic method that its security is based on the difficulty to factorize the product of two large prime numbers in an acceptable time. RSA is used for key agreement, encryption and digital signatures.
RTE	Real Time Ethernet
Security Component	A software component that is capable of performing security functionality. Typically, the security component is part of a netX firmware running on the device.
TLS	Transport Layer Security
X.509 Certificate	A widely-used format of digital certificates that supports various encodings.

Table 1. Terms, Abbreviations and Definitions

1.4 References to documents

This document refers to the following documents:

[1]	Hilscher Gesellschaft für Systemautomation mbH: Dual-Port Memory Manual, netX Dual-Port Memory Interface, Revision 17, English, 2020.
[2]	Hilscher Gesellschaft für Systemautomation mbH: Packet API, netX Dual-Port Memory, Packet-based services (netX 90/4000/4100), Revision 5, English, 2020.
[3]	Hilscher Gesellschaft für Systemautomation mbH: Authentication Manager, User Database, Revision X, English, 2022.
[4]	RFC-5280, https://www.rfc-editor.org/rfc/rfc5280

Table 2. References to Documents

Chapter 2 Hilscher General Security Firmware

2.1 Structure of the Hilscher Security Firmware

The figure below shows the internal structure of a Hilscher LFW with security features. The LFW consists of the RTE protocol stack (green highlighted components), optional Network Services (yellow highlighted components) and the security relevant components (red highlighted components).

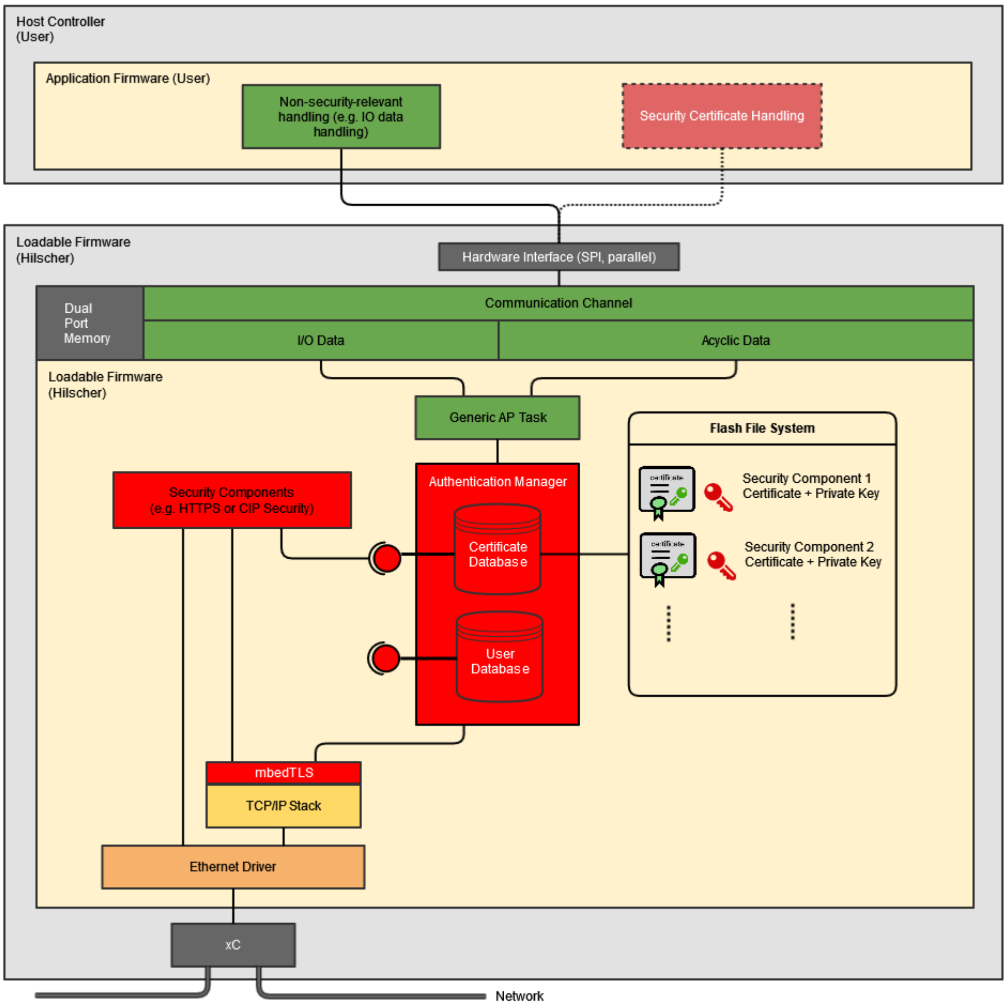

Figure 1. Hilscher Security Firmware Structure

In the following only the security related components are briefly described.

The **Security Components** use the mbedTLS stack and the Certificate Database to implement security features (e.g. CIP Security or HTTPS)

The **mbedTLS** is a lightweight open source security library which provides SSL/TLS functionality and symmetric/asymmetric cryptographic building blocks. The security components use this library to implement security functionality. This library is firmware internal and not available to the application.

The **Authentication Manager** provides the Certificate Database API for key and certificate management and the User Database API for user access management [3]. The services are available via DPM communication channel.

2.2 Security Resources Organization

The following entity relationship diagram explains the security resources available for each Security Component in the firmware. The red highlighted security resources form the Default Security Configuration. The blue highlighted security resources form the non-default Security configuration (in the following called Operational Security Configuration).

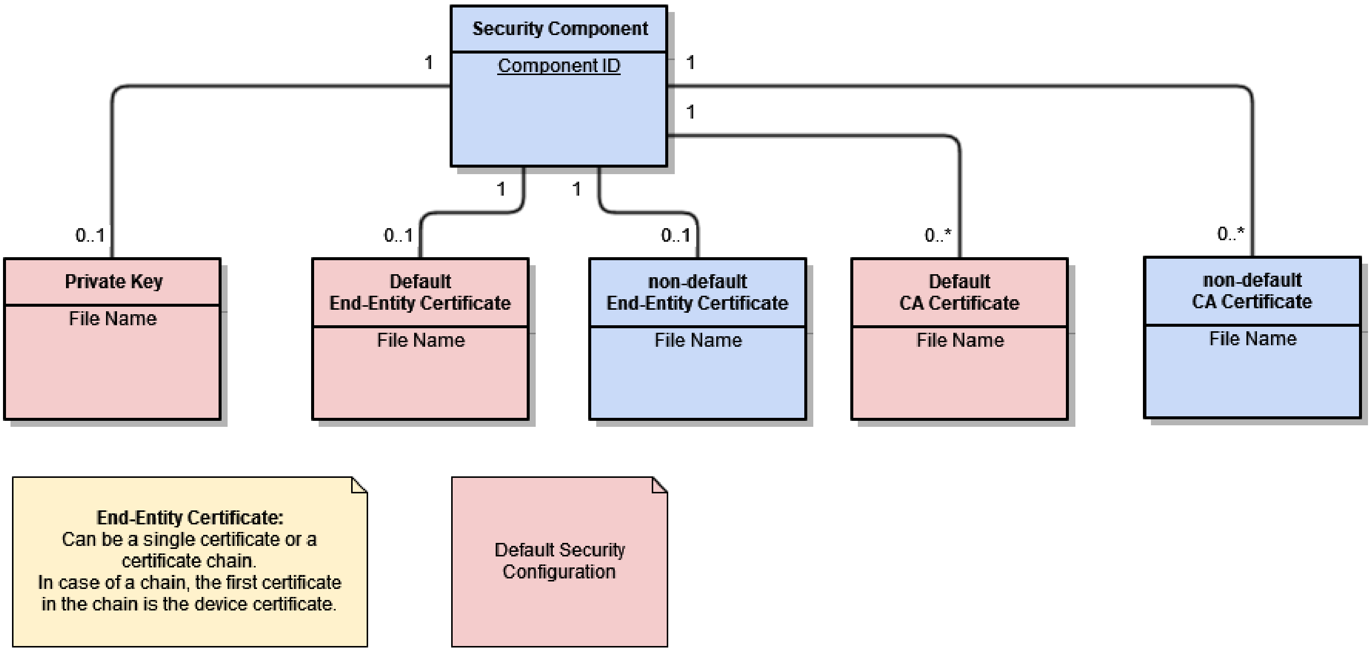
Security Component
Component ID
1
1
1
1
1
0..1
0..1
0..1
0..*
0..*
Private Key
File Name
Default End-Entity Certificate
File Name
non-default End-Entity Certificate
File Name
Default CA Certificate
File Name
non-default CA Certificate
File Name
End-Entity Certificate: Can be a single certificate or a certificate chain. In case of a chain, the first certificate in the chain is the device certificate.
Default Security Configuration

Figure 2. Organization of Security Resources Overview

The particular security resource is described in the following chapters.

2.2.1 Private Key

The private key that belongs to a security component. Each security component can have one private key. The key is used in following scenarios:

- Symmetric key exchange during TLS handshake
- Generate CSR and sign

The private is stored in the Certificate Database as single encrypted file and only the firmware internal components can read its content.

2.2.2 End-Entity (EE) Certificate

The certificate that a security component presents during TLS handshake in order to prove its identity. Each security component can have one default and one operational EE Certificate.

The EE certificate is stored as file in the Certificate Database and can be accessed using the Hilscher filesystem API [2].

2.2.3 Certification Authority (CA) Certificates

A hierarchical chain of trust to prove that a certificate has been approved by a trusted CA. The chain can contain several certificates (A root and a chain of intermediate certificates) which all are CA certificates. Each security component can have one or more default and operational CA certificates.

The CA certificates are stored as files in the Certificate Database and can be accessed using the Hilscher filesystem API [2].

2.3 Default Security Configuration

The Default Security Configuration is the minimal configuration that each security component in the firmware needs in order to be able to execute security functionality.

The Default Security Configuration consist of the following artifacts:

- Private Key: One private key for each security component must be configured. The private key is kept secret and no explicit services are available to extract the key from the device. The application can download the key to the device or trigger the creation of the key on the device.
- Default Certificate: One default certificate for each security component can be configured. The application can download the security component's default certificate or trigger the creation of the default certificate on the device.
- Default CA certificate or CA chain (optional)

The Certificate Database component of the Authentication Manager provides the API to create the Default Security Configuration. This can be done during commissioning and it is possible to re-create/overwrite a possible existing configuration. Please refer to section Commissioning of the Default Security Configuration for further details.

In general, the security component decides in which case the Default Security Configuration is valid or not. In case it is not valid, the security component is not able to execute security functionality.

Following points the security component may consider when checking the Default Security Configuration for completeness and consistency.

- The Private Key has the expected cryptographic format (e.g. RSA or EC)
- The Private Key has the expected (minimum) bit length (e.g. 256 bits, 512 bits, 2048 bits, ...)
- The Default EE Certificate is based on the Private Key
- The Default EE Certificate, in conjunction with the CA chain, is verifiable, i.e. a valid certificate chain can be constructed from it, which terminates in a (self-signed) root certificate

NOTE Please refer to the description of the respective security component in order to obtain the specific conditions on the Default Security Configuration.

Chapter 3 Getting Started/Configuration

3.1 Host application behavior

The following activity diagram shows how to integrate the step "Create Default Security Configuration" into the common host application behavior.

NOTE Please refer to the description of the respective security component in order to obtain the specific conditions on this step.

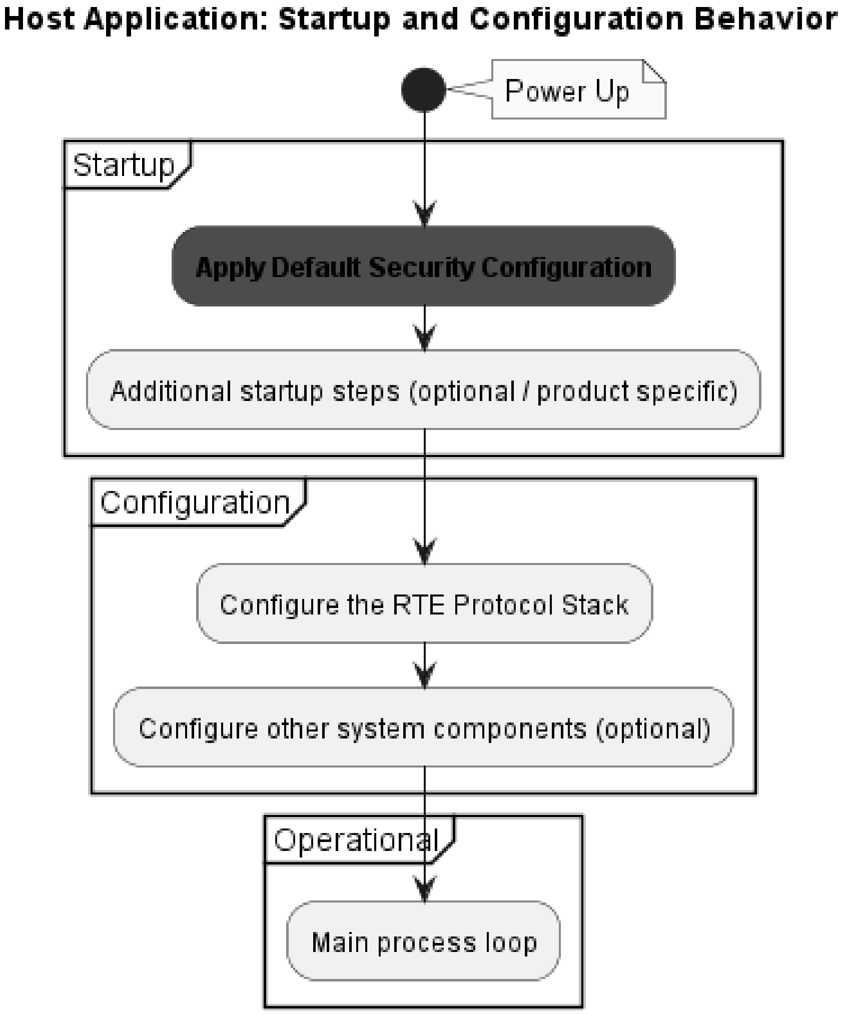
Host Application: Startup and Configuration Behavior
Power Up
Startup
Apply Default Security Configuration
Additional startup steps (optional / product specific)
Configuration
Configure the RTE Protocol Stack
Configure other system components (optional)
Operational
Main process loop

Figure 3. Host application: Startup and Configuration Behavior

In the step "Create Default Security Configuration" the host application shall provide the Default Security Configuration for the security components. Since the configuration is stored non-volatile, it needs to be applied only once. The procedure is described in section Commissioning of the Default Security Configuration.

3.2 Commissioning of the Default Security Configuration

The goal of the commissioning is to apply a default security configuration into the device, one for each security component. The application can chose between different use cases which are explained in the following table.

Use case	Description	Key		EE/CA Certificate		Recommendation
		netX	Ext PKI	netX	Ext PKI	
1: Generate Key and Certificates on netX	The key and certificates are generated on the device.	✓	-	✓	-	- PKI and CA not available - Self-signed EE certificate is sufficient
2: Generate Key on netX and Download Certificates	The key is generated on the device and the external EE and CA certificates are provided by application.	✓	-	-	✓	- PKI available - Key management not required - EE certificate signed with external PKI - CA certificate required
3: Download Key and Generate Certificates on netX	The application downloads the key to the device and generates the self-signed EE certificate on the device.	-	✓	✓	-	- PKI not available - Key management on application side is required - Self-signed EE certificate is sufficient
4: Download Key and Certificates	The key and certificates are downloaded to the device by application.	-	✓	-	✓	- PKI available - Key management on application side is required - EE certificate signed with external PKI - CA certificate required

Table 3. Commissioning Use Cases of Default Security Configuration

The commissioning use cases from table Commissioning Use Cases of Default Security Configuration are summarized in the following activity diagram. It shows the additional steps required by the application in order to apply the private key and certificates for a particular security component. Each use case is described in detail in the following sections.

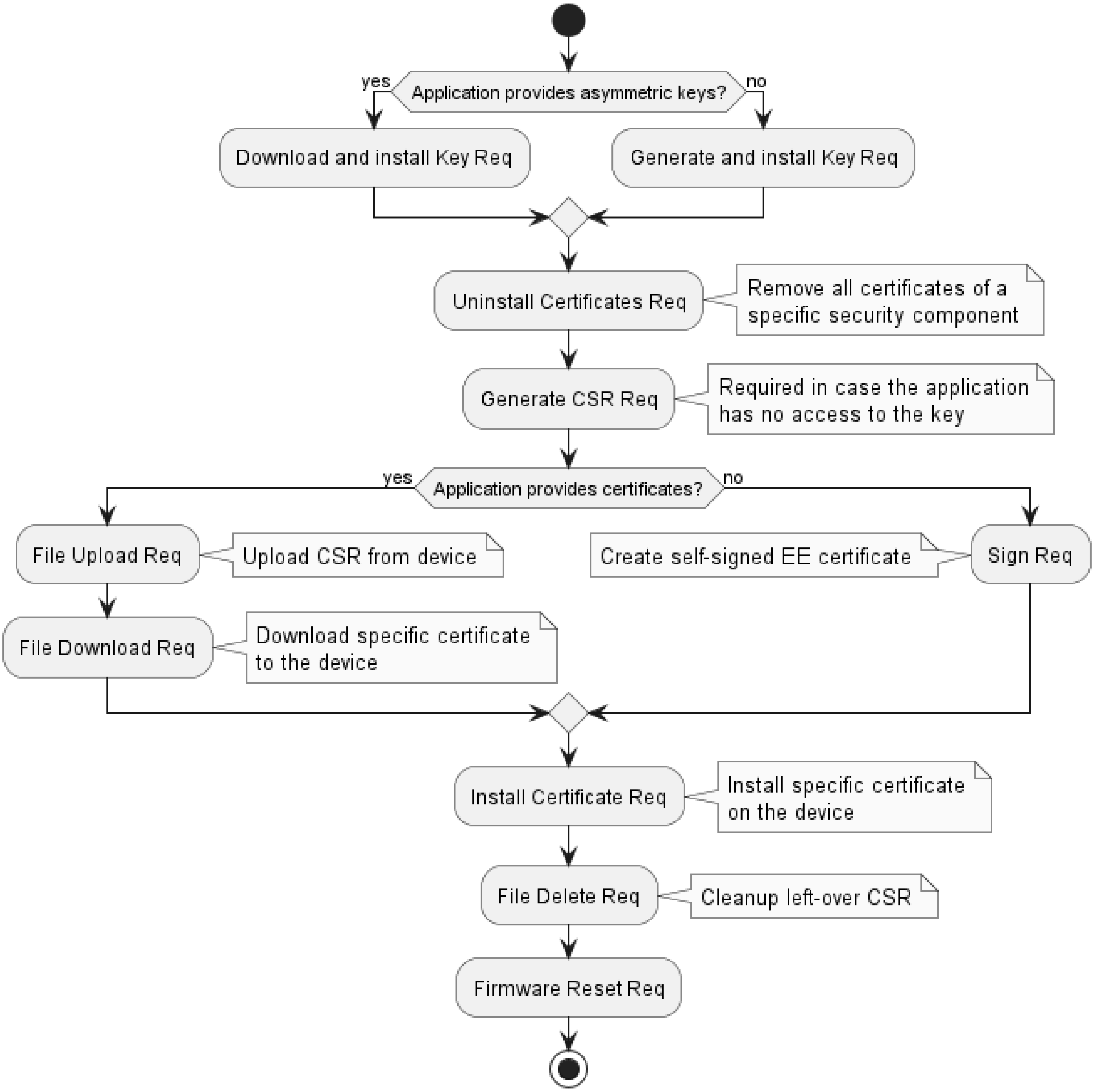
Application provides asymmetric keys?
yes
no
Download and install Key Req
Generate and install Key Req
Uninstall Certificates Req
Remove all certificates of a specific security component
Generate CSR Req
Required in case the application has no access to the key
Application provides certificates?
yes
no
File Upload Req
Upload CSR from device
Create self-signed EE certificate
Sign Req
File Download Req
Download specific certificate to the device
Install Certificate Req
Install specific certificate on the device
File Delete Req
Cleanup left-over CSR
Firmware Reset Req

Figure 4. Commissioning of the Default Security Configuration

3.2.1 Configuration packets

Packet name	Definition	Command Code
Generate and install Key	AUTHMGR_CRTDB_GENERATE_AND_INSTALL_KEY_REQ	0xB080/0xB081
Download and install Key	AUTHMGR_CRTDB_DOWNLOAD_AND_INSTALL_KEY_REQ	0xB090/0xB091
Generate CSR	AUTHMGR_CRTDB_GENERATE_CSR_REQ	0xB082/0xB083
Sign a Certificate Signing Request	AUTHMGR_CRTDB_SIGN_REQ	0xB084/0xB085
Install Certificate	AUTHMGR_CRTDB_INSTALL_CERT_REQ	0xB086/0xB087
Uninstall Certificates	AUTHMGR_CRTDB_UNINSTALL_ALL_CERTS_REQ	0xB088/0xB089
File Upload 1)	HIL_FILE_UPLOAD_REQ	0x1E60/0x1E61
File Download 1)	HIL_FILE_DOWNLOAD_REQ	0x1E62/0x1E63
File Delete 1)	HIL_FILE_DELETE_REQ	0x1E6A/0x1E6B
Firmware Reset 1)	HIL_FIRMWARE_RESET_REQ	0x1E00/0x1E01
1) Please consult the DPM Packet API manual [2] for the description of the system services.		

Table 4. Default Security Configuration Packets

3.2.2 Use case 1: Generate Key and Certificates on netX

The asymmetric keys and certificates are created on the device. The configuration sequence is illustrated in the following figure.

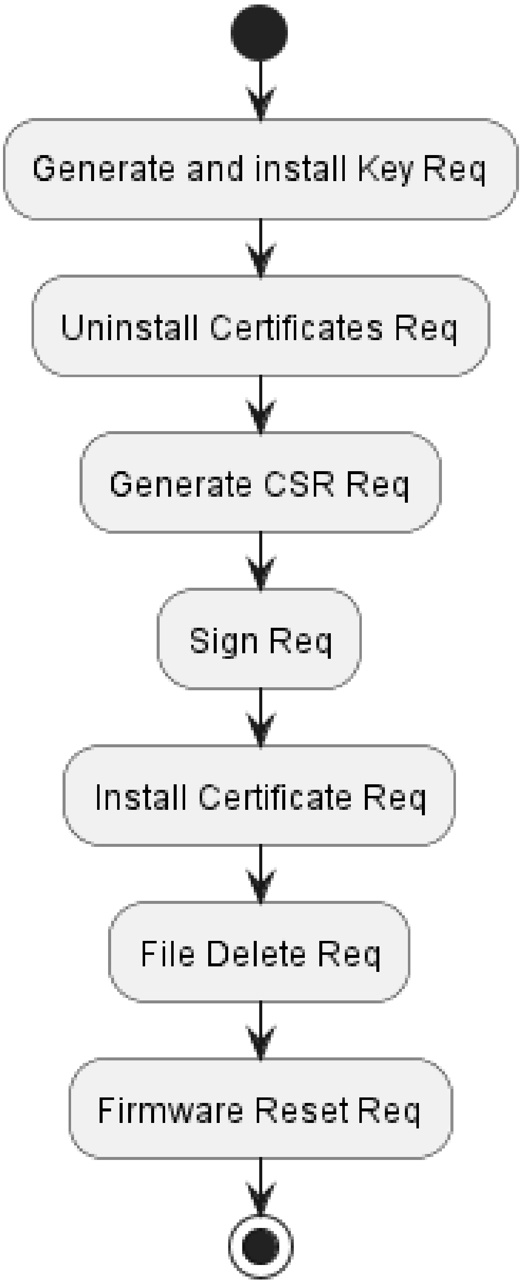

Figure 5. Generate Key and Certificates on the Device

The application uses the Generate and Install Key Request to trigger the asymmetric key generation on the device for a specific security component. Subsequently, the Uninstall all certificates request service needs to be issued for each security component when installing new certificates. The service also ensures that all old certificates related to the security component are uninstalled, since they become useless with the newly generated key.

In the following, the newly installed key is used to create and install a self-signed EE certificate on the device. For that purpose, the application triggers the CSR generation by using the Generate CSR Request. The CSR in combination with the Sign Request is used to generate the certificate. Finally, the certificate is installed by using the Install Certificate Request service.

The left-over CSR is not required anymore and can be removed from the filesystem with the File Delete Request. After a firmware reset the default security configuration is ready to be used.

3.2.3 Use case 2: Generate Key on netX and Download Certificates

The asymmetric keys are generated on the device and the EE and CA certificates are provided by the application.

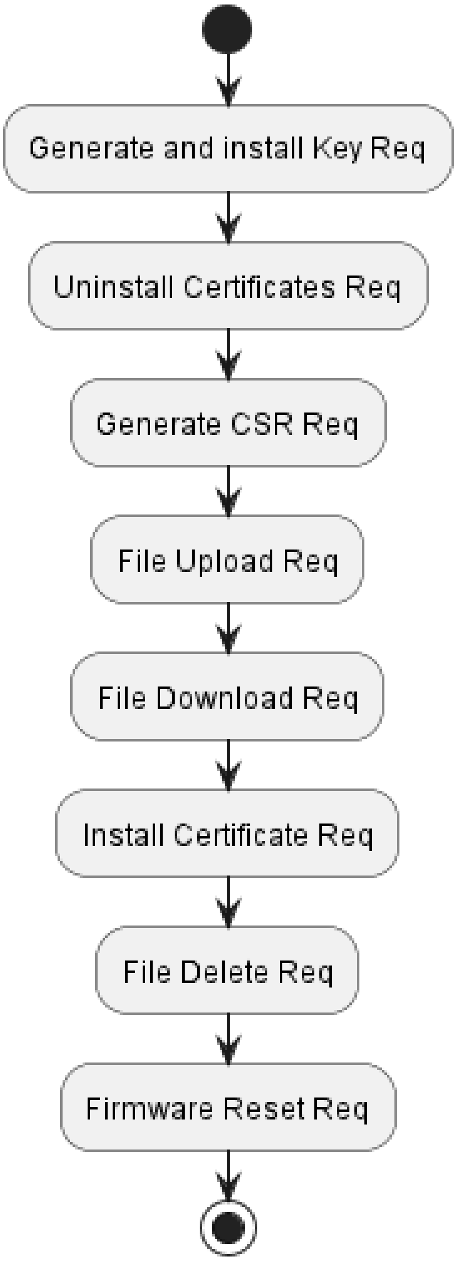

Figure 6. Generate Key on the Device and Download Certificates

The application uses the Generate and Install Key Request to trigger the asymmetric key generation on the device for a specific security component. Subsequently, the Uninstall all certificates request service needs to be issued for each security component when installing new certificates. The service also ensures that all old certificates related to the security component are uninstalled, since they become useless with the newly generated key.

In the following, the application provides the EE certificate and the CA certificate. For that purpose, the application triggers the CSR generation on the device by using the Generate CSR Request. The application obtains the CSR with the File Upload Request service in order to create the certificates. The certificates are downloaded to the device with File Download Request. Finally, the Install Certificate Request service has to be called for each of the certificates.

The left-over CSR is not required anymore and can be removed from the filesystem with the File Delete Request. After a firmware reset the default security configuration is ready to be used.

3.2.4 Use case 3: Download Key and Generate Certificates on netX

The asymmetric keys are provided by the application and the self-signed EE certificate is generated on the device.

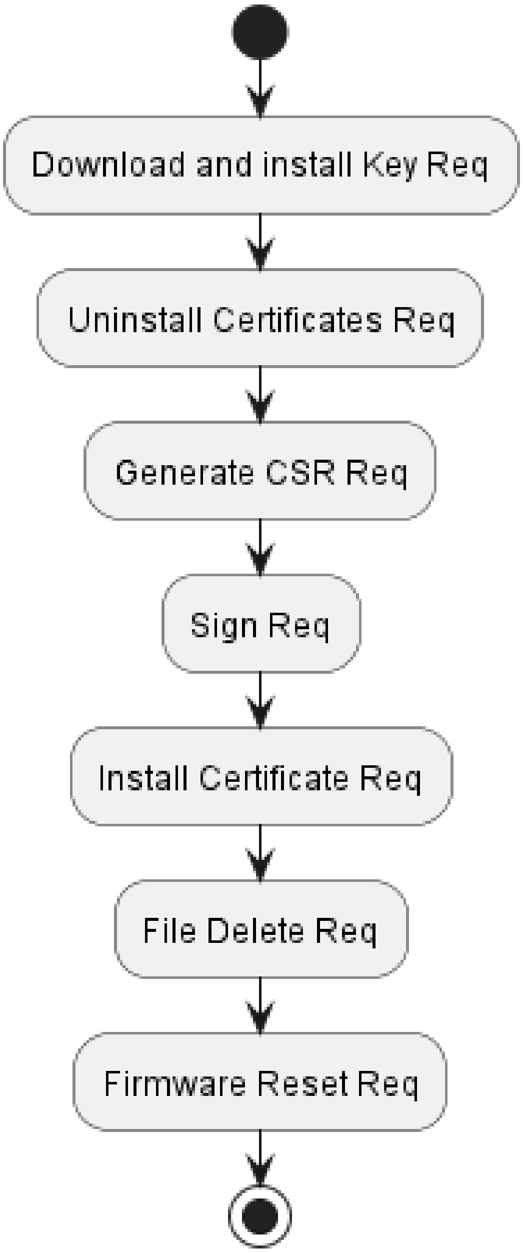

Figure 7. Download Key and Generate Certificates on the Device

The application uses the Download and Install Key Request to download and install an asymmetric key for a specific security component on the device. Subsequently, the Uninstall all certificates request service needs to be issued for each security component when installing new certificates. The service also ensures that all old certificates related to the security component are uninstalled, since they become useless with the newly generated key.

In the following, the application key is used to create and install a self-signed EE certificate on the device. For that purpose, the application triggers the CSR generation by using the Generate CSR Request. The CSR in combination with the Sign Request is used to generate the certificate. Finally, the certificate is installed by using the Install Certificate Request service.

The left-over CSR is not required anymore and can be removed from the filesystem with the File Delete Request. After a firmware reset the default security configuration is ready to be used.

3.2.5 Use case 4: Download Key and Certificates

The asymmetric keys and certificates are provided by the application.

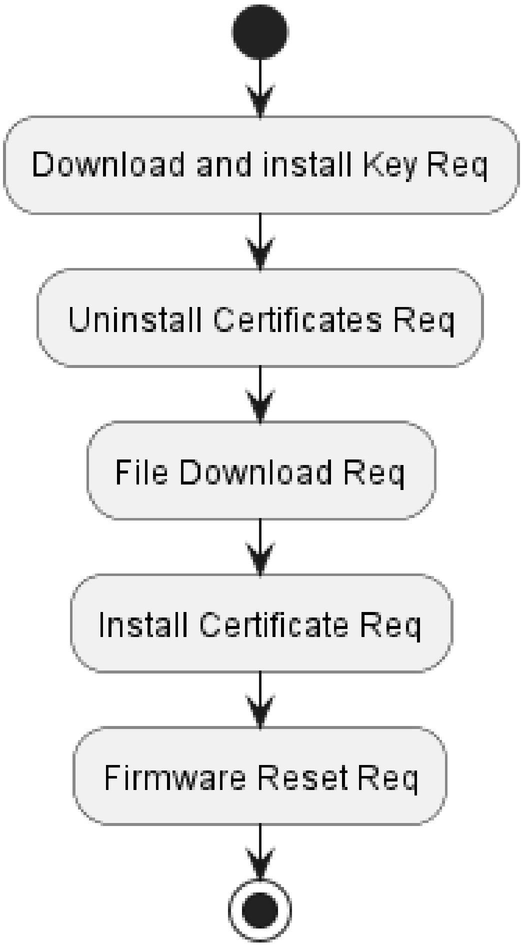

Figure 8. Download Key and Certificates

The application uses the Download and Install Key Request to download and install an asymmetric key for a specific security component on the device. Subsequently, the Uninstall all certificates request service needs to be issued for each security component when installing new certificates. The service also ensures that all old certificates related to the security component are uninstalled, since they become useless with the newly generated key.

The application downloads each certificate with the File Download Request. Each certificate is installed with the Install Certificate Request service. After a firmware reset the default security configuration is ready to be used.

Chapter 4 Application Interface

This chapter defines the application interface of the Authentication Manager. Certificate Database

4.1 Security Resources

4.1.1 Resource Flags

Value	Name Description
0x00000000	AUTH_CRTDB_RSC_FLAGS_DEFAULT_RSC
	Default resource for a security component.
0x00000001	AUTH_CRTDB_RSC_FLAGS_OPERATIONAL_RSC
	Operational(non-default) resource for a security component.

Table 5. AUTH_CRTDB_RSC_FLAGS_

4.1.2 Option Flags

Value	Name Description
0x00000000	AUTH_CRTDB_OPTION_FLAGS_DER_ENCODING
	Resource is DER format encoded
0x00000001	AUTH_CRTDB_OPTION_FLAGS_PEM_ENCODING
	Resource is PEM format encoded

Table 6. AUTH_CRTDB_OPTION_FLAGS_

4.1.3 Resource Types

Security firmware Resource types.

Value	Name Description
0	AUTH_CRTDB_RSC_TYPE_NONE
	Undefined security resource.
1	AUTH_CRTDB_RSC_TYPE_KEY
	Private Key that belongs to the security component.
2	AUTH_CRTDB_RSC_TYPE_EE_CERTIFICATE
	End Entity Certificate. either does not contain a "Basic Constraints extension", or contains such an extension with the CA flag set to False.
3	AUTH_CRTDB_RSC_TYPE_CA_CERTIFICATE
	A Certificate[chain] that belongs to a certificate Authority. This is a part of certificate path that proves an end entity certificate has been approved by a trusted authority.

Table 7. AUTH_CRTDB_SECURITY_RESOURCE_TYPE_E

4.1.4 Resources limits

4.1.4.1 Maximum Filename Length

AUTH_CRTDB_MAX_FILENAME_LEN	128

4.1.4.2 Maximum Subject Distinguished Name Length

AUTH_CRTDB_MAX_SDN_LEN	512

4.1.4.3 Maximum Key Length

AUTH_CRTDB_KEY_DOWNLOAD_BUF_LEN	4096

4.2 Component and Key Types

Security component IDs.

Each security component gets a range of 16 IDs assigned. This allows having multiple security resources based on different keys for the same component (e.g. different security resources for cyclic and acylic communication). Refer to the security component documentation to check which IDs are used and their purposes

Value	Name	Description
0x00000010	AUTH_CRTDB_SEC_COMPONENT_ID_CIP_SECURITY	CIP Security component uses the IDs 0x00000010-0x0000001F.
0x00000020	AUTH_CRTDB_SEC_COMPONENT_ID_HTTPS	HTTPS component uses the IDs 0x00000020-0x0000002F.
0x00000030	AUTH_CRTDB_SEC_COMPONENT_ID_OPCUA	OPCUA component uses the IDs 0x00000030-0x0000003F.
0x00000040	AUTH_CRTDB_SEC_COMPONENT_ID_OMB	OpenModbus Security component uses the IDs 0x00000040-0x0000004F.
0x00000050	AUTH_CRTDB_SEC_COMPONENT_ID_MQTT	MQTT component uses the IDs 0x00000050-0x0000005F.
0x00000060	AUTH_CRTDB_SEC_COMPONENT_ID_NETCONF	NETCONF component uses the IDs 0x00000060-0x0000006F.
0x00000070	AUTH_CRTDB_SEC_COMPONENT_ID_PN_SECURITY	PROFINET Security component uses the IDs 0x00000070-0x0000007F

Table 8. AUTH_CRTDB_SECURITY_COMPONENT_ID_E

Supported public key IDs.

Value	Name	Description
0x00000010	AUTH_CRTDB_PK_ID_RSA_2048	RSA key with 2048 bit length.
0x00000011	AUTH_CRTDB_PK_ID_RSA_3072	RSA key with 3072 bit length.
0x00000012	AUTH_CRTDB_PK_ID_RSA_4096	RSA key with 4096 bit length.
0x00000050	AUTH_CRTDB_PK_ID_EC_SECP256R1	EC key based on elliptic curve secp256r1.
0x00000051	AUTH_CRTDB_PK_ID_EC_SECP384R1	EC key based on elliptic curve secp384r1.

Table 9. AUTH_CRTDB_PK_ID_E

4.3 Generate and Install Key

Generate and Install Key Command Request

AUTHMGR_CRTDB_GENERATE_AND_INSTALL_KEY_REQ	0x0000B080

Generate and Install Key Command Confirmation

AUTHMGR_CRTDB_GENERATE_AND_INSTALL_KEY_CNF	0x0000B081

Generate and Install Key Request Packet Description

Generate and install an asymmetric key pair for a specific security component. This service replaces an existing key of that security component, so that previous security resources belonging to that security component (e.g. certificates, CSRs, ...) will no longer be valid.

NOTE The service does not verify if the key (type, length) complies to the corresponding security component requirements. It is the responsibility of the application to install keys supported by the security component.

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	8
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_GENERATE_AND_INSTALL_KEY_REQ
tData	**AUTHMGR_CRTDB_GENERATE_AND_INSTALL_KEY_REQ_DATA_T**	
`ulComponentId`	uint32_t	Security component identifier of type AUTH_CRTDB_SECURITY_COMPONENT_ID_E
`ulKeyId`	uint32_t	Key type identifier of type AUTH_CRTDB_PK_ID_E

Table 10. AUTHMGR_CRTDB_GENERATE_AND_INSTALL_KEY_REQ_T

Generate and Install Key Confirmation Packet Description

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	0
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_GENERATE_AND_INSTALL_KEY_CNF

Table 11. AUTHMGR_CRTDB_GENERATE_AND_INSTALL_KEY_CNF_T

4.4 Download and Install Key

Download and Install Key Command Request

AUTHMGR_CRTDB_DOWNLOAD_AND_INSTALL_KEY_REQ	0x0000B090

Download and Install Key Command Confirmation

AUTHMGR_CRTDB_DOWNLOAD_AND_INSTALL_KEY_CNF	0x0000B091

Download and Install Key Request Packet Description

Download and install a key for a specific security component. This service replaces an existing key of that security component, so that previous security resources belonging to that security component (e.g. certificates, CSRs, …) will no longer be valid.

> **NOTE** The service does not verify if the key (type, length) complies to the corresponding security component requirements. It is the responsibility of the application to install keys supported by the security component.

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	4 + key length
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_DOWNLOAD_AND_INSTALL_KEY_REQ
tData	**AUTHMGR_CRTDB_DOWNLOAD_AND_INSTALL_KEY_REQ_DATA_T**	
`ulComponentId`	uint32_t	Security component identifier (see AUTH_CRTDB_SECURITY_COMPONENT_ID_E)
`abKey[__HIL_VARIABLE_LENGTH_ARRAY]`	uint8_t	Holds a DER or PEM encoded key (maximum AUTH_CRTDB_KEY_DOWNLOAD_BUF_LEN bytes allowed). The key length is implicitly provided within the packet data length (tHead.ulLen)

Table 12. AUTHMGR_CRTDB_DOWNLOAD_AND_INSTALL_KEY_REQ_T

Download and Install Key Confirmation Packet Description

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	0
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_DOWNLOAD_AND_INSTALL_KEY_CNF

Table 13. AUTHMGR_CRTDB_DOWNLOAD_AND_INSTALL_KEY_CNF_T

4.5 Generate CSR

Generate CSR Command Request

AUTHMGR_CRTDB_GENERATE_CSR_REQ	0x0000B082

Generate CSR Command Confirmation

AUTHMGR_CRTDB_GENERATE_CSR_CNF	0x0000B083

Generate CSR Request Packet Description

Generate a PEM/DER encoded Certificate Signing Request for a specific security component (ulComponentId). The CSR is stored as a file on the file system.

The Subject Distinguished Name (szSubjectName), includes a sequence of attribute type/value pairs separated by a comma. Syntax: type=value[,type=value]

'type' is one of the following short names:

Short form	Description	ASN.1 OID
CN	commonName	2.5.4.3
SN	surName	2.5.4.4
serialNumber	serialNumber	2.5.4.5
C	countryName	2.5.4.6 (2 letter ISO 3166 Country Code)
L	localityName	2.5.4.7
ST	state	2.5.4.8
O	organizationName	2.5.4.10
OU	organizationalUnit	2.5.4.11
title	title	2.5.4.12
postalAddress	postalAddress	2.5.4.16
postalCode	postalCode	2.5.4.17
GN	givenName	2.5.4.42
initials	initials	2.5.4.43
generationQualifier	generationQualifier	2.5.4.44
uniqueIdentifier	uniqueIdentifier	2.5.4.45
dnQualifier	dnQualifier	2.5.4.46
pseudonym	pseudonym	2.5.4.47
emailAddress	emailAddress	1.2.840.113549.1.9.1 (Deprecated, use an altName extension instead)
DC	domainComponent	0.9.2342.19200300.100.1.25

Example: "CN=My Certificate Authority,C=DE,O=My Company"

The extensions buffer holds a sequence of all x509v3 extensions to be included in the CSR. Each extension must be described with the following structure:

Element	Data type	Description
Critical	UINT8	Indicates if the "critical" flag of the extension has to be set. (0 = FALSE / others = TRUE)
OID Tag Length	UINT32	The length of the OID tag
OID Tag	Array of UINT8	The OID Tag of the extension
OID Data Length	UINT32	The length of the OID data
OID Data	Array of UINT8	The OID data of the extension

NOTE : The value TRUE for the critical flag can only be used for the standard extensions defined in RFC5280

> **NOTE** : The OID Data may contain one single value (e.g. an octet string) or multiple values (e.g. a constructed sequence). The value data type is coded within the OID Data. The values follow the ASN.1 syntax "ASN.1 type | length | value".

> **NOTE** : Some helper functions are provided in the file "AuthMgr_CrtDB_X509v3Ext_helpers.h" to simplify the generation of the extensions buffer

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	648 + extensions length
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_GENERATE_CSR_REQ
tData	**AUTHMGR_CRTDB_GENERATE_CSR_REQ_DATA_T**	
`ulComponentId`	uint32_t	Security component identifier (see AUTH_CRTDB_SECURITY_COMPONENT_ID_E)
`ulOptionFlags`	uint32_t	Option flags to customize the process (see AUTH_CRTDB_OPTION_FLAGS_*)
`szFileName[AUTH_CRTDB_MAX_FILENAME_LEN]`	char	Null-terminated CSR File Name (including file path) (e.g. "file://SYSVOLUME/PORT_0/CSR_0020.pem") (maximum 128 characters, including '\0')
`szSubjectName[AUTH_CRTDB_MAX_SDN_LEN]`	char	Null-terminated Subject Distinguished Name (maximum 512 characters, including '\0')
`abExtentions[__HIL_VARIABLE_LENGTH_ARRAY]`	uint8_t	Buffer holding the X509v3 extensions to be set in the CSR. The buffer length is implicitly provided within the packet data length (tHead.ulLen)

Table 14. AUTHMGR_CRTDB_GENERATE_CSR_REQ_T

Generate CSR Confirmation Packet Description

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	0
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_GENERATE_CSR_CNF

Table 15. AUTHMGR_CRTDB_GENERATE_CSR_CNF_T

Examples of x509v3 extensions coding

Example 1: standard extension 2.5.29.14 "subjectKeyIdentifier", single value, non-critical:

Element	Value	Description
Critical	0x00	Critical = FALSE
OID Tag Length	0x03, 0x00, 0x00, 0x00	OID Tag length = 3 bytes
OID Tag	0x55, 0x1D, 0x0E,	OID Tag = 2.5.29.14
OID Data Length	0x16, 0x00, 0x00, 0x00,	OID value length = 22 bytes
OID Data - ASN1 Type	0x04,	ASN1_OCTET_STRING
OID Data - ASN1 Length	0x14,	Length = 20 bytes
OID Data - ASN1 Data	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13, 0x14,	Data

Table 16. Extention example 1

Example 2: standard extension 2.5.29.15 "keyUsage", single value, critical:

Element	Value	Description
Critical	0x00	Critical = TRUE
OID Tag Length	0x03, 0x00, 0x00, 0x00,	OID Tag length = 3 bytes
OID Tag	0x55, 0x1D, 0x0F,	OID Tag = 2.5.29.15
OID Data Length	0x04, 0x00, 0x00, 0x00,	OID value length = 4 bytes
OID Data - ASN1 Type	0x03,	ASN1_BIT_STRING
OID Data - ASN1 Length	0x02,	Length = 2 bytes
OID Data - ASN1 Data	0x02, 0xA4,	Data (key used for digitalSignature, keyEncipherment and keyCertSign)

Table 17. Extention example 2

Example 3: non-standard extension 1.3.6.1.4.1.50316.802.1 "OMB RoleOID", single value, non-critical:

Element	Value	Description
Critical	0x00	Critical = FALSE
OID Tag Length	0x0B, 0x00, 0x00, 0x00,	OID Tag length = 11 bytes
OID Tag	0x2B, 0x06, 0x01, 0x04, 0x01, 0x83, 0x89, 0x0C, 0x86, 0x22, 0x01,	OID Tag = 1.3.6.1.4.1.50316.802.1
OID Data Length	0x0A, 0x00, 0x00, 0x00,	OID value length = 10 bytes
OID Data - ASN1 Type	0x0C,	ASN1_UTF8_STRING
OID Data - ASN1 Length	0x08,	Length = 8 bytes
OID Data - ASN1 Data	0x4F, 0x70, 0x65, 0x72, 0x61, 0x74, 0x6F, 0x72,	Data (RoleOID = "Operator")

Table 18. Extention example 3

Example 4: standard extension 2.5.29.17 "subjectAltName", multiple values, non-critical:

Element	Value	Description
Critical	0x00	Critical = FALSE
OID Tag Length	0x03, 0x00, 0x00, 0x00,	OID Tag length = 3 bytes
OID Tag	0x55, 0x1D, 0x11,	OID Tag = 2.5.29.17
OID Data Length	0x20, 0x00, 0x00, 0x00,	OID value length = 32 bytes
OID Data - ASN1 Type	0x30,	ASN1_CONSTRUCTED | ASN1_SEQUENCE (multiple values constructed as a sequence)
OID Data - ASN1 Length	0x1E,	Length = 30 bytes
OID Data - ASN1 Data	0x30,	Data type = ASN1_CONSTRUCTED | ASN1_SEQUENCE → multiple values constructed as a sequence
	0x1E,	Total data length = 30 bytes
	Value 1	
	0x82,	Data type = ASN1_CONTEXT_SPECIFIC | dnsName
	0x0E,	Data length = 14 bytes
	0x2A, 0x2E, 0x68, 0x69, 0x6C, 0x73, 0x63, 0x68, 0x65, 0x72, 0x2E, 0x63, 0x6F, 0x6D,	Data = "*.hilscher.com"
	Value 2	
	0x82,	Data type = ASN1_CONTEXT_SPECIFIC | dnsName
	0x0C,	Data length = 12 bytes
	0x68, 0x69, 0x6C, 0x73, 0x63, 0x68, 0x65, 0x72, 0x2E, 0x63, 0x6F, 0x6D,	Data = "hilscher.com"

Table 19. Extention example 4

4.6 Sign Request

Sign Command Request

AUTHMGR_CRTDB_SIGN_REQ	0x0000B084

Sign Command Confirmation

AUTHMGR_CRTDB_SIGN_CNF	0x0000B085

Sign Request Packet Description

Sign a Certificate Signing Request and store the resulted certificated on the file system.

> **NOTE** The CSR must be present in the file system before using this service.

> **NOTE** to indicate that a certificate has no well-defined expiration date, the szNotValidAfter SHOULD be assigned the GeneralizedTime value of 99991231235959 (RFC5280).

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	294
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_SIGN_REQ
tData	**AUTHMGR_CRTDB_SIGN_REQ_DATA_T**	
`ulComponentId`	uint32_t	Security component identifier which is going to sign the certificate (see AUTH_CRTDB_SECURITY_COMPONENT_ID_E)
`ulOptionFlags`	uint32_t	Option flags to customize the process (see AUTH_CRTDB_OPTION_FLAGS_*)
`szNotValidBefore[AUTH_CRTDB_UTCTIME_LEN]`	char	Null-terminated validity date notBefore UTC time in "YYYYMMDDhhmmss" format
`szNotValidAfter[AUTH_CRTDB_UTCTIME_LEN]`	char	Null-terminated validity date notAfter UTC time in "YYYYMMDDhhmmss" format
`szCsrFileName[AUTH_CRTDB_MAX_FILENAME_LEN]`	char	Null-terminated CSR File Name (including file path) (maximum 128 characters, including '\0')
`szCrtFileName[AUTH_CRTDB_MAX_FILENAME_LEN]`	char	Null-terminated Certificate File Name (including file path) (maximum 128 characters, including '\0')

Table 20. AUTHMGR_CRTDB_SIGN_REQ_T

Sign Confirmation Packet Description

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	0
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_SIGN_CNF

Table 21. AUTHMGR_CRTDB_SIGN_CNF_T

4.7 Install Certificate

Install Certificate Command Request

AUTHMGR_CRTDB_INSTALL_CERT_REQ	0x0000B086

Install Certificate Command Confirmation

AUTHMGR_CRTDB_INSTALL_CERT_CNF	0x0000B087

Install Certificate Request Packet Description

Install a certificate (szFileName) for a specific security component (ulComponentId).

NOTE | The certificate needs to be available in the file system before using this service.

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	136
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_INSTALL_CERT_REQ
tData	**AUTHMGR_CRTDB_INSTALL_CERT_REQ_DATA_T**	
`ulComponentId`	uint32_t	Security component identifier (see AUTH_CRTDB_SECURITY_COMPONENT_ID_E)
`usCertificateType`	uint16_t	Certificate type identifier (see AUTH_CRTDB_SECURITY_RESOURCE_TYPE_E)
`usFlags`	uint16_t	bit field certificate attributes (see AUTH_CRTDB_RSC_FLAGS_*)
`szFileName[AUTH_CRTDB_MAX_FILENAME_LEN]`	char	Null-terminated certificate File Name (including file path) (e.g. "file://SYSVOLUME/PORT_0/dflt_crt.cer") (maximum 128 characters, including '\0')

Table 22. AUTHMGR_CRTDB_INSTALL_CERT_REQ_T

Install Certificate Confirmation Packet Description

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	0
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_INSTALL_CERT_CNF

Table 23. AUTHMGR_CRTDB_INSTALL_CERT_CNF_T

4.8 Uninstall Certificates

Uninstall Certificates Command Request

AUTHMGR_CRTDB_UNINSTALL_ALL_CERTS_REQ	0x0000B088

Uninstall Certificates Command Confirmation

AUTHMGR_CRTDB_UNINSTALL_ALL_CERTS_CNF	0x0000B089

Uninstall All Certificates Request Packet Description

Un-install all certificates related to a specific security component.

NOTE This service does not remove the corresponding certificates from the file system. Manual effort is required to remove the certificates from the file system.

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	6
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_UNINSTALL_ALL_CERTS_REQ
tData	**AUTHMGR_CRTDB_UNINSTALL_ALL_CERTS_REQ_DATA_T**	
`ulComponentId`	uint32_t	Security component identifier (see AUTH_CRTDB_SECURITY_COMPONENT_ID_E)
`usFlags`	uint16_t	Bit field certificate attributes (see AUTH_CRTDB_RSC_FLAGS_*)

Table 24. AUTHMGR_CRTDB_UNINSTALL_ALL_CERTS_REQ_T

Uninstall All Certificates Confirmation Packet Description

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	0
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_UNINSTALL_ALL_CERTS_CNF

Table 25. AUTHMGR_CRTDB_UNINSTALL_ALL_CERTS_CNF_T

4.9 Factory Reset

Factory Reset Command Request

AUTHMGR_CRTDB_FACTORY_RESET_REQ	0x0000B08A

Factory Reset Command Confirmation

AUTHMGR_CRTDB_FACTORY_RESET_CNF	0x0000B08B

Factory Reset Request Packet Description

Remove all certificate and key files from the system and all the entries from the internal database.

> **NOTE** : After the reset is done, the certDB does not contain any security configuration anymore. This may lead to errors or unexpected behaviors in the security components

> **NOTE** : it is recommended to perform a system reset afterwards to run the "out-of-box" behavior and download the (default) security configurations again.

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	0
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_FACTORY_RESET_REQ

Table 26. AUTHMGR_CRTDB_FACTORY_RESET_REQ_T

Factory Reset Confirmation Packet Description

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	0
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_FACTORY_RESET_CNF

Table 27. AUTHMGR_CRTDB_FACTORY_RESET_CNF_T

4.10 Get Component Stat

Get Component Stat. Command Request

AUTHMGR_CRTDB_GET_COMPONENT_STAT_REQ	0x0000B092

Get Component Stat. Command Confirmation

AUTHMGR_CRTDB_GET_COMPONENT_STAT_CNF	0x0000B093

Get Component Stat Request Packet Description

Retrieve basic information(Existence/Cardinality) about the security configuration artifacts for a specific security component.

NOTE Artifacts are considered "existing", when both of the following criteria are met:

1. There is a corresponding file for it on the file system.
2. The artifact has been installed in the AuthMgr Certificate Database.

NOTE No validation/verification of resources are provided within this service.

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	4
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_GET_COMPONENT_STAT_REQ
tData	**AUTHMGR_CRTDB_GET_COMPONENT_STAT_REQ_DATA_T**	
`ulComponentId`	uint32_t	Security component identifier (see AUTH_CRTDB_SECURITY_COMPONENT_ID_E)

Table 28. AUTHMGR_CRTDB_GET_COMPONENT_STAT_REQ_T

Get Component Stat Confirmation Packet Description

Variable	Type	Description
tHead	**HIL_PACKET_HEADER_T**	
`ulDest`	uint32_t	
`ulLen`	uint32_t	6
`ulSta`	uint32_t	0
`ulCmd`	uint32_t	AUTHMGR_CRTDB_GET_COMPONENT_STAT_CNF
tData	**AUTHMGR_CRTDB_GET_COMPONENT_STAT_CNF_DATA_T**	
`tComopnentStat`	AUTH_CRTDB_COMPONENT_STAT_T	

Table 29. AUTHMGR_CRTDB_GET_COMPONENT_STAT_CNF_T