ONVIF™
Advanced Security Service Specification

Version 1.3
February 2016

CONTENTS

1 Scope

This document defines the web service interface for ONVIF Advanced Security Features such as a keystore and a TLS server on an ONVIF device.

Web service usage is outside of the scope of this document. Please refer to the ONVIF core specification.

2 Normative References

IEEE 802.1X, Port-Based Network Access Control

<http://standards.ieee.org/getieee802/download/802.1X-2004.pdf>

ONVIF Core Specification

<http://www.onvif.org/specs/core/ONVIF-Core-Specification-v260.pdf>

ONVIF Device Management Service

<http://www.onvif.org/specs/core/ONVIF-Core-Specification-v250.pdf>

RFC 2246 The TLS Protocol Version 1.0

<http://www.ietf.org/rfc/rfc2246.txt>

RFC 2898 PKCS#5 Password-based Cryptography Specification v2.0

<http://www.ietf.org/rfc/rfc2898.txt>

RFC 2986 PKCS #10: Certification RequestSyntaxSpecification Version 1.7

<http://www.ietf.org/rfc/rfc2986.txt>

RFC 3279 Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

<http://www.ietf.org/rfc/rfc3279.txt>

RFC 3447 Public Key Cryptography Standards #1: RSA Cryptogaphy Specifications Version 2.1

<http://www.ietf.org/rfc/rfc3447.txt>

RFC 4055 Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

<http://www.ietf.org/rfc/rfc4055.txt>

RFC 4346 The Transport Layer Security (TLS) Protocol Version 1.1

<http://www.ietf.org/rfc/rfc4346.txt>

RFC 5208 Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax Specification v1.2

<http://www.ietf.org/rfc/rfc5208.txt>

RFC 5280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

<http://www.ietf.org/rfc/rfc5280.txt>

RFC 5958 Asymmetric Key Packages

<http://www.ietf.org/rfc/rfc5958.txt>

RFC 5959 Algorithms for Asymmetric Key Package Content Type

<http://www.ietf.org/rfc/rfc5959.txt>

Unified Modeling Language (UML)

<http://www.omg.org/spec/UML>

PKCS#5 Password-based Encryption Standard v1.5, RSA Laboratories, 1993

PKCS#12: Personal Information Exchange Syntax v1.0, RSA Laboratories, 1999

3 Terms and Definitions

3.1 Definitions

Alias	An alias is a name for an object on the device that is chosen by the client and treated transparently by the device.
Certificate	A certificate as used in this specification binds a public key to a subject entity. The certificate is digitally signed by the certificate issuer (the certification authority) to allow for verifying its authenticity.
Certification Authority	A certification authority is an entity that issues certificates to subject entities.
Certification Path	A certification path is a sequence of certificates in which the signature of each certificate except for the last certificate can be verified with the subject public key in the next certificate in the sequence.
Certificate Revocation List	A certificate revocation list is a digitally signed list of IDs of certificates that have been revoked by the issuing CA.
Digital Signature	A digital signature for an object allows to verify the object's authenticity, i.e., to check whether the object has in fact been created by the signer and has not been modified afterwards. A digital signature is based on a key pair, where the private key is used to create the signature and the public key is used for verification of the signature.
Inner	The authentication protocol used after having established a secure connection using the outer authentication protocol. Also called Stage 2.
Key	A key is an input to a cryptographic algorithm. Sufficient randomness of the key is usually a necessary condition for the security of the algorithm. This specification supports RSA key pairs as keys.
Key Pair	A key that consists of a public key and (optionally) a private key.
Outer	The initial authentication protocol used to establish a secure connection between the device and the IEEE 802.1X authentication server. Also called Stage 1.
Passphrase	A secret string that is shared between two or more parties. A passphrase may be used to derive keys.
RSA key pair	A key pair that is accepted as input by the RSA algorithm.
TLS Server	TLS-enabled HTTP Server (HTTPS)

3.2 Abbreviations

CA	Certification Authority
CRL	Certificate Revocation List
CSR	Certificate Signing Request (also called Certification Request)
EAP	Extensible Authentication Protocol

IEEE	Institute of Electrical and Electronics Engineers
MAC	Message Authentication Code
MD5	Message Digest Algorithm 5
MSCHAPv2	Microsoft's Challenge Handshake Authentication Protocol version 2
ONVIF	Open Network Video Interface Forum
PEAP	Protected EAP
SHA	Secure Hashing Algorithm
SSL	Secure Socket Layer
TLS	Transport Layer Security
TTLS	Tunneled TLS

3.3 Namespace

This document references the following namespaces:

Prefix	Namespace URI
env	http://www.w3.org/2003/05/soap-envelope
tas	http://www.onvif.org/ver10/advancedsecurity/wsdl
ter	http://www.onvif.org/ver10/error
tt	http://www.onvif.org/ver10/schema
xs	http://www.w3.org/2001/XMLSchema

4 Overview

4.1 General Structure

This specification covers the following advanced security features:

- Keys and certificates management interface (keystore)
- TLS server configuration interface
- IEEE 802.1X

Basic security features such as user authentication based on WS UsernameToken and HTTP Authentication as well as a default access policy are specified in the [ONVIF Core Specification] as part of the device management service.

WSDL for the Advanced Security service is specified in <http://www.onvif.org/ver10/advancedsecurity/wsdl/advancedsecurity.wsdl>.

All sections in this specification are normative unless explicitly marked as informative.

4.2 Certificate-based Client Authentication

4.2.1 Overview

A client may be authenticated based on a certification path that it presents to the server, e.g., a TLS client is authenticated by a TLS server.

The certificate-based authentication is performed according to a certification path validation algorithm that enforces a certification path validation policy. A certification path validation policy contains at least one trust anchor (in the form of a certificate) that the device shall assume to be correct. Furthermore, a certification path validation policy contains sources of revocation information to be considered when determining whether a certificate in question has been revoked. This specification uses CRLs as sources of revocation information, while future versions may include additional sources.

A certificate revocation list (CRL) contains certificates that have been revoked by the issuing CA. Therefore, the private key corresponding to the public key in a revoked certificate shall be considered compromised.

4.2.2 Certification path validation

This section defines an algorithm to validate a certification path that is, e.g., supplied by an ONVIF client to an ONVIF device.

Algorithm input:

1. Certification path $c_1,...,c_m$
2. Current date and time
3. Certification path validation policy

Algorithm output:

- *valid* if the certification path is considered valid, *invalid* otherwise.

Algorithm steps:

1. Construct all prospective certification paths $c_1,...,c_n$ from the input certification path $c_1,...,c_m$ as specified in Sect. 4.2.3. If no prospective certification path can be constructed from the input certification path, return *invalid*.
2. For all prospective certification paths $c_1,...,c_n$
    a. Determine whether $c_1,...,c_n$ is valid by applying the algorithm defined in Sect. 4.2.4 with inputs
        - Prospective certification path $c_1,...,c_n$
        - Current date and time
        - Certification path validation policy
    b. If $c_1,...,c_n$ is valid, output *valid.*
3. Output *invalid*.

A device that supports certification path validation as defined in this specification shall implement this algorithm.

4.2.3 Construct Prospective Certification Paths

This section defines an algorithm to construct prospective certification paths from an input certification path as indicated by the certification path validation policy.

Algorithm input:

1. Certification path $c_1,...,c_m$
2. Certification path validation policy

Algorithm output:

1. Prospective certification paths $c_1,...,c_n$ if at least one prospective certification path exists.

2. NULL if no prospective certification path exists.

Algorithm steps:

1. For all i in {1,...,m} such that c_i is considered a trust anchor according to the certification path validation policy, add $c_1,...,c_i$ to the set of prospective certification paths.

2. Determine all certification path extensions $c_{m+1},...,c_n$ such that

   - for all c_i with i in {m+1,...n-1}, the issuer of c_i is the subject of c_{i+1}

   - c_n is considered a trust anchor according to the certification path validation policy

3. Add all extended certification paths $c_1,...,c_m,c_{m+1},...,c_n$ to the set of prospective certification paths.

4. Return the set of prospective certification paths.

4.2.4 Validate Prospective Certification Path

This section defines an algorithm to validate a prospective certification path.

Algorithm input:

1. Prospective certification path $c_1,...,c_n$

2. Current date and time *t*

3. Certification path validation policy

Algorithm output:

- *True* if certification path is considered valid under the certification path validation policy, *false* otherwise.

Algorithm steps:

1. Execute the algorithm specified in [RFC 5280], Sect. 6.1, with inputs

   a. Prospective certification path $c_n,...,c_1$

   b. Current date and time *t*

   c. *User-initial-policy-set* as defined in the certification path validation policy

   d. *Initial-policy-mapping-inhibit* as defined in the certification path validation policy

   e. *Initial-explicit-policy* as defined in the certification path validation policy

   f. *Initial-any-policy-inhibit* as defined in the certification path validation policy

   g. *Initial-permitted-subtrees* as defined in the certification path validation policy

   h. *Initial-excluded-subtrees* as defined in the certification path validation policy

2. If the output of step (1) contains a success indication, return *true*. Otherwise, return *false*.

For determining the revocation status of a given certificate *cert* (Step (3) in [RFC 5280, Sect. 6.1.3]) in Step (1), the device shall use the algorithm defined in Sect. 4.2.5 with inputs

a. Certificate := *cert*

b. Certification path validation policy

In order to determine whether an X.509 version 1 or version 2 certificate is a CA certificate as required in [RFC 5280], Sect. 6.1.4, step (k), the device shall use the source specified in the cA-information-source-for-v1-and-v2 information of the certification path validation policy.

4.2.5 Determine Certificate Revocation Status

Algorithm input:

1. Certificate *cert*
2. Certification path validation policy

Algorithm output:

- REVOKED if certificate is considered revoked.
- UNREVOKED if certificate is considered to have been released from hold.
- UNDERTERMINED if the certificate is considered neither revoked nor released from hold.

Algorithm steps:

1. For all CRLs *I* that shall be considered according to the certification path validation policy, execute the CRL validation algorithm defined in [RFC 5280], Sect. 6.3 with inputs

   a. Certificate := the certificate *cert*

   b. *use-deltas* as defined in the certification path validation policy

2. For all other sources of revocation information to be considered according to the certification path validation policy, determine the revocation status of the certificate *cert* based on the certification path validation policy.

3. Combine the outputs of steps (1) and (2) as specified by the certification path validation policy and output the result.

4.2.6 Certification Path Validation Policy

4.2.6.1 Certification Path Validation Algorithm Parameters

By default, a device shall use the values defined in Table 1 for the algorithm parameters defined in [RFC 5280, Sect. 6.1.1] for the certification path validation algorithm defined in Sect. 4.2.4.

Table 1: Default parameter values for the certification path validation algorithm

Parameter	Default Value	Default Value Semantics
User-initial-policy-set	Any-policy	The device is not concerned about certificate policy.
Initial-policy-mapping-inhibit	0	Policy mapping is not inhibited.
Initial-explicit-policy	0	The prospective certification path does not have to be valid for at least one certificate policy in the user-initial-

		policy-set
Initial-any-policy-inhibit	0	The any-policy identifier, if asserted in a certificate, does not have to be ignored
Initial-permitted-subtrees	(not specified)	No restrictions on the subtree within which all subject names in every certificate in the prospective certification path must fall.
Initial-excluded-subtrees	(not specified)	No restrictions on the subtree within which no subject names in any certificate in the prospective certification path may fall.
cA-information-source-for-v1-and-v2	None	The device shall consider X.509 version 1 and version 2 certificates as non-CA certificates.

4.2.6.2 Revocation Status Checking

By default, a device shall use the parameter values defined in Table 2 for the parameters defined in [RFC 5280, Sect. 6.1.1] for the CRL-based certificate revocation status checking algorithm defined in Sect. 4.2.5.

Table 2: Default parameter values for the revocation status checking algorithm

Parameter	Default Value	Default Value Semantics
Use-deltas	False	Delta CRLs, if available, are applied to CRLs.
Relevant-reason-codes	All-reasons	The device considers a certificate revoked if it has been revoked for any reason defined in [RFC 5280].

By default, a device shall consider all trusted sources of revocation information that it has access to when determining the revocation status of a certificate. The device shall consider the certificate in question revoked if and only if at least one such source indicates that the certificate in question is revoked. If one such source is unavailable, the device shall behave as if this source had provided the reply UNDETERMINED.

A device shall consider at least the CRLs that are present in the keystore of the device.

By default, certificates that are considered revoked shall not be included in prospective certification paths.

4.2.6.3 Trust Anchors

The trust anchors assigned to the certification path validation policy shall be used as trust anchor input to the certification path validation algorithm specified in Sect. 4.2.4.

4.2.6.4 Certificate Repository for constructing Certification Paths

By default, the certification path validation algorithm specified in Sect. 4.2.2 shall consider all certificates in the keystore on the device when constructing prospective certification paths.

4.2.6.5 Specific certification path validation parameters

Table 3 defines additional certification path validation parameters.

Table 3: Specific certification path validation parameters

Parameter	Default Value	Default Value Semantics
RequireTLSWWWClientAuthExtendedKeyUsage	False	If true, a TLS server shall only allow TLS clients to connect that present a client certificate containing the Require WWW client auth extended key usage extension as specified in RFC 5280, Sect. 4.2.1.12.

4.2.7 Validate CRLs

By default, the device shall use the following algorithm to obtain and validate the certification path for a CRL issuer in Step (f) of the CRL processing algorithm defined in [RFC 5280, Sect. 6.3.3].

Algorithm input:

1. CRL *I*
2. Certification path validation policy

Algorithm output: *valid* if the CRL is considered valid, *invalid* otherwise

Algorithm steps:

1. Let $c_1,\ldots,c_m$ denote the certificates in the certificate repository for constructing certification paths as defined by the certification path validation policy that contain the issuer of *I* as subject.
2. For each certificate c_i
   a. Execute the certification path validation algorithm defined in Sect. 4.2.2 with input
      - The certification path with c_i as the only certificate in the path
      - Current time and date
      - Certification path validation policy
   b. If c_i is valid, verify the signature of the CRL with the subject public key in c_i.
   c. If the signature verification was successful, return *valid*.
3. Return *invalid.*

4.3 IEEE 802.1X

IEEE 802.1X is an IEEE standard for port based network access control for the purpose of providing authentication and authorization of the devices attached to LAN ports. It allows access to the LAN port to devices that are configured for access, and prevents access to the LAN port to devices that are not correctly configured.

This specification recommends the adoption of IEEE 802.1X for port based authentication for wireless networks.

This specification defines a set of commands to configure and manage a device's IEEE 802.1X configurations, both for wireless and hardwired network interfaces. It assumes that

IEEE 802.1X configuration and reconfiguration is performed outside of the IEEE 802.1X-secured network.

Many schema elements in this specification include Dot1X as shorthand for IEEE 802.1X. This convention increases the readability of source code generated from the WSDL.

5 Advanced Security Service

5.1 General Structure

This section covers the security features

- Keystore
- TLS server
- IEEE 802.1X

The design and data model of the ONVIF Advanced Security Service is reflected in Figure 1.

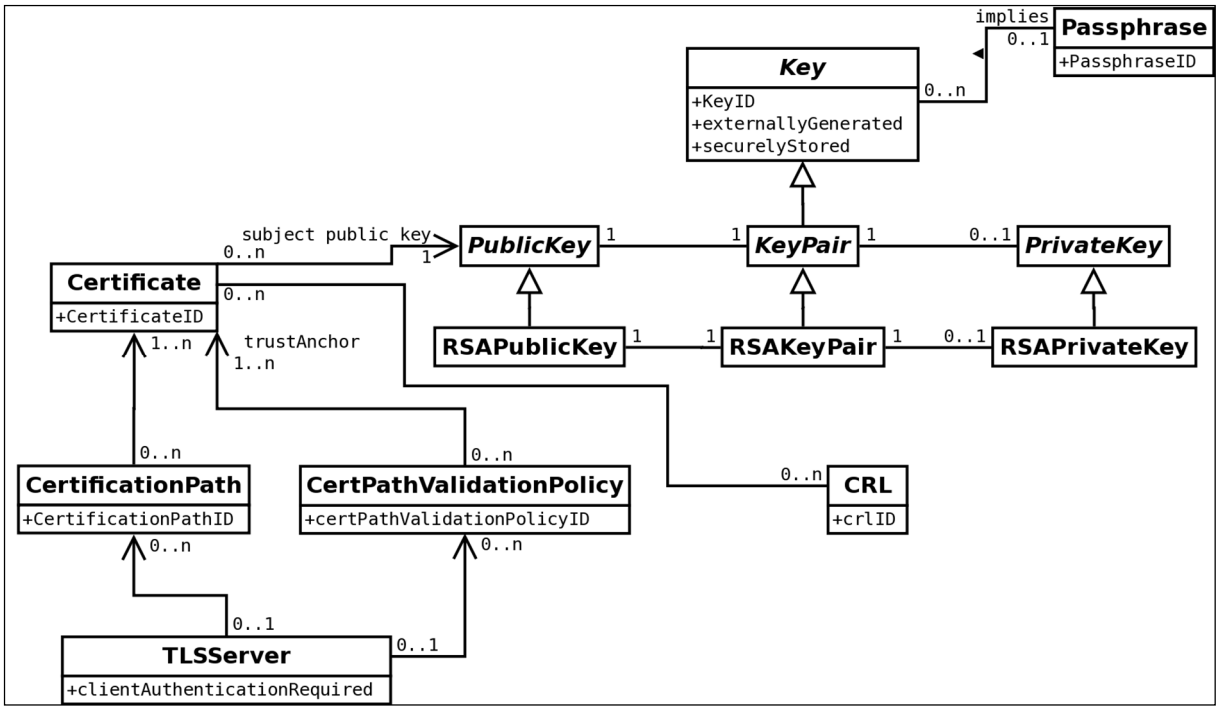

Figure 1 ONVIF Advanced Security Service [UML] Class Diagram

5.2 Keystore

5.2.1 Elements of the Keystore

The keystore security feature handles the storage and management of passphrases, keys, and certificates on an ONVIF device.

The keystore specified in this document supports passphrases, keys, key pairs, which are a particular type of key, RSA key pairs, which are a particular type of key pairs, certificates, certification paths, certificate revocation lists, and certification path validation policies.

The boolean attribute *externallyGenerated* of a key shall be true if and only if the key was generated outside the device.

The boolean attribute *securelyStored* of a key shall be true if and only if the key is stored in a specially protected hardware component (e.g., a trusted platform module) inside the device.

5.2.2 Unique Identifiers

An ID is used to uniquely identify objects of a particular type in the keystore on a device, i.e., no two objects of the same type shall have the same ID at any time.

Passphrases in the keystore shall be uniquely identified by passphrase IDs, keys shall be uniquely identified by key IDs, certificates shall be uniquely identified by certificate IDs, certification paths in the keystore shall be uniquely identified by certification path IDs, certificate revocation lists shall be uniquely identified by certificate revocation list IDs, certification path validation policies shall be uniquely identified by certification path validation policy IDs, and IEEE 802.1X configurations shall be uniquely identified by IEEE 802.1X configuration IDs.

It shall be noted that while IDs within a specific type shall be unique, no requirement exists for the uniqueness of IDs across different types. For example, there may be a key and a certificate in the keystore that share the same ID.

Devices may assign the ID of a deleted identified object to another, subsequently generated object. However, devices should avoid re-using IDs as long as possible to avoid race conditions on the client side.

A client may supply an alias for passphrases, keys, certificates, certification paths, certificate revocation lists, certification path validation policies and IEEE 802.1X configurations upon creation, e.g., to facilitate recognizing the created object at a later time. The device shall treat such aliases as unstructured data.

5.2.3 Uniqueness of Objects in the Keystore

A device shall allow multiple copies of the same passphrase to be present in the keystore under different IDs simultaneously.

A device shall allow multiple copies of the same certificate and multiple copies of the same certification path to be present in the keystore under different IDs, respectively.

A device shall allow multiple copies of the same certificate revocation list to be present in the keystore under different IDs, respectively.

A device shall allow multiple copies of the same certification path validation policy to be present in the keystore under different IDs, respectively.

A device shall allow multiple copies of the same IEEE 802.1X configuration to be present in the keystore under different IDs simultaneously.

A device shall not allow multiple copies of the same key to be present in the keystore simultaneously.

5.2.4 Referential Integrity

The keystore design relies on associations between

- Keys, especially key pairs, and certificates
- Public keys and private keys in key pairs
- Certificates and certification paths
- Keys and security features
- Certification paths and IEEE 802.1X configurations
- Passphrases and IEEE 802.1X configurations
- IEEE 802.1X configurations and security features
- Certificates and security features

- Certification path validation policies and certificates
- Certificate revocation lists and certificates
- Certification path validation policies and security features

A device shall enforce the following referential integrity rules for delete operations:

- A key shall not be deleted if it is referenced by a certificate or a security feature.
- A certificate shall not be deleted if it is referenced by a certification path, a certificate revocation list, a certification path validation policy, or a security feature.
- A certification path shall not be deleted if it is referenced by an IEEE 802.1X configuration or a security feature.
- A passphrase shall not be deleted if it is referenced by an IEEE 802.1X configuration.
- A certification path validation policy shall not be deleted if it is referenced by a security feature.
- An IEEE 802.1X configuration shall not be deleted if it is referenced by a security feature.

This integrity rule may be enforced by the following mechanism. Reference counters are maintained for keys, certificates, certification paths, passphrases, and IEEE 802.1X configurations. Each time a reference to an object of these types is added, e.g., by associating a certificate to a key pair or assigning a key pair or certificate to a security feature, the reference counter of the object is incremented. Conversely, if a reference to an object is deleted, the reference counter of the referenced object is decremented. Deleting a key, certificate, or certification path is only permitted if the corresponding reference counter is equal to zero.

A device shall enforce the following referential integrity rules for update operations:

- A key shall not be updated if it is referenced by a certificate or a security feature. However, a private key may be added to an existing key pair if the private key matches the public key in the key pair. If a private key is about to be added to a key pair that already contains the private key to be added, the adding operation shall have no effect.
- A certificate shall not be updated if it is referenced by a certification path, a certificate revocation list, a certification path validation policy, or a security feature.
- A certification path validation policy shall not be updated if it is referenced by a security feature.

This specification omits APIs for modifying keys or certificates. If a key or certificate is to be updated, it has to be deleted and newly generated with the updated information. If other API exists that allows for modification of keys or certificates, special care shall be taken in order not to break the referential integrity rule.

A device shall enforce the following invariants:

- The private key and the public key in an asymmetric key pair in the keystore shall always match, i.e., the asymmetric operation under the public key is the inverse of the corresponding operation under the private key.
- The public key in a certificate in the keystore and the public key in an associated key pair in the keystore shall always be equal for all associated key pairs.

5.2.5 Key Status

A key in the keystore is always in exactly one of the following states:

- *ok* (The key is ready to be used)
- *generating* (The key is being generated and not yet ready for use)
- *corrupt* (The key is corrupt and shall not be used, e.g., because it was not properly generated or a hardware fault corrupted a key that was ready to be used)

5.2.6 Keystore Operations

5.2.6.1 Passphrase Management

5.2.6.1.1 Upload Passphrase

This operation uploads a passphrase to the keystore of the device.

Passphrases are uniquely identified using passphrase IDs. The device shall generate a new passphrase ID for the uploaded passphrase.

If the command was successful, the device shall return the ID of the uploaded passphrase.

If the device does not have enough storage capacity for storing the passphrase to be uploaded, the device shall produce a maximum number of passphrases reached fault and shall not upload the supplied passphrase.

If the device cannot process the passphrase to be uploaded, the device shall produce a BadPassphrase fault and shall not upload a passphrase.

Table 4: UploadPassphrase command

UploadPassphrase	Access Class: WRITE_SYSTEM
Message name	**Description**
UploadPassphraseRequest	*This message contains a request for the device to upload a passphrase to the keystore.* xs:string **Passphrase** [1][1] xs:string **PassphraseAlias** [0][1]
UploadPassphraseResponse	*This message contains the ID of the successfully uploaded passphrase.* tas:PassphraseID **PassphraseID** [1][1]
Fault codes	**Description**
env:Receiver ter:Action ter:MaximumNumberOfPassphrasesReached	*The device does not have enough storage space to store the passphrase to be uploaded.*
env:Sender ter:InvalidArgVal ter:BadPassphrase	*The supplied passphrase cannot be processed by the device.*

5.2.6.1.2 Get All Passphrases

This operation returns information about all passphrases that are stored in the keystore of the device.

This operation may be used, e.g., if a client lost track of which passphrases are present on the device.

If no passphrase is stored on the device, the device shall return an empty list.

Table 5: GetAllPassphrases command

GetAllPassphrases	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetAllPassphrasesRequest	*This message contains a request for the device to return information about all passphrases in the keystore.* *This is an empty message.*
GetAllPassphrasesResponse	*This message contains information about all passphrases in the keystore.* tas:PassphraseAttribute **PassphraseAttribute** [0][unbounded]
Fault codes	**Description**
	No command-specific fault codes.

5.2.6.1.3 Delete Passphrase

This operation deletes a passphrase from the keystore of the device.

Passphrases are uniquely identified using passphrase IDs. If no passphrase is stored under the requested passphrase ID in the keystore, a device shall produce an invalid passphrase ID fault. If there is a passphrase under the requested passphrase ID stored in the keystore and the passphrase could not be deleted, a device shall produce a passphrase deletion failed fault.

After a passphrase is successfully deleted, the device may assign its former ID to other passphrases.

Table 6: DeletePassphrase command

DeletePassphrase	Access Class: UNRECOVERABLE
Message name	**Description**
DeletePassphraseRequest	*This message contains a request for the device to delete a passphrase from the keystore.* tas:PassphraseID **PassphraseID** [1][1]
DeletePassphraseResponse	*This is an empty message.*
Fault codes	**Description**
env:Receiver ter:Action ter:PassphraseDeletionFailed	*Deleting the passphrase with the requested PassphraseID failed.*
env:Sender ter:InvalidArgVal ter:PassphraseID	*No passphrase is stored under the requested PassphraseID.*

5.2.6.2 Key Management

5.2.6.2.1 Create RSA Key Pair

This operation triggers the asynchronous generation of an RSA key pair of a particular keylength (specified as the number of bits) as specified in [RFC 3447], with a suitable key generation mechanism on the device. Keys, especially RSA key pairs, are uniquely identified using key IDs.

If the device does not have not enough storage capacity for storing the key pair to be created, the maximum number of keys reached fault shall be produced and no key pair shall be generated. Otherwise, the operation generates a keyID for the new key and associates the

generating status to it. Immediately after key generation has started, the device shall return the keyID to the client and continue to generate the key pair. The client may query the device with the GetKeyStatus operation (see Sect. 5.2.6.2.3) whether the generation has finished. The client may also subscribe to Key Status events (see Sect. 5.6.1) to be notified about key status changes.

The device also returns a best-effort estimate of how much time it requires to create the key pair.[1] A client may use this information as an indication how long to wait before querying the device whether key generation is completed.

After the key has been successfully created, the device shall assign it the *ok* status. If the key generation fails, the device shall assign the key the *corrupt* status.

Table 7:CreateRSAKeyPair command

CreateRSAKeyPair	Access Class: WRITE_SYSTEM
Message name	**Description**
CreateRSAKeyPairRequest	*This message contains a request for the device to generate an RSA key pair (i.e., a public and a private key).* xs:nonNegativeInteger **KeyLength** [1][1] xs:string **Alias** [0][1]
CreateRSAKeyPairResponse	*This message contains the key ID of the key pair being generated.* tas:KeyID **KeyID** [1][1] xs:duration **EstimatedCreationTime**[1][1]
Fault codes	**Description**
env:Receiver ter:Action ter:MaximumNumberOfKeysReached	*The keystore does not have enough storage space to store the key pair that has to be generated.*
env:Sender ter:InvalidArgVal ter:KeyLength	*The specified key length is not supported by the device.*

5.2.6.2.2 Upload Key Pair in PKCS#8

This operation uploads a key pair in a PKCS#8 data structure as specified in [RFC 5958, RFC 5959].

If an encryption passphrase ID is supplied in the request, the device shall assume that the KeyPair parameter contains an EncryptedPrivateKeyInfo ASN.1 structure that is encrypted under the passphrase in the keystore that corresponds to the supplied ID, where the EncryptedPrivateKeyInfo structure contains both the private key and the corresponding public key. If no encryption passphrase ID is supplied, the device shall assume that the KeyPair parameter contains a OneAsymmetricKey ASN.1 structure which contains both the private key and the corresponding public key. If a passphrase is supplied, the device shall ignore an eventually supplied passphrase ID and assume that the KeyPair parameter contains an EncryptedPrivateKeyInfo ASN.1 structure that is encrypted under the supplied passphrase, where the EncryptedPrivateKeyInfo structure contains both the private key and the corresponding public key.

[1]Implementors may estimate the key generation time for a fixed key length as the average elapsed time of a number of key generation operations for this key length.

If the supplied key pair cannot be processed by the device, the device shall produce an UnsupportedPublicKeyAlgorithm fault and shall not store the uploaded key pair in the keystore.

Key pairs are uniquely identified using key IDs. If a key pair exists in the keystore with the public key equal to the public key in the request and this key pair does not contain a private key, the device shall add the supplied private key to the existing key pair and return the ID of this key pair.

If a key pair exists in the keystore with the public key equal to the public key in the request and this key pair contains a private key, the device shall leave the key pair unchanged and return the ID of this key pair.

If the existing key pair does not have status *ok*, the device shall produce an InvalidKeyStatus fault and shall not modify the existing key pair.

If no key pair exists in the keystore with the public key equal to the public key in the request, the device shall generate a new key pair with the supplied private key and the supplied public key, status *ok* and the externally generated attribute set to *true*. Furthermore, the device shall return the ID of this key pair.

If a new key pair is created, the device shall assign the supplied alias to it. Otherwise, the device shall ignore an eventually supplied alias.

If decryption of the EncryptedPrivateKeyInfo failed, the device shall produce a DecryptionFailed fault and shall not store the uploaded key pair in the keystore.

If the device does not have not enough storage capacity for storing the key pair that eventually has to be created, the device shall generate a maximum number of keys reached fault. Furthermore the device shall not generate a key pair.

If no passphrase exists under the ID specified by EncryptionPassphraseID, the device shall produce an invalid passphrase ID fault and shall not store the uploaded key pair in the keystore.

If the supplied PKCS#8 data structure cannot be processed by the device, the device shall produce a BadPKCS8File fault and shall not store the uploaded key pair in the keystore.

If the public key in the uploaded key pair does not match the uploaded private key, the device shall produce a PublicPrivateKeyMismatch fault and shall not store the uploaded key pair in the keystore.

If the command was successful, the device shall return the ID of the key pair in the keystore that contains the supplied public and private key.

Table 8: UploadKeyPairInPKCS8 command

UploadKeyPairInPKCS8	Access Class: WRITE_SYSTEM
Message name	**Description**
UploadKeyPairInPKCS8Request	*This message contains a request for the device to upload a DER-encoded key pair in a PKCS#8 data structure.* tas:Base64DERencodedASN1Value **KeyPair** [1][1] xs:string **Alias** [0][1] tas:PassphraseID **EncryptionPassphraseID** [0][1] xs:string **EncryptionPassphrase**[0][1]
UploadKeyPairInPKCS8Response	*This message contains the ID of the successfully uploaded key pair.* tas:KeyID **KeyID** [1][1]
Fault codes	**Description**
env:Receiver ter:Action ter:MaximumNumberOfKeysReached	*The device does not have enough storage space to store the key pair that has to be generated.*
env:Sender ter:InvalidArgVal ter:PassphraseID	*No passphrase is stored under the requested PassphraseID.*
env:Sender ter:InvalidArgVal ter:DecryptionFailed	*The given data could not be decrypted.*
env:Sender ter:InvalidArgVal ter:UnsupportedPublicKeyAlgorithm	*The public key algorithm of the supplied key pair is not supported by the device.*
env:Sender ter:InvalidArgVal ter:InvalidKeyStatus	*The key with the requested KeyID has an inappropriate status.*
env:Sender ter:InvalidArgVal ter:BadPKCS8File	*The PKCS#8 data structure cannot be processed by the device.*
env:Sender ter:InvalidArgVal ter:PublicPrivateKeyMismatch	*The supplied private key does not match the supplied public key.*

5.2.6.2.3 Get Key Status

This operation returns the status of a key as defined in Sect. 5.2.5.

Keys are uniquely identified using key IDs. If no key is stored under the requested key ID in the keystore, an InvalidKeyID fault is produced. Otherwise, the status of the key is returned.

Table 9: GetKeyStatus command

GetKeyStatus	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetKeyStatusRequest	*This message contains a request for the device to return the status of a key in the keystore.* tas:KeyID **KeyID**[1][1]
GetKeyStatusResponse	*This message contains the status of the requested KeyID.* tas:KeyStatus **KeyStatus**[1][1]
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:KeyID	*No key is stored under the requested KeyID.*

5.2.6.2.4 Get Private Key Status (deprecated)

This operation returns whether a key pair contains a private key.

Keys are uniquely identified using key IDs. If no key is stored under the requested key ID in the keystore, an invalid key ID fault shall be produced. If a key is stored under the requested key ID in the keystore, but this key is not a key pair, an invalid key type fault shall be produced.

Otherwise, this operation returns *true* if the key pair identified by the key ID contains a private key, and *false* otherwise.

This command is deprecated. Use GetAllKeys (see Sect. 5.2.6.2.5) instead.

Table 10: GetPrivateKeyStatus command

GetPrivateKeyStatus	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetPrivateKeyStatusRequest	*This message contains a request for the device to return whether a key pair contains a private key.* tas:KeyID **KeyID**[1][1]
GetPrivateKeyStatusResponse	*This message contains the status for the requested KeyID.* xs:boolean **hasPrivateKey**[1][1]
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:KeyID	*No key is stored under the requested KeyID.*
env:Sender ter:InvalidArgVal ter:InvalidKeyType	*The key stored in the keystore under the requested KeyID is of an invalid type.*

5.2.6.2.5 Get All Keys

This operation returns information about all keys that are stored in the device's keystore.

This operation may be used, e.g., if a client lost track of which keys are present on the device.

If no key is stored on the device, an empty list is returned.

Table 11: GetAllKeys command

GetAllKeys	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetAllKeysRequest	*This message contains a request for the device to return information about all keys in the keystore.* *This is an empty message.*
GetAllKeysResponse	*This message contains information about all keys in the keystore.* tas:KeyAttribute **KeyAttribute** [0][unbounded]
Fault codes	**Description**
	No command-specific fault codes.

5.2.6.2.6 Delete Key

This operation deletes a key from the device's keystore.

Keys are uniquely identified using key IDs. If no key is stored under the requested key ID in the keystore, a device shall produce an InvalidArgVal fault. If a reference exists for the specified key, a device shall produce the corresponding fault and shall not delete the key. If there is a key under the requested key ID stored in the keystore and the key could not be deleted, a device shall produce a KeyDeletion fault. If the key has the status *generating*, a device shall abort the generation of the key and delete from the keystore all data generated for this key.

After a key is successfully deleted, the device may assign its former ID to other keys.

Table 12: DeleteKey command

DeleteKey	Access Class: UNRECOVERABLE
Message name	**Description**
DeleteKeyRequest	*This message contains a request for the device to delete a key from the keystore.* tas:KeyID **KeyID**[1][1]
DeleteKeyResponse	*This is an empty message.*
Fault codes	**Description**
env:Receiver ter:Action ter:KeyDeletionFailed	*Deleting the key with the requested KeyID failed.*
env:Sender ter:InvalidArgVal ter:KeyID	*No key is stored under the requested KeyID.*
env:Sender ter:InvalidArgVal ter:ReferenceExists	*A reference exists for the object that is to be deleted.*

5.2.6.3 Certificate Management

5.2.6.3.1 Create PKCS#10 Certification Request

This operation generates a DER-encoded PKCS#10 v1.7 certification request (sometimes also called certificate signing request or CSR) as specified in [RFC 2986] for a public key on the device.

The key pair that contains the public key for which a certification request shall be produced is specified by its key ID. If no key is stored under the requested KeyID or the key specified by the requested KeyID is not an asymmetric key pair, an invalid key ID fault shall be produced and no CSR shall be generated.

The subject parameter describes the entity that the public key belongs to. Additional attributes can be included in the attribute parameter.

Distinguished name attribute values shall be supplied either in UTF-8 or in hexadecimal form as specified in RFC 4514.

If the distinguished name attribute value is supplied in hexadecimal form, the device shall encode the attribute in the format given in the hexadecimal format.

If the distinguished name attribute value is supplied in UTF-8 and the attribute value has a uniquely defined encoding (e.g., CountryName is defined as PrintableString), the device shall encode the attribute as the defined encoding. Otherwise, the device shall encode the attribute value as UTF-8.

The signature algorithm parameter determines which signature algorithm shall be used for signing the certification request with the public key specified by the key ID parameter. If the specified signature algorithm is not supported by the device, an UnsupportedSignatureAlgorithm fault shall be produced and no CSR shall be generated. If the public key identified by the requested Key ID is an invalid input to the specified signature algorithm, a KeySignatureAlgorithmMismatch fault shall be produced and no CSR shall be generated. If the specified subject is invalid or incomplete, a Subject invalid fault shall be produced and no CSR shall be created. If an attribute is invalid or incomplete, an Attribute invalid fault shall be produced and no CSR shall be generated.

If the key pair does not have status *ok*, a device shall produce an InvalidKeyStatus fault and no CSR shall be generated.

Table 13: CreatePKCS10CSR command

CreatePKCS10CSR	Access Class: READ_SYSTEM
Message name	**Description**
CreatePKCS10CSRRequest	*This message contains a request for the device to create a PKCS#10 certification request for one of its public keys.* tas:DistinguishedName **Subject** [1][1] tas:KeyID **KeyID**[1][1] tas:CSRAttribute **Attribute** [0][unbounded] tas:AlgorithmIdentifier **SignatureAlgorithm**[1][1]
CreatePKCS10CSRResponse	*This message contains the DER encoded PKCS#10 certification request.* tas:Base64DERencodedASN1Value **PKCS10CSR** [1][1]
Fault codes	**Description**
env:Receiver ter:Action ter:CSRCreationFailed	*The generation of the PKCS#10 certification request failed.*
env:Sender ter:InvalidArgVal ter:KeyID	*No key is stored under the requested KeyID.*
env:Sender ter: InvalidArgVal ter:UnsupportedSignatureAlgorithm	*The specified signature algorithm is not supported by the device.*
env:Sender ter: InvalidArgValter:KeySignatureAlgorithmMismatch	*The specified public key is an invalid input to the specified signature algorithm.*
env:Sender ter:InvalidArgVal ter:InvalidKeyStatus	*The key with the requested KeyID has an inappropriate status.*
env:Sender ter:InvalidArgVal ter:InvalidSubject	*The specified subject is invalid or incomplete.*
env:Sender ter:InvalidArgVal ter:InvalidAttribute	*The specified attribute is invalid or incomplete.*

5.2.6.3.2 Create Self-Signed Certificate

This operation generates for a public key on the device a self-signed X.509 certificate that complies to [RFC 5280].

The X509Version parameter specifies the version of X.509 that the generated certificate shall comply to. A device that supports this command shall support the generation of X.509v3 certificates as specified in [RFC 5280] and may additionally be able to handle other X.509 certificate formats as indicated by the X.509Versions capability. If no X509Version is specified in the request, the device shall produce an X.509v3 certificate.

The key pair that contains the public key for which a self-signed certificate shall be produced is specified by its key pair ID. The subject parameter describes the entity that the public key belongs to.

If the key pair does not have status *ok*, a device shall produce an InvalidKeyStatus fault and no certificate shall be generated.

If the specified subject is invalid or incomplete, a Subject invalid fault shall be produced and no certificate shall be created.

The notValidBefore parameter specifies at which point in time the validity period of the generated certificate shall begin. If this parameter is not specified in the request, the device shall use its current time or a time before its current time as starting point of the validity period. The notValidAfter parameter specifies at which point in time the validity period of the generated certificate shall end. If this parameter is not specified in the request, the device shall assign the GeneralizedTime value of 99991231235959Z as specified in [RFC 5280] to the notValidAfter parameter. If the notValidBefore parameter is invalid, an invalid DateTime fault shall be produced and no certificate shall be generated. If the notValidAfter parameter is invalid, an invalid DateTime fault shall be produced and no certificate shall be generated.

The signature algorithm parameter determines which signature algorithm shall be used for signing the certification request with the public key specified by the key ID parameter.

The Extensions parameter specifies potential X509v3 extensions that shall be contained in the certificate. A device that supports this command shall support the extensions that are defined in [RFC5280, Sect. 4.2] as mandatory for CAs that issue self-signed certificates.

Distinguished name attribute values shall be supplied either in UTF-8 or in hexadecimal form as specified in RFC 4514.

If the distinguished name attribute value is supplied in hexadecimal form, the device shall encode the attribute inthe format given in the hexadecimal format.

If the distinguished name attribute value is supplied in UTF-8 and the attribute value has a uniquely defined encoding (e.g., CountryName is defined as PrintableString), the device shall encode the attribute as the defined encoding. Otherwise, the device shall encode the attribute value as UTF-8.

[RFC 5280, Sect. 4.1.2.2] mandates that the certificate serial numbers be unique for each certificate issued by a given issuer (a CA). Since the subject is equal to the issuer in a self-signed certificate, the serial number shall be unique for each self-signed certificate that the device issues for a given subject.

The generated certificate shall not contain a unique identifier as specified in [RFC 5280], Sect. 4.1.2.8. The device shall not mark the generated certificate as trusted.

Certificates are uniquely identified using certificate IDs. If the command was successful, the device generates a new ID for the generated certificate and returns this ID.

If the device does not have not enough storage capacity for storing the certificate to be created, the maximum number of certificates reached fault shall be produced and no certificate shall be generated.

Table 14: CreateSelfSignedCertificate command

CreateSelfSignedCertificate	Access Class: WRITE_SYSTEM
Message name	**Description**
CreateSelfSignedCertificateRequest	*This message contains a request for the device to create for a public key on the device a self-signed, RFC 5280 compliant certificate.* xs:positive **X509Version** [0][1] tas:DistinguishedName **Subject** [1][1] tas:KeyID **KeyID**[1][1] xs:string **Alias** [0][1] xs:dateTime **notValidBefore** [0][1] xs:dateTime **notValidAfter**[0][1] tas:AlgorithmIdentifier **SignatureAlgorithm**[1][1] tas:X509v3Extension **Extension** [0][unbounded]
CreateSelfSignedCertificateResponse	*This message contains the certificate ID of the successfully created certificate.* tas:CertificateID **CertificateID** [1][1]
Fault codes	**Description**
env:Receiver ter:Action ter:CertificateCreationFailed	*The generation of the self-signed certificate failed.*
env:Receiver ter:Action ter:MaximumNumberOfCertificatesReached	*The device does not have enough storage space to store the certificate to be created.*
env:Sender ter:InvalidArgVal ter:UnsupportedX509Version	*The specified X.509 version is not supported by the device.*
env:Sender ter:InvalidArgVal ter:KeyID	*No key is stored under the requested KeyID.*
env:Sender ter: InvalidArgVal ter:UnsupportedSignatureAlgorithm	*The specified signature algorithm is not supported by the device.*
env:Sender ter: InvalidArgValter:KeySignatureAlgorithmMismatch	*The specified public key is an invalid input to the specified signature algorithm.*
env:Sender ter: InvalidArgVal ter:X509VersionExtensionsMismatch	*The request contains extensions which are not supported by the X509Versionin the request.*
env:Sender ter:InvalidArgVal ter:InvalidKeyStatus	*The key with the requested KeyID has an inappropriate status.*
env:Sender ter:InvalidArgVal ter:InvalidSubject	*The specified subject is invalid or incomplete.*
env:Sender ter:InvalidArgVal ter:InvalidDateTime	*A specified dateTime is invalid.*

5.2.6.3.3 Upload Certificate

This operation uploads an X.509 certificate as specified by [RFC 5280]in DER encoding and the public key in the certificate to a device's keystore. A device that supports this command shall be able to handle X.509v3 certificates as specified in [RFC 5280] and may additionally be able to handle other X.509 certificate formats as indicated by the X.509Versions capability.

Certificates are uniquely identified using certificate IDs, and key pairs are uniquely identified using key IDs. The device shall generate a new certificate ID for the uploaded certificate.

Certain certificate usages, e.g. TLS server authentication, require the private key that corresponds to the public key in the certificate to be present in the keystore. In such cases, the client may indicate that it expects the device to produce a fault if the matching private key for the uploaded certificate is not present in the keystore by setting the PrivateKeyRequired argument in the upload request to *true*.

The uploaded certificate has to be linked to a key pair in the keystore.

If no private key is required for the public key in the certificate and a key pair exists in the keystore with a public key equal to the public key in the certificate, the uploaded certificate is linked to the key pair identified by the supplied key ID by adding a reference from the certificate to the key pair.

If no private key is required for the public key in the certificate and no key pair exists with the public key equal to the public key in the certificate, a new key pair with status *ok* is created with the public key from the certificate, and this key pair is linked to the uploaded certificate by adding a reference from the certificate to the key pair.

If a private key is required for the public key in the certificate, and a key pair exists in the keystore with a private key that matches the public key in the certificate, the uploaded certificate is linked to this key pair by adding a reference from the certificate to the key pair. If a private key is required for the public key and no such keypair exists in the keystore, the NoMatchingPrivateKey fault shall be produced and the certificate shall not be stored in the keystore.

The device shall assign the supplied Alias to the uploaded certificate.

If a new key pair is generated, the device shall assign the supplied KeyAlias to it. Otherwise, the device shall ignore an eventually supplied KeyAlias.

How the link between the uploaded certificate and a key pair is established is illustrated in Figure 2.

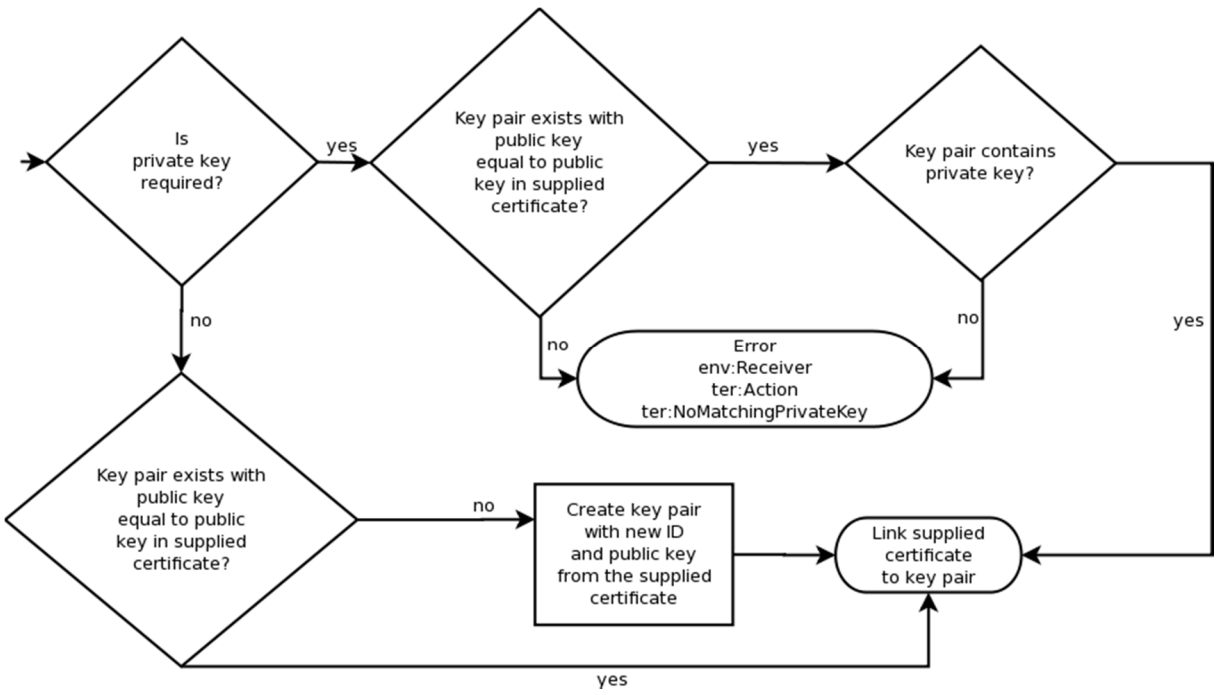

Figure 2 Link establishment between certificate and key pair for Upload Certificate

If the key pair that the certificate shall be linked to does not have status *ok*, an InvalidKeyID fault is produced, and the uploaded certificate is not stored in the keystore.

If the signature algorithm that the signature of the supplied certificate is based on is not supported by the device, the device shall generate an UnsupportedSignatureAlgorithm fault and shall not store the uploaded certificate nor the contained public key in the keystore.

If the device cannot process the uploaded certificate, a BadCertificate fault is produced and neither the uploaded certificate nor the public key are stored in the device's keystore. The BadCertificate fault shall not be produced based on the mere fact that the device's current time lies outside the interval defined by the notBefore and notAfter fields as specified by [RFC 5280], Sect. 4.1.

The device shall not mark the uploaded certificate as trusted.

If the device does not have not enough storage capacity for storing the certificate to be uploaded, the maximum number of certificates reached fault shall be produced and no certificate shall be uploaded.

If the device does not have not enough storage capacity for storing the key pair that eventually has to be created, the device shall generate a maximum number of keys reached fault. Furthermore the device shall not generate a key pair and no certificate shall be stored.

If the command was successful, the device returns the ID of the uploaded certificate and the ID of the key pair that contains the public key in the certificate.

Table 15: Upload Certificate command

UploadCertificate	Access Class: WRITE_SYSTEM
Message name	**Description**
UploadCertificateRequest	*This message contains a request for the device to upload a DER-encoded certificate to the keystore.* tas:Base64DERencodedASN1Value **Certificate** [1][1] xs:string **Alias** [0][1] xs:string **KeyAlias** [0][1] xs:boolean **PrivateKeyRequired** [0][1]
UploadCertificateResponse	*This message contains the ID of the successfully uploaded certificate and the ID of the key pair that contains the public key in the certificate.* tas:CertificateID **CertificateID** [1][1] tas:KeyID **KeyID** [1][1]
Fault codes	**Description**
env:Receiver ter:Action ter:MaximumNumberOfCertificatesReached	*The device does not have enough storage space to store the certificate to be created.*
env:Receiver ter:Action ter:MaximumNumberOfKeysReached	*The keystore does not have enough storage space to store the key pair that has to be generated.*
env:Sender ter:InvalidArgVal ter:BadCertificate	*The supplied certificate file cannot be processed by the device.*
env:Receiver ter:Action ter:NoMatchingPrivateKey	*The keystore does not contain a key pair with a private key that matches the public key in the uploaded certificate.*
env:Sender ter:InvalidArgVal ter:UnsupportedPublicKeyAlgorithm	*The public key algorithm of the public key in the certificate is not supported by the device.*
env:Sender ter:InvalidArgVal ter:UnsupportedSignatureAlgorithm	*The specified signature algorithm is not supported by the device.*
env:Sender ter:InvalidArgVal ter:InvalidKeyStatus	*The key with the requested KeyID has an inappropriate status.*

5.2.6.3.4 Upload Certificate with Private Key in PKCS#12

This operation uploads a certification path consisting of X.509 certificates as specified by [RFC 5280] in DER encoding along with a private key to a device's keystore. Certificates and private key are supplied in the form of a PKCS#12 file as specified in [PKCS#12].

The device shall support PKCS#12 files that contain the following safe bags:

- one or more instances of CertBag [PKCS#12, Sect. 4.2.3]
- either exactly one instance of KeyBag [PKCS#12, Sect. 4.3.1] or exactly one instance of PKCS8ShroudedKeyBag [PKCS#12, Sect. 4.2.2].

If the IgnoreAdditionalCertificates parameter has the value *true*, the device shall behave as if the client had supplied only the first CertBag in the sequence of CertBag instances.

The device shall support PKCS#12 passphrase integrity mode for integrity protection of the PKCS#12 PFX as specified in [PKCS#12, Sect. 4]. The device shall support PKCS8ShroudedKeyBags that are encrypted with the same passphrase as the CertBag instances.

If an integrity passphrase ID is supplied, the device shall use the corresponding passphrase in the keystore to check the integrity of the supplied PKCS#12 PFX. If an integrity passphrase ID is supplied, but the supplied PKCS#12 PFX has no integrity protection, the device shall produce a BadPKCS12File fault and shall not store the uploaded certificates nor the uploaded key pair in the keystore.

If an encryption passphrase ID is supplied, the device shall use the corresponding passphrase in the keystore to decrypt the PKCS8ShroudedKeyBag and the CertBag instances.

If an EncryptionPassphraseID is supplied, but a CertBag is not encrypted, the device shall ignore the supplied EncryptionPassphraseID when processing this CertBag. If an EncryptionPassphraseID is supplied, but a KeyBag is provided instead of a PKCS8ShroudedKeyBag, the device shall ignore the supplied EncryptionPassphraseID when processing the KeyBag.

If a passphrase is supplied, the device shall ignore an eventually supplied integrity passphrase ID and an eventually supplied encryption passphrase ID, and the device shall use the supplied passphrase to check the integrity of the PKCS#12 PFX and to decrypt the PKCS8ShroudedKeyBag and the CertBag instances. If a passphrase is supplied, but the supplied PKCS#12 PFX has no integrity protection, the device shall not check the integrity of the PKCS#12 PFX. If a passphrase is supplied, but a CertBag is not encrypted, the device shall ignore the supplied passphrase when processing this CertBag. If a passphrase is supplied, but a KeyBag is supplied instead of a PKCS8ShroudedKeyBag, the device shall ignore the supplied passphrase when processing the KeyBag.

If decryption of either the PKCS8ShroudedKeyBag or an encrypted CertBag failed, the device shall produce a DecryptionFailed fault and shall not store the uploaded certificates nor key pair in the keystore.

If the signature algorithm of a supplied certificate is not supported by the device, the device shall produce an UnsupportedSignatureAlgorithm fault and shall not upload a certificate nor key pair.

If the supplied key pair cannot be processed by the device, the device shall produce an UnsupportedPublicKeyAlgorithm fault and shall not store the uploaded key pair nor the uploaded certificates in the keystore.

Certificates are uniquely identified using certificate IDs. The device shall store the uploaded certificates in the keystore and generate a new certificate ID for each of the uploaded certificates.

Certification paths are uniquely identified using certification path IDs. The device shall create a certification path from the uploaded certificates. In this certification path, the certificates shall appear in the same order as in the PKCS#12 file. The device shall generate a new certification path ID for the created certification path and assign the eventually supplied CertificationPathAlias to the created certification path.

The signature of each certificate in the sequence of uploaded certificates except for the last one shall be verifiable with the public key contained in the next certificate in the sequence. If there is a certificate in the request other than the last certificate for which the signature cannot be verified with the public key in the next certificate, the device shall produce an invalid certification path fault and shall not store the uploaded certificates nor uploaded private key in the keystore.

If the device cannot process one of the uploaded certificates, it shall produce a BadCertificate fault and neither store the uploaded certificates nor private key in the keystore. The BadCertificate fault shall not be produced based on the mere fact that the device’s current time lies outside the interval defined by the notBefore and notAfter fields as specified by [RFC 5280], Sect. 4.1.

The device shall not mark the uploaded certificates as trusted.

The uploaded certificates have to be linked to key pairs in the keystore. Key pairs are uniquely identified using key IDs.

If a key pair exists in the keystore with the public key equal to the public key in a certificate in the request, the device shall link the uploaded certificate to the key pair in the keystore by adding a reference from the certificate to the key pair. If the key pair in the keystore does not contain a private key and the private key contained in the KeyBag or PKCS8ShroudedKeyBag that matches the public key in the key pair, the device shall add the private key contained in the KeyBag or PKCS8ShroudedKeyBag to the key pair.

If no key pair exists in the keystore with the public key equal to the public key in a certificate in the request, the device shall create a new key pair with status ok, externally generated attribute set to *true*, and the public and private keys from the request, and shall link this key pair to the uploaded certificate by adding a reference from the certificate to the key pair.

If a new key pair is created for the uploaded private key, the device shall assign the supplied KeyAlias to it. Otherwise, the device shall ignore an eventually supplied KeyAlias.

How the link between an uploaded certificate and a key pair is established is illustrated in Figure 3.

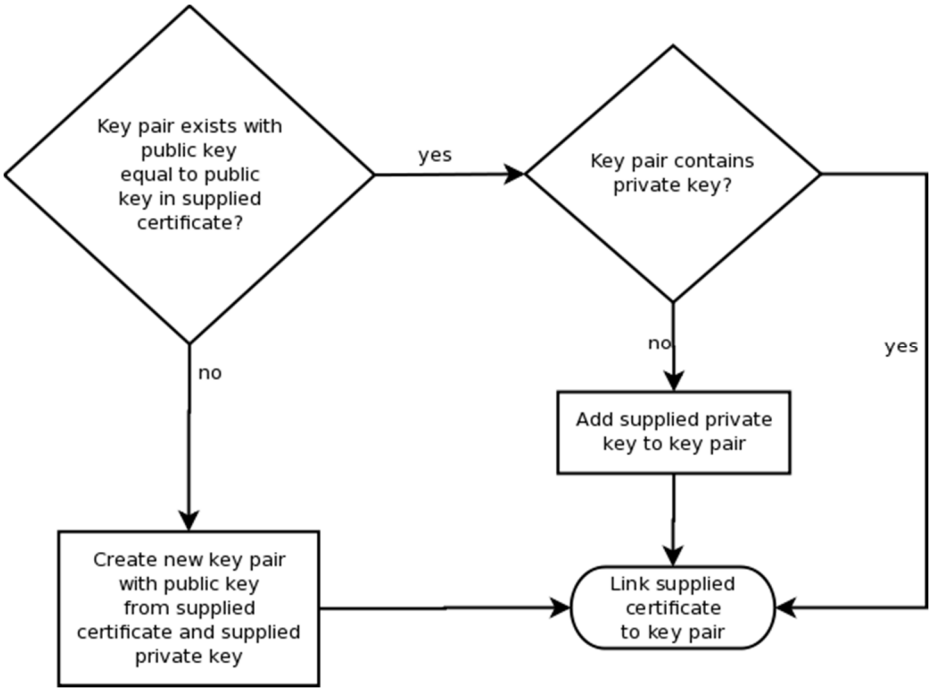

Figure 3 Link establishment between certificates and key pair for Upload Certificate with Private Key in PKCS#12

If the key pair that a certificate shall be linked to does not have status *ok*, the device shall produce an invalid key status fault and shall not store the uploaded certificates nor the uploaded key pair in the keystore.

If the device does not have not enough storage capacity for storing the certificates to be uploaded, the device shall produce a maximum number of certificates reached fault and shall not store the uploaded certificates nor the uploaded key pair in the keystore.

If the device does not have not enough storage capacity for storing the key pair that eventually has to be created, the device shall generate a maximum number of keys reached fault. Furthermore the device shall not store a key pair and shall not store the uploaded certificates in the keystore.

If the device does not have enough storage capacity for storing the certification path to be created, the device shall produce a maximum number of certification paths reached fault and shall not store the uploaded certificates nor the uploaded key pair in the keystore.

If no passphrase exists under the ID specified by IntegrityPassphraseID, the device shall produce an invalid passphrase ID fault and shall not store the uploaded certificates nor the uploaded key pair in the keystore.

If no passphrase exists under the ID specified by EncryptionPassphraseID, the device shall produce an invalid passphrase ID fault and shall not store the uploaded certificates nor the uploaded key pair in the keystore.

If the supplied PKCS#12 data structure cannot be processed by the device, the device shall produce a BadPKCS12File fault and shall not store the uploaded certificates nor the uploaded key pair in the keystore.

If the public key in the first uploaded certificate does not match the uploaded private key, the device shall produce a PublicPrivateKeyMismatch fault and shall not store the uploaded certificates nor the uploaded key pair in the keystore.

If the command was successful, the device shall return the ID of the created certification path and the ID of the key pair that contains the public key in the certificate.

Table 16: UploadCertificateWithPrivateKeyInPKCS12 command

UploadCertificateWithPrivateKeyInPKCS12	Access Class: WRITE_SYSTEM
Message name	**Description**
UploadCertificateWithPrivateKeyInPKCS12Request	*This message contains a request for the device to upload a path of DER-encoded certificates to the keystore along with a private key in a PKCS#12 data structure.* xs:Base64DERencodedASN1Value **CertWithPrivateKey** [1][1] xs:string **CertificationPathAlias** [0][1] xs:string **KeyAlias** [0][1] xs:boolean **IgnoreAdditionalCertificates**[0][1] tas:PassphraseID **IntegrityPassphraseID** [0][1] tas:PassphraseID **EncryptionPassphraseID** [0][1] xs:string **Passphrase** [0][1]
UploadCertificateWithPrivateKeyInPKCS12Response	*This message contains the ID of the successfully uploaded certificate and the ID of the key pair that contains the public key in the certificate.* tas:CertificationPathID **CertificationPathID** [1][1] tas:KeyID **KeyID** [1][1]
Fault codes	**Description**
env:Receiver ter:Action ter:MaximumNumberOfCertificatesReached	*The device does not have enough storage space to store the certificate to be uploaded.*
env:Receiver ter:Action ter:MaximumNumberOfKeysReached	*The device does not have enough storage space to store the key pair that has to be generated.*
env:Receiver ter:Action ter:MaximumNumberOfCertificationPathsReached	*The device does not have enough storage space to store the certification path to be uploaded.*
env:Sender ter: InvalidArgVal ter:PassphraseID	*No passphrase is stored under the requested PassphraseID.*
env:Sender ter:InvalidArgVal ter:DecryptionFailed	*The given data could not be decrypted.*

env:Sender ter:InvalidArgVal ter:BadCertificate	*The supplied certificate file cannot be processed by the device.*
env:Sender ter:InvalidArgVal ter:UnsupportedPublicKeyAlgorithm	*The public key algorithm of the public key in the certificate is not supported by the device.*
env:Sender ter:InvalidArgVal ter:UnsupportedSignatureAlgorithm	*The signature algorithm that the signature of the supplied certificate is based on is not supported by the device.*
env:Sender ter:InvalidArgVal ter:InvalidKeyStatus	*The key with the requested KeyID has an inappropriate status.*
env:Sender ter:InvalidArgVal ter:BadPKCS12File	*The PKCS#12 data structure cannot be processed by the device.*
env:Sender ter:InvalidArgVal ter:PublicPrivateKeyMismatch	*The supplied private key does not match the supplied public key.*
env:Sender ter: InvalidArgVal ter:InvalidCertificationPath	*At least one certificate in the certification path is not correctly signed with the public key in the next certificate in the path.*

5.2.6.3.5 Get Certificate

This operation returns a specific certificate from the device's keystore.

Certificates are uniquely identified using certificate IDs. If no certificate is stored under the requested certificate ID in the keystore, an InvalidArgVal fault is produced.

The certificate shall be returned in DER encoding.

It shall be noted that this command does not return the private key that is associated with the public key in the certificate.

Table 17: GetCertificate command

GetCertificate	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetCertificateRequest	*This message contains a request for the device to return a certificate from the keystore.* tas:CertificateID **CertificateID**[1][1]
GetCertificateResponse	*This message contains in DER encoding the certificate that is stored in the keystore under the given ID.* tas:X509Certificate **Certificate**[1][1]
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:CertificateID	*No certificate is stored under the requested CertificateID.*

5.2.6.3.6 Get All Certificates

This operation returns all certificates that are stored in the device's keystore.

This operation may be used, e.g., if a client lost track of which certificates are present on the device.

The certificates shall be returned in DER encoding.

If no certificate is stored in the device's keystore, an empty list is returned.

Table 18: GetAllCertificates command

GetAllCertificates	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetAllCertificatesRequest	*This message contains a request for the device to return all certificates from the keystore.* *This is an empty message.*
GetAllCertificatesResponse	*This message contains in DER encoding all certificates in the keystore and their certificate IDs.* tas:X509Certificate **Certificate** [0][unbounded]
Fault codes	**Description**
	No command-specific fault codes.

5.2.6.3.7 Delete Certificate

This operation deletes a certificate from the device's keystore.

The operation shall not delete the public key that is contained in the certificate from the keystore.

Certificates are uniquely identified using certificate IDs. If no certificate is stored under the requested certificate ID in the keystore, an InvalidArgVal fault is produced. If there is a certificate under the requested certificate ID stored in the keystore and the certificate could not be deleted, a CertificateDeletion fault is produced.

If a reference exists for the specified certificate, the certificate shall not be deleted and the corresponding fault shall be produced.

After a certificate has been successfully deleted, the device may assign its former ID to other certificates.

Table 19: DeleteCertificate command

DeleteCertificate	Access Class: UNRECOVERABLE
Message name	**Description**
DeleteCertificateRequest	*This message contains a request for the device to delete a certificate from the keystore.* tas:CertificateID **CertificateID**[1][1]
DeleteCertificateResponse	*This is an empty message.*
Fault codes	**Description**
env:Receiver ter:Action ter:CertificateDeletionFailed	*Deleting the certificate with the requested CertificateID failed.*
env:Sender ter:InvalidArgVal ter:CertificateID	*No certificate is stored under the requested CertificateID.*
env:Sender ter:InvalidArgVal ter:ReferenceExists	*A reference exists for the specified certificate.*

5.2.6.3.8 Create Certification Path

This operation creates a sequence of certificates that may be used, e.g., for certification path validation or for TLS server authentication.

Certification paths are uniquely identified using certification path IDs. Certificates are uniquely identified using certificate IDs. A certification path contains a sequence of certificate IDs.

If there is a certificate ID in the sequence of supplied certificate IDs for which no certificate exists in the device's keystore, the corresponding fault shall be produced and no certification path shall be created.

The signature of each certificate in the certification path except for the last one shall be verifiable with the public key contained in the next certificate in the path. If there is a certificate ID in the request other than the last ID for which the corresponding certificate cannot be verified with the public key in the certificate identified by the next certificate ID, an InvalidCertificateChain fault shall be produced and no certification path shall be created.

Table 20: CreateCertificationPath command

CreateCertificationPath	Access Class: WRITE_SYSTEM
Message name	**Description**
CreateCertificationPathRequest	*This message contains a request for the device to create a certification path.* tas:CertificateIDs **CertificateIDs**[1][1] xs:string **Alias** [0][1]
CreateCertificationPathResponse	*This message contains the ID of the newly generated certification path.* tas:CertificationPathID **CertificationPathID** [1][1]
Fault codes	**Description**
env:Receiver ter:Action ter:MaximumNumberOfCertificationPathsReached	*The maximum number of certification paths that may be assigned to the TLS server simultaneously is reached.*
env:Sender ter:InvalidArgVal ter:CertificateID	*No certificate is stored under the requested CertificateID.*
env:Sender ter:InvalidArgVal ter:InvalidCertificationPath	*At least one certificate in the certification path is not correctly signed with the public key in the next certificate in the path.*
env:Receiver ter:Action ter:CertificationPathCreationFailed	*Creating the certification path failed.*

5.2.6.3.9 Get Certification Path

This operation returns a specific certification path from the device's keystore.

Certification paths are uniquely identified using certification path IDs. If no certification path is stored under the requested ID in the keystore, an InvalidArgVal fault is produced.

Table 21: GetCertificationPath command

GetCertificationPath	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetCertificationPathRequest	*This message contains a request for the device to return a certification path from the keystore.* tas:CertificationPathID **CertificationPathID** [1][1]
GetCertificationPathResponse	*This message contains the certification path that is stored under the given ID in the keystore.* tas:CertificationPath **CertificationPath**[1][1]
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:CertificationPathID	*No certification path is stored under the requested certification path ID.*

5.2.6.3.10 Get All Certification Paths

This operation returns the IDs of all certification paths that are stored in the device's keystore.

This operation may be used, e.g., if a client lost track of which certificates are present on the device.

If no certification path is stored on the device, an empty list is returned.

Table 22: GetAllCertificationPaths command

GetAllCertificationPaths	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetAllCertificationPathsRequest	*This message contains a request for the device to return the IDs of all certification paths in the keystore.* *This is an empty message.*
GetAllCertificationPathsResponse	*This message contains the IDs of all certification paths in the keystore.* tas:CertificationPathID **CertificationPathID** [0][unbounded]
Fault codes	**Description**
	No command-specific fault codes.

5.2.6.3.11 Delete Certification Path

This operation deletes a certification path from the device's keystore.

This operation shall not delete the certificates that are referenced by the certification path.

Certification paths are uniquely identified using certification path IDs. If no certification path is stored under the requested certification path ID in the keystore, an InvalidArgVal fault is produced. If there is a certification path under the requested certification path ID stored in the keystore and the certification path could not be deleted, a CertificationPathDeletion fault is produced.

If a reference exists for the specified certification path, the certification path shall not be deleted and the corresponding fault shall be produced.

After a certification path is successfully deleted, the device may assign its former ID to other certification paths.

Table 23: DeleteCertificationPath command

DeleteCertificationPath	Access Class: UNRECOVERABLE
Message name	**Description**
DeleteCertificationPathRequest	*This message contains a request for the device to delete a certification path.* tas:CertificationPathID **CertificationPathID**[1][1]
DeleteCertificationPathResponse	*This message is empty.*
Fault codes	**Description**
env:Receiver ter:Action ter:CertificationPathDeletionFailed	*Deleting the certification path with the requested certification path ID failed.*
env:Sender ter:InvalidArgVal ter:CertificationPathID	*No certification path is stored under the requested certification path ID.*
env:Sender ter:InvalidArgVal ter:ReferenceExists	*A reference exists for the object that is to be deleted.*

5.2.6.4 CRL Management

5.2.6.4.1 Upload Certificate Revocation List

This operation uploads a certificate revocation list (CRL) as specified in [RFC 5280] to the keystore on the device.

If the device does not have enough storage space to store the CRL to be uploaded, the device shall produce a MaximumNumberOfCRLsReached fault and shall not store the supplied CRL.

If the device is not able to process the supplied CRL, the device shall produce a BadCRL fault and shall not store the supplied CRL.

If the device does not support the signature algorithm that was used to sign the supplied CRL, the device shall produce an UnsupportedSignatureAlgorithm fault and shall not store the supplied CRL.

Table 24: UploadCRL command

UploadCRL	Access Class: WRITE_SYSTEM
Message name	**Description**
UploadCRLRequest	*This message contains a request for the device to upload a CRL.* tas:Base64DERencodedASN1Value **Crl** [1][1] xs:string **Alias** [0][1]
UploadCRLResponse	*This message contains the ID of the successfully uploaded CRL.* tas:CRLID **CrlID** [1][1]
Fault codes	**Description**
env:Receiver ter:Action ter:MaximumNumberOfCRLsReached	*The device does not have enough storage space to store the CRL to be uploaded.*
env:Sender ter:InvalidArgVal ter:BadCRL	*The supplied CRL cannot be processed by the device.*
env:Sender ter:InvalidArgVal ter:UnsupportedSignatureAlgorithm	*The specified signature algorithm is not supported by the device.*

5.2.6.4.2 Get Certificate Revocation List

This operation returns a specific certificate revocation list (CRL) from the keystore on the device.

Certification revocation lists are uniquely identified using CRLIDs. If no CRL is stored under the requested CRLID, the device shall produce a CRLID fault.

Table 25: GetCRL command

GetCRL	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetCRLRequest	*This message contains a request for the device to return a CRL from the keystore.* tas:CRLID **CrlID** [1][1]
GetCRLResponse	*This message contains the CRL that is stored under the given ID in the keystore.* tas:CRL **Crl** [1][1]
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:CRLID	*No CRL is stored under the requested CRL ID.*

5.2.6.4.3 Get All Certificate Revocation Lists

This operation returns all certificate revocation lists (CRLs) that are stored in the keystore on the device.

If no certificate revocation list is stored in the device's keystore, an empty list is returned.

Table 26: GetAllCRLs command

GetAllCRLs	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetAllCRLsRequest	*This message contains a request for the device to return all CRLs in the keystore.* *This is an empty message.*
GetAllCRLsResponse	*This message contains all CRLs in the keystore.* tas:CRL **Crl** [0][unbounded]
Fault codes	**Description**
	No command-specific fault codes.

5.2.6.4.4 Delete Certificate Revocation List

This operation deletes a certificate revocation list (CRL) from the keystore on the device.

Certification revocation lists are uniquely identified using CRLIDs. If no CRL is stored under the requested CRLID, the device shall produce a CRLID fault.

If a reference exists for the specified CRL, the device shall produce a ReferenceExists fault and shall not delete the CRL.

After a CRL has been successfully deleted, a device may assign its former ID to other CRLs.

Table 27: DeleteCRL command

DeleteCRL	Access Class: UNRECOVERABLE
Message name	**Description**
DeleteCRLRequest	*This message contains a request for the device to delete a CRL from the keystore.* tas:CRLID **CrlID** [1][1]
DeleteCRLResponse	*This is an empty message.*
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:CRLID	*No CRL is stored under the requested CRL ID.*
env:Sender ter:InvalidArgVal ter:ReferenceExists	*A reference exists for the object that is to be deleted.*

5.2.6.5 Certification Path Validation Policy Management

5.2.6.5.1 Create Certification Path Validation Policy

This operation creates a certification path validation policy.

Certification path validation policies are uniquely identified using certification path validation policy IDs. The device shall generate a new certification path validation policy ID for the created certification path validation policy.

For the certification path validation parameters that are not represented in the certPathValidationParameters data type, the device shall use the default values specified in Sect. 3.

If the device does not have enough storage capacity for storing the certification path validation policy to be created, the device shall produce a maximum number of certification path validation policies reached fault and shall not create a certification path validation policy.

If there is at least one trust anchor certificate ID in the request for which there exists no certificate in the device's keystore, the device shall produce a CertificateID fault and shall not create a certification path validation policy.

If the device cannot process the supplied certification path validation parameters, the device shall produce a CertPathValidationParameters fault and shall not create a certification path validation policy.

Table 28: CreateCertPathValidationPolicy command

CreateCertPathValidationPolicy	Access Class: WRITE_SYSTEM
Message name	**Description**
CreateCertPathValidationPolicyRequest	*This message contains a request for the device to create a certification path validation policy.* xs:string **Alias** [0][1] tas:CertPathValidationParameters **Parameters** [1][1] tas:TrustAnchor **TrustAnchor** [1][unbounded]
CreateCertPathValidationPolicyResponse	*This message contains the ID of the successfully created certification path validation policy.* tas:CertPathValidationPolicyID **CertPathValidationPolicyID** [1][1]
Fault codes	**Description**
env:Receiver ter:Action ter:MaximumNumberOfCertPathValidationPoliciesReached	*The device does not have enough storage to store the certification path validation policy to be created.*
env:Sender ter:InvalidArgVal ter:CertificateID	*No certificate is stored under the requested CertificateID.*
env:Sender ter:InvalidArgVal ter:CertPathValidationParameters	*The specified certification path validation parameters are invalid.*

5.2.6.5.2 Get Certification Path Validation Policy

This operation returns a certification path validation policy from the keystore on the device.

Certification path validation policies are uniquely identified using certification path validation policy IDs. If no certification path validation policy is stored under the requested certification path validation policy ID, the device shall produce a CertPathValidationPolicyID fault.

Table 29: GetCertPathValidationPolicy command

GetCertPathValidationPolicy	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetCertPathValidationPolicyRequest	*This message contains a request for the device to return a certification path validation policy from the keystore.* tas:CertPathValidationPolicyID **CertPathValidationPolicyID** [1][1]
GetCertPathValidationPolicyResponse	*This message contains the certification path validation policiy that is stored in the keystore under the given ID.* tas:CertPathValidationPolicy **CertPathValidationPolicy** [1][1]
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:CertPathValidationPolicyID	*No certification path validation policy is stored under the requested certification path validation policy ID.*

5.2.6.5.3 Get All Certification Path Validation Policies

This operation returns all certification path validation policies that are stored in the keystore on the device.

If no certification path validation policy is stored in the device's keystore, an empty list is returned.

Table 30: GetAllCertPathValidationPolicies command

GetAllCertPathValidationPolicies	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetAllCertPathValidationPoliciesRequest	*This message contains a request for the device to return all certification path validation policies in the keystore.* *This is an empty message.*
GetAllCertPathValidationPoliciesResponse	*This message contains all certification path validation policies in the keystore.* tas:CertPathValidationPolicy **CertPathValidationPolicy** [0][unbounded]
Fault codes	**Description**
	No command-specific fault codes.

5.2.6.5.4 Delete Certification Path Validation Policy

This operation deletes a certification path validation policy from the keystore on the device.

Certification path validation policies are uniquely identified using certification path validation policy IDs. If no certification path validation policy is stored under the requested certification path validation policy ID, the device shall produce an CertPathValidationPolicyID fault.

If a reference exists for the requested certification path validation policy, the device shall produce a ReferenceExists fault and shall not delete the certification path validation policy.

After the certification path validation policy has been deleted, the device may assign its former ID to other certification path validation policies.

Table 31: DeleteCertPathValidationPolicy command

DeleteCertPathValidationPolicy	Access Class: UNRECOVERABLE
Message name	**Description**
DeleteCertPathValidationPolicyRequest	*This message contains a request for the device to delete a certification path validation policy from the keystore.* tas:CertPathValidationPolicyID **CertPathValidationPolicyID** [1][1]
DeleteCertPathValidationPolicyResponse	*This is an empty message.*
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:CertPathValidationPolicyID	*No certification path validation policy is stored under the requested certification path validation policy ID.*
env:Sender ter:InvalidArgVal ter:ReferenceExists	*A reference exists for the object that is to be deleted.*

5.3 TLS Server

5.3.1 Elements of the TLS Server

The TLS server security feature implements a TLS server as specified in [RFC 2246] and subsequent specifications.

This specification defines how to manage the associations between certification paths and the TLS server. All other TLS server configuration actions are outside the scope of this specification. In particular, enabling and disabling the TLS server on the device shall be performed using the device management service specified in the [ONVIF Core Specification].

5.3.2 TLS Server Operations

5.3.2.1 Add Server Certificate Assignment

This operation assigns a key pair and certificate along with a certification path (certificate chain) to the TLS server on the device. The TLS server shall use this information for key exchange during the TLS handshake, particularly for constructing server certificate messages as specified in [RFC 4346, RFC 2246].

Certification paths are identified by their certification path IDs in the keystore. The first certificate in the certification path shall be the TLS server certificate.

Since each certificate has exactly one associated key pair, a reference to the key pair that is associated with the server certificate is not supplied explicitly. Devices shall obtain the private key or results of operations under the private key by suitable internal interaction with the keystore.

If a device chooses to perform a TLS key exchange based on the supplied certification path, it shall use the key pair that is associated with the server certificate for key exchange and transmit the certification path to TLS clients as-is, i.e., the device shall not check conformance of the certification path to [RFC 4346, RFC 2246].

In order to use the server certificate during the TLS handshake, the corresponding private key is required. Therefore, if the key pair that is associated with the server certificate, i.e., the first certificate in the certification path, does not have an associated private key, the NoPrivateKey fault is produced and the certification path is not associated with the TLS server.

A TLS server may present different certification paths to different clients during the TLS handshake instead of presenting the same certification path to all clients. Therefore more

than one certification path may be assigned to the TLS server. If the maximum number of certification paths that may be assigned to the TLS server simultaneously is reached, the device shall generate a MaximumNumberOfTLSCertificationPathsReached fault and the requested certification path shall not be assigned to the TLS server.

If the certification path identified by the supplied certification path ID is already assigned to the TLS server, this command shall have no effect.

Table 32: AddServerCertificateAssignment command

AddServerCertificateAssignment	Access Class:WRITE_SYSTEM
Message name	**Description**
AddServerCertificateAssignmentRequest	*This message contains a request for the device to assign a certificate along with a certification path to the TLS server.* tas:CertificationPathID **CertificationPathID**[1][1]
AddServerCertificateAssignmentResponse	*This is an empty message.*
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:CertificationPathID	*No certification path is stored in the keystore under the given certification path ID.*
env:Sender ter:InvalidArgVal ter:NoPrivateKey	*The key pair that is associated with the first certificate in the certificate chain does not have an associated private key.*
env: Receiver ter: Action ter:MaximumNumberOfTLSCertificationPathsReached	*The maximum number of certification paths that may be assigned to the TLS server simultaneously is reached.*

5.3.2.2 Remove Server Certificate Assignment

This operation removes a key pair and certificate assignment (including certification path) to the TLS server on the device.

Certification paths are identified using certification path IDs. If the supplied certification path ID is not associated with the TLS server, an InvalidArgVal fault is produced.

If the TLS server on the device is enabled, the device shall produce a ReferenceExists fault and shall not remove the server certificate assignment.

Table 33: RemoveServerCertificateAssignment command

RemoveServerCertificateAssignment	Access Class:WRITE_SYSTEM
Message name	**Description**
RemoveServerCertificateAssignmentRequest	*This message contains a request for the device to remove a TLS server certificate assignment along with a corresponding certification path from the TLS server.* tas:CertificationPathID **CertificationPathID**[1][1]
RemoveServerCertificateAssignmentResponse	*This is an empty message.*
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:OldCertificationPathID	*No certification path under the given certification path ID is associated with the TLS server.*
env:Sender ter:InvalidArgVal ter:ReferenceExists	*A reference exists for the object that is to be deleted.*

5.3.2.3 Replace Server Certificate Assignment

This operation replaces an existing key pair and certificate assignment to the TLS server on the device by a new key pair and certificate assignment (including certification paths).

After the replacement, the TLS server shall use the new certificate and certification path exactly in those cases in which it would have used the old certificate and certification path. Therefore, especially in the case that several server certificates are assigned to the TLS server, clients that wish to replace an old certificate assignment by a new assignment should use this operation instead of a combination of the Add TLS Server Certificate Assignment and the Remove TLS Server Certificate Assignment operations.

Certification paths are identified using certification path IDs. If the supplied old certification path ID is not associated with the TLS server, or no certification path exists under the new certification path ID, the corresponding InvalidArgVal faults are produced and the associations are unchanged.

The first certificate in the new certification path shall be the TLS server certificate.

Since each certificate has exactly one associated key pair, a reference to the key pair that is associated with the new server certificate is not supplied explicitly. Devices shall obtain the private key or results of operations under the private key by suitable internal interaction with the keystore.

If a device chooses to perform a TLS key exchange based on the new certification path, it shall use the key pair that is associated with the server certificate for key exchange and transmit the certification path to TLS clients as-is, i.e., the device shall not check conformance of the certification path to [RFC 4346, RFC 2246].

In order to use the server certificate during the TLS handshake, the corresponding private key is required. Therefore, if the key pair that is associated with the server certificate, i.e., the first certificate in the certification path, does not have an associated private key, the NoPrivateKey fault is produced and the certification path is not associated with the TLS server.

Table 34: ReplaceServerCertificateAssignment command

ReplaceServerCertificateAssignment	Access Class:WRITE_SYSTEM
Message name	**Description**
ReplaceServerCertificateAssignmentRequest	*This message contains a request for the device to replace a TLS server certificate assignment to the TLS server by a new key pair and certificate assignment.* tas:CertificationPathID **OldCertificationPathID**[1][1] tas:CertificationPathID **NewCertificationPathID**[1][1]
ReplaceServerCertificateAssignmentResponse	*This is an empty message.*
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:OldCertificationPathID	*No certification path under the given certification path ID is associated with the TLS server.*
env:Sender ter:InvalidArgVal ter:NewCertificationPathID	*No certification path is stored in the keystore under the given certification path ID.*
env:Sender ter:InvalidArgVal ter:NoPrivateKey	*The key pair that is associated with the first certificate in the certificate chain does not have an associated private key.*

5.3.2.4 Get Assigned Server Certificates

This operation returns the IDs of all certification paths that are assigned to the TLS server on the device.

This operation may be used, e.g., if a client lost track of the certification path assignments on the device.

If no certification path is assigned to the TLS server, an empty list is returned.

Table 35: GetAssignedServerCertificates command

GetAssignedServerCertificates	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetAssignedServerCertificatesRequest	*This message contains a request for the device to return the IDs of all certification paths that are assigned to the TLS server on the device.* *This is an empty message*
GetAssignedServerCertificatesResponse	*This message contains the IDs of all certification paths that are assigned to the TLS server on the device.* tas:CertificationPathID **CertificationPathID** [0][unbounded]
Fault codes	**Description**
	No command-specific fault codes.

5.3.2.5 Set Client Authentication Required

This operation activates or deactivates TLS client authentication for the TLS server on the device.

The TLS server on the device shall require client authentication if and only if clientAuthenticationRequired is set to *true*.

If TLS client authentication is requested to be enabled and no certification path validation policy is assigned to the TLS server, the device shall return an EnablingClientAuthenticationFailed fault and shall not enable TLS client authentication.

The device shall execute this command regardless of the TLS enabled/disabled state configured in the [ONVIF Device Management Service].

Table 36: SetClientAuthenticationRequired command

SetClientAuthenticationRequired	Access Class: WRITE_SYSTEM
Message name	**Description**
SetClientAuthenticationRequiredRequest	*This message contains a request for the device to activate or deactivate TLS client authentication.* xs:boolean **clientAuthenticationRequired** [1][1]
SetClientAuthenticationRequiredResponse	*This is an empty message.*
Fault codes	**Description**
env:Receiver ter:ActionNotSupported ter:EnablingClientAuthenticationFailed	*The device does not support TLS client authentication, or TLS client authentication is not configured appropriately.*

5.3.2.6 Get Client Authentication Required

This operation returns whether TLS client authentication is active.

Table 37: GetClientAuthenticationRequired command

GetClientAuthenticationRequired	Access Class: READ_SYSTEM
Message name	**Description**
GetClientAuthenticationRequiredRequest	*This message contains a request for the device to return whether TLS client authentication is active.* *This is an empty message.*
GetClientAuthenticationRequiredResponse	*This message contains whether TLS client authentication is active.* xs:boolean **clientAuthenticationRequired** [1][1]
Fault codes	**Description**
	No command-specific faults.

5.3.2.7 Add Certification Path Validation Policy Assignment

This operation assigns a certification path validation policy to the TLS server on the device. The TLS server shall enforce the policy when authenticating TLS clients and consider a client authentic if and only if the algorithm defined in Sect. 4.2.2 returns *valid*.

If no certification path validation policy is stored under the requested CertPathValidationPolicyID, the device shall produce a CertPathValidationPolicyID fault.

A TLS server may use different certification path validation policies to authenticate clients. Therefore more than one certification path validation policy may be assigned to the TLS server. If the maximum number of certification path validation policies that may be assigned to the TLS server simultaneously is reached, the device shall produce a MaximumNumberOfTLSCertPathValidationPoliciesReached fault and shall not assign the requested certification path validation policy to the TLS server.

Table 38: AddCertPathValidationPolicyAssignment command

AddCertPathValidationPolicyAssignment	Access Class: WRITE_SYSTEM
Message name	**Description**
AddCertPathValidationPolicyAssignmentRequest	*This message contains a request for the device to add a certification path validation policy assignment to the TLS server.* tas:CertPathValidationPolicyID **CertPathValidationPolicyID** [1][1]
AddCertPathValidationPolicyAssignmentResponse	*This is an empty message.*
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:CertPathValidationPolicyID	*No certification path validation policy is stored under the requested CertPathValidationPolicyID.*
env:Receiver ter:Action ter:MaximumNumberOfTLSCertPathValidationPoliciesReached	*The maximum number of certification path validation policies that may be assigned to the TLS server simultaneously is reached.*

5.3.2.8 Remove Certification Path Validation Policy Assignment

This operation removes a certification path validation policy assignment from the TLS server on the device.

If the certification path validation policy identified by the requested CertPathValidationPolicyID is not associated to the TLS server, the device shall produce a CertPathValidationPolicyID fault.

Table 39: RemoveCertPathValidationPolicyAssignment command

RemoveCertPathValidationPolicyAssignment	Access Class: WRITE_SYSTEM
Message name	**Description**
RemoveCertPathValidationPolicyAssignmentRequest	*This message contains a request for the device to remove a certification path validation policy assignment from the TLS server.* tas:CertPathValidationPolicyID **CertPathValidationPolicyID** [1][1]
RemoveCertPathValidationPolicyAssignmentResponse	*This is an empty message.*
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:CertPathValidationPolicyID	*No certification path validation policy is stored under the requested CertPathValidationPolicyID.*

5.3.2.9 Replace Certification Path Validation Policy Assignment

This operation replaces a certification path validation policy assignment to the TLS server on the device with another certification path validation policy assignment.

If the certification path validation policy identified by the requested OldCertPathValidationPolicyID is not associated to the TLS server, the device shall produce an OldCertPathValidationPolicyID fault and shall not associate the certification path validation policy identified by the NewCertPathValidationPolicyID to the TLS server.

If no certification path validation policy exists under the requested NewCertPathValidationPolicyID in the device's keystore, the device shall produce a NewCertPathValidationPolicyID fault and shall not remove the association of the old certification path validation policy to the TLS server.

Table 40: ReplaceCertPathValidationPolicyAssignment command

ReplaceCertPathValidationPolicyAssignment	Access Class: WRITE_SYSTEM
Message name	**Description**
ReplaceCertPathValidationPolicyAssignmentRequest	*This message contains a request for the device to replace a certification path validation policy assignment to the TLS server with another certification path validation policy assignment.* tas:CertPathValidationPolicyID **OldCertPathValidationPolicyID** [1][1] tas:CertPathValidationPolicyID **NewCertPathValidationPolicyID** [1][1]
ReplaceCertPathValidationPolicyAssignmentResponse	*This is an empty message.*
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:OldCertPathValidationPolicyID	*No certification path validation policy under the given OldCertPathValidationPolicyID is associated with the TLS server.*
env:Sender ter:InvalidArgVal ter:NewCertPathValidationPolicyID	*No certification path validation policy under the given NewCertPathValidationPolicyID is stored in the device's keystore.*

5.3.2.10 Get Assigned Certification Path Validation Policies

This operation returns the IDs of all certification path validation policies that are assigned to the TLS server on the device.

Table 41: GetAssignedCertPathValidationPolicies command

GetAssignedCertPathValidationPolicies	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetAssignedCertPathValidationPoliciesRequest	*This message contains a request for the device to return the IDs of all certification path validation policies that are assigned to the TLS server.* *This is an empty message.*
GetAssignedCertPathValidationPoliciesResponse	*This message contains the IDs of all certification path validation policies that are assigned to the TLS server.* tas:CertPathValidationPolicyID **CertPathValidationPolicyID** [0][unbounded]
Fault codes	**Description**
	No command-specific fault codes.

5.4 IEEE 802.1X

5.4.1 Add IEEE 802.1X Configuration

This operation adds an IEEE 802.1X configuration to the device.

Configurations are uniquely identified using IEEE 802.1X configuration IDs. The device shall ignore Dot1XID in the request, if present, and shall generate a unique configuration ID for the added configuration.

If the command was successful, the device shall return the ID of the configuration.

If the device does not have capacity for the configuration, the device shall produce a MaximumNumberOfDot1XConfigurationsReached fault and shall not add the configuration.

If Identity is used as an anonymous identity for the corresponding authentication method, the device shall ignore an eventually supplied passphrase ID in the same Dot1XStage. Otherwise, if the device cannot process a passphrase ID included in the configuration to be added, the device shall produce a PassphraseID fault and shall not add the configuration.

If the device cannot process a certification path ID included in the configuration to be added, the device shall produce a CertificationPathID fault and shall not add the configuration.

If the device cannot process an authentication method included in the configuration to be added (e.g., unrecognized method or missing configuration parameter), the device shall produce a Dot1XMethod fault and shall not add the configuration.

If the device cannot process the authentication method combination in the configuration to be added, the device shall produce a Dot1XMethodCombination fault and shall not add the configuration.

A device signalling support for IEEE 802.1X configuration with the MaximumNumberOfDot1XConfigurations capability shall support this command.

Table 42: AddDot1XConfiguration command

AddDot1XConfiguration	Access Class: WRITE_SYSTEM
Message name	**Description**
AddDot1XConfigurationRequest	*This message contains a request for the device to add an IEEE 802.1X configuration.* tas:Dot1XConfiguration **Dot1XConfiguration** [1][1] xs:string **Alias** [0][1]
AddDot1XConfigurationResponse	*This message contains the ID of the successfully added IEEE 802.1X configuration.* tt:ReferenceToken **Dot1XID** [1][1]
Fault codes	**Description**
env:Receiver ter:Action ter:MaximumNumberOfDot1X ConfigurationsReached	*The device already has the number of configurations specified by MaximumNumberOfDot1XConfigurations.*
env:Sender ter:InvalidArgVal ter:PassphraseID	*A supplied passphrase ID cannot be processed by the device.*
env:Sender ter:InvalidArgVal ter:CertificationPathID	*A supplied certification path ID cannot be processed by the device.*
env:Sender ter:InvalidArgVal ter: Dot1XMethod	*A supplied IEEE 802.1X authentication method cannot be processed by the device.*
env:Sender ter:InvalidArgVal ter: Dot1XMethodCombination	*The combination of IEEE 802.1X authentication methods cannot be processed by the device.*

5.4.2 Get All IEEE 802.1X Configuration IDs

This operation returns details of all IEEE 802.1X configurations that are on the device. This operation may be used, e.g., if a client lost track of which IEEE 802.1X configurations are present on the device.

If no IEEE 802.1X configurations exist on the device, an empty list is returned.

A device signalling support for IEEE 802.1X configuration with the MaximumNumberOfDot1XConfigurations capability shall support this command.

Table 43: GetAllDot1XConfigurations command

GetAllDot1XConfigurations	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetAllDot1XConfigurationsRequest	*This is an empty message.*
GetAllDot1XConfigurationsResponse	*This message contains a list of all IEEE 802.1X configurations on the device.* tas:Dot1XConfiguration **Configuration** [0][unbounded]
Fault codes	**Description**
	No command-specific fault codes.

5.4.3 Get IEEE 802.1X Configuration Details

This operation returns details of a specific IEEE 802.1X configuration on the device.

If the device cannot process the provided IEEE 802.1X configuration ID, the device shall produce a Dot1XConfigurationID fault.

A device signalling support for IEEE 802.1X configuration with the MaximumNumberOfDot1XConfigurations capability shall support this command.

Table 44: GetDot1XConfiguration command

GetDot1XConfiguration	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetDot1XConfigurationRequest	*This message contains a request for the device to return an existing IEEE 802.1X configuration.* tt:ReferenceToken **Dot1XID** [1][1]
GetDot1XConfigurationResponse	*This message contains the IEEE 802.1X configuration details associated with the requested ID.* tas:Dot1XConfiguration **Dot1XConfiguration** [1][1]
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:Dot1XConfigurationID	*The supplied IEEE 802.1X configuration ID cannot be processed by the device.*

5.4.4 Delete IEEE 802.1X Configuration

This operation deletes an IEEE 802.1X configuration from the device.

If the device cannot process the provided IEEE 802.1X configuration ID, the device shall produce a Dot1XConfigurationID fault.

If a reference exists for the specified IEEE 802.1X configuration, the device shall produce a ReferenceExists fault and shall not delete the configuration.

After an IEEE 802.1X configuration has been successfully deleted, the device may assign its former ID to a new configuration.

A device signalling support for IEEE 802.1X configuration with the MaximumNumberOfDot1XConfigurations capability shall support this command.

Table 45: DeleteDot1XConfiguration command

DeleteDot1XConfiguration	Access Class: UNRECOVERABLE
Message name	**Description**
DeleteDot1XConfigurationRequest	*This message contains a request for the device to delete an IEEE 802.1X configuration.* tt:ReferenceToken **Dot1XID** [1][1]
DeleteDot1XConfigurationResponse	*This is an empty message.*
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:Dot1XConfigurationID	*The supplied IEEE 802.1X configuration ID cannot be processed by the device.*
env:Sender ter:InvalidArgVal ter:ReferenceExists	*A network interface reference exists for the specified IEEE 802.1X configuration.*

5.4.5 Bind IEEE 802.1X Configuration to a Network Interface

This operation binds an IEEE 802.1X configuration to a network interface on the device. This operation shall either create a new binding or replace an existing binding. On failure when an existing binding already exists, the existing binding shall remain.

The Device Management SetNetworkInterface operation provides a method of binding an IEEE 802.1X configuration to an IEEE 802.11 (wireless) interface, and that operation may still be used. But there is no ability for SetNetworkInterface to bind an IEEE 802.1X configuration to a hardwired interface. This operation is provided to bind an IEEE 802.1X configuration to either type of interface.

If SetNetworkInterfaceDot1XConfiguration is used to bind an IEEE 802.1X configuration to an IEEE 802.11 (wireless) interface, then the DeviceManagement GetNetworkInterfaces operation shall return the IEEE 802.1X configuration ID along with the rest of that interface's configuration information.

If the device cannot process the provided network interface token, the device shall produce an InvalidNetworkInterface fault.

If the device cannot process the provided IEEE 802.1X configuration ID, the device shall produce a Dot1XConfigurationID fault.

A device signalling support for IEEE 802.1X configuration with the MaximumNumberOfDot1XConfigurations capability shall support this command.

Table 46: SetNetworkInterfaceDot1XConfiguration command

SetNetworkInterfaceDot1XConfiguration	Access Class: WRITE_SYSTEM
Message name	**Description**
SetNetworkInterfaceDot1XConfiguration Request	*This message contains a request for the device to bind an IEEE 802.1X configuration to a network interface.* tt:ReferenceToken **token** [1][1] tt:ReferenceToken **Dot1XID** [1][1]
SetNetworkInterfaceDot1XConfiguration Response	*An indication if a reboot is needed due to changes in IEEE 802.1X configuration.* xs:boolean **RebootNeeded** [1][1]
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:InvalidNetworkInterface	*The supplied network interface token does not exist.*
env:Sender ter:InvalidArgVal ter:Dot1XConfigurationID	*The supplied IEEE 802.1X configuration ID cannot be processed by the device.*

5.4.6 Get IEEE 802.1X Configuration for a Network Interface

This operation returns the IEEE 802.1X ID and configuration associated with a network interface on the device. If there is no IEEE 802.1X configuration associated with the specified network interface, then the response shall be empty.

If the Device Management SetNetworkInterface operation was used to bind an IEEE 802.1X configuration to an IEEE 802.11 (wireless) interface, then this operation shall return the IEEE 802.1X configuration information as if the SetNetworkInterfaceDot1XConfiguration operation had been used.

If the device cannot process the provided network interface token, the device shall produce an InvalidNetworkInterface fault.

A device signalling support for IEEE 802.1X configuration with the MaximumNumberOfDot1XConfigurations capability shall support this command.

Table 47: GetNetworkInterfaceDot1XConfiguration command

GetNetworkInterfaceDot1XConfiguration	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetNetworkInterfaceDot1XConfiguration Request	*This message contains a request for the device to report the IEEE 802.1X configuration associated with a network interface.* tt:ReferenceToken **token** [1][1]
GetNetworkInterfaceDot1XConfiguration Response	*This message contains the IEEE 802.1X configuration ID. The message is empty if there is no associated configuration.* tt:ReferenceToken **Dot1XID** [0][1]
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:InvalidNetworkInterface	*The supplied network interface token does not exist.*

5.4.7 Unbind IEEE 802.1X Configuration from a Network Interface

This operation unbinds the IEEE 802.1X configuration associated with a network interface on the device. If there is no IEEE 802.1X configuration associated with the specified network interface, then the operation does nothing.

The Device Management SetNetworkInterface operation provides a method of unbinding an IEEE 802.1X configuration from an IEEE 802.11 (wireless) interface by omitting the configuration ID, and that operation may still be used. But there is no ability for SetNetworkInterface to unbind an IEEE 802.1X configuration from a hardwired interface. This operation is provided to unbind an IEEE 802.1X configuration from either type of interface.

If the Device Management SetNetworkInterface operation was used to bind an IEEE 802.1X configuration to an IEEE 802.11 (wireless) interface, then this operation shall unbind the IEEE 802.1X configuration information as if the SetNetworkInterfaceDot1XConfiguration operation had been used.

If the device cannot process the provided network interface token, the device shall produce an InvalidNetworkInterface fault.

A device signalling support for IEEE 802.1X configuration with the MaximumNumberOfDot1XConfigurations capability shall support this command.

Table 48: DeleteNetworkInterfaceDot1XConfiguration command

DeleteNetworkInterfaceDot1XConfiguration	Access Class: WRITE_SYSTEM
Message name	**Description**
DeleteNetworkInterfaceDot1XConfiguration Request	*This message contains a request for the device to unbind the IEEE 802.1X configuration associated with a network interface.* tt:ReferenceToken **token** [1][1]
DeleteNetworkInterfaceDot1XConfiguration Response	*An indication if a reboot is needed due to changes in IEEE 802.1X configuration.* xs:boolean **RebootNeeded** [1][1]
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:InvalidNetworkInterface	*The supplied network interface token does not exist.*

5.5 Capabilities

5.5.1 Advanced Security Service Capabilities

The capabilities reflect optional functions and functionality of the different features in the advanced security service. The service capabilities consist of keystore capabilities and TLS server capabilities. The information is static and does not change during device operation.

A device shall support this command.

Table 49: GetServiceCapabilitites command

GetServiceCapabilities	Access Class: PRE_AUTH
Message name	**Description**
GetServiceCapabilitiesRequest	*This is an empty message.*
GetServiceCapabilitiesResponse	*The capability response message contains the requested service capabilities using a hierarchical XML capability structure.* tas:Capabilities **Capabilities** [1][1]
Fault codes	**Description**
	No command-specific fault codes.

5.5.2 Keystore Capabilities

The keystore capabilities reflect optional functions and functionality of the keystore on a device. The following capabilites are available:

Table 50: Keystore Capabilities

Capability Name	**Capability Semantics**
MaximumNumberOfPassphrases	Indicates the maximum number of passphrases that the device is able to store simultaneously.
MaximumNumberOfKeys	Indicates the maximum number of keys that the device is able store simultaneously.
MaximumNumberOfCertificates	Indicates the maximum number of certificates that the device is able to store simultaneously.
MaximumNumberOfCertificationPaths	Indicates the maximum number of certificate paths that the device is able to store simultaneously.
RSAKeyPairGeneration	Indicates support for on-board RSA key pair generation.
RSAKeyLengths	Indicates which RSA key lengths are supported by the device.
PKCS8RSAKeyPairUpload	Indicates support for uploading an RSA key pair in a PKCS#8 data structure.
PKCS12CertificateWithRSAPrivateKeyUpload	Indicates support for uploading a certificate along with an RSA private key in a PKCS#12 data structure.

PKCS10ExternalCertificationWithRSA	Indicates support for creating PKCS#10 requests for RSA keys and uploading the certificate obtained from a CA.
SelfSignedCertificateCreationWithRSA	Indicates support for creating self-signed certificates for RSA keys.
SignatureAlgorithms	Indicates which signature algorithms are supported by the device.
PasswordBasedEncryptionAlgorithms	Indicates which password-based encryption algorithms are supported by the device.
PasswordBasedMACAlgorithms	Indicates which password-based MAC algorithms are supported by the device.
X.509Versions	Indicates which X.509 versions are supported by the device. [2] X.509 versions shall be encoded as version numbers, e.g., 1, 2, 3.
MaximumNumberOfCRLs	Indicates the maximum number of CRLs that the device is able to store simultaneously.
MaximumNumberOfCertificationPathValidationPolicies	Indicates the maximum number of certification path validation policies that the device is able to store simultaneously.
EnforceTLSWebClientAuthExtKeyUsage	Indicates whether a device supports checking for the TLS WWW client auth extended key usage extension while validating certification paths.

5.5.3 TLS Server Capabilities

The TLS server capabilities reflect optional functions and functionality of the TLS server. The information is static and does not change during device operation. The following capabilites are available:

Table 51: TLS Server Capabilities

TLSServerSupported	Indicates which TLS server versions are supported by the device. Server versions shall be encoded as version numbers, e.g., 1.0, 1.1., 1.2.

[2] If a device supports X.509v3 certificates, this fact shall also be signalled by this capability.

MaximumNumberOfTLSCertificationPaths	Indicates the maximum number of certification paths that may be assigned to the TLS server simultaneously.
TLSClientAuthSupported	Indicates whether the device supports TLS client authentication as defined in this specification.
MaximumNumberOfTLSCertificationPathValidationPolicies	Indicates the maximum number of certification path validation policies that may be assigned to the TLS server simultaneously

5.5.4 IEEE 802.1X Capabilities

The IEEE 802.1X configuration capabilities reflect optional functions and functionality of IEEE 802.1X configuration on a device. The following additional capabilites are defined:

Table 52: Additional IEEE 802.1X Configuration Capabilities

Capability Name	Capability Semantics
MaximumNumberOfDot1XConfigurations	Indicates the maximum number of IEEE 802.1X configurations that the device is able to configure simultaneously.
Dot1XMethods	A list of authentication method outer/inner (phase1/phase2) combinations supported by the device.

5.5.5 Capability-implied Requirements

Table 53 summarizes for each capability the minimum requirements that a device signaling this capability shall satisfy; it should not be seen as a recommendation.

Table 53: Requirements implied by Capabilities

Capability	Implied Requirements
MaximumNumberOfPassphrases	If greater than zero, the following commands shall be supported: • UploadPassphrase • GetAllPassphrases • DeletePassphrase If greater than zero, the device shall support passphrases that consist of characters from the ASCII character set and that have a length of up to 40 characters.
MaximumNumberOfKeys	If greater than zero, then the following commands shall be supported:

	• GetKeyStatus • GetAllKeys • DeleteKey
MaximumNumberOfCertificates	If greater than zero, then MaximumNumberOfKeys>0 shall hold.
MaximumNumberOfCertificationPaths	If greater than zero, MaximumNumberOfCertificates>=2 shall hold.
RSAKeyPairGeneration	If true, the following commands shall be supported: • CreateRSAKeyPair If true, the list of supported RSA key lengths as indicated by the RSAKeyLenghts capability shall not be empty. If true, MaximumNumberOfKeys>0 shall hold.
PKCS8RSAKeyPairUpload	If true, the following commands shall be supported: • UploadKeyPairInPKCS8 If true, MaximumNumberOfPassphrases >0 shall hold. If true, MaximumNumberOfKeys > 0 shall hold. If true, the list of supported RSA key lengths as indicated by the RSAKeyLenghts capability shall not be empty. If true, the list of supported password-based encryption algorithms as indicated by the PasswordBasedEncryptionAlgorithms capability shall contain at least the algorithm pbeWithSHAAnd3-KeyTripleDES-CBC.
PKCS12CertificateWithRSAPrivateKeyUpload	If true, the following commands shall be supported: • UploadCertificateWithPrivateKeyInPKCS12 • GetCertificate • GetAllCertificates • DeleteCertificate • GetCertificationPath • GetAllCertificationPaths • DeleteCertificationPath If true, MaximumNumberOfPassphrases >0 shall hold. If true, MaximumNumberOfKeys >=2 shall hold. If true, MaximumNumberOfCertificates >=2 shall hold.

	If true, MaximumNumberOfCertificattionPaths >0 shall hold. If true, the list of supported RSA key lengths as indicated by the RSAKeyLenghts capability shall not be empty. If true, the list of supported password-based encryption algorithms as indicated by the PasswordBasedEncryptionAlgorithms capability shall contain at least the algorithm pbeWithSHAAnd3-KeyTripleDES-CBC. If true, the list of supported password-based MAC algorithms as indicated by the PasswordBasedMACAlgorithms capability shall contain at least the algorithm hmacWithSHA256. If true, the list of supported X.509 versions as indicated by the X.509Versions capability shall contain at least the value 3. If true, the list of supported signature algorithms as indicated by the SignatureAlgorithms capability shall contain at least the algorithms sha1-WithRSAEncryption and sha256WithRSAEncryption.
PKCS10ExternalCertificationWithRSA	If true, the following operations shall be supported: • RSA key pair generation as signaled by the RSAKeyPairGeneration capability or RSA key pair upload as signaled by the PKCS8RSAKeyPairUpload capability or RSA key pair upload as signaled by the PKCS12CertificateWithRSAPrivateKeyUpload capability • Creating a CSR with the CreatePKCS10CSR command. • GetCertificate • GetAllCertificates • DeleteCertificate • Uploading the certificate created for the CSR as well as the certificate of the created certificate's signer with the UploadCertificate command. If true, MaximumNumberOfCertificates>=2 and MaximumNumberOfCertificationPaths>0 shall hold. If true, MaximumNumberOfKeys>=2 shall hold. If true, the list of supported signature algorithms as indicated by the SignatureAlgorithms capability shall contain at least the algorithms sha1-WithRSAEncryption and sha256WithRSAEncryption.
SelfSignedCertificateCreationWithRSA	If true, the following commands shall be supported: • CreateSelfSignedCertificate • GetCertificate • GetAllCertificates • DeleteCertificate

	If true, the following operations shall be supported: • RSA key pair generation as signaled by the RSAKeyPairGeneration capability or RSA key pair upload as signaled by the PKCS8RSAKeyPairUpload capability or RSA key pair upload as signaled by the PKCS12CertificateWithRSAPrivateKeyUpload capability If true, MaximumNumberOfCertificates> 0 shall hold. If true, the list of supported signature algorithms as indicated by the SignatureAlgorithms capability shall contain at least the algorithms sha1-WithRSAEncryption and sha256WithRSAEncryption.
TLSServerSupported	If not empty, the value 1.0 shall be contained in the list of supported TLS versions. If not empty, PKCS10ExternalCertificationWithRSA shall be true or SelfSignedCertificateCreationWithRSA shall be true. If not empty, the following commands shall be supported: • CreateCertificationPath • GetCertificationPath • GetAllCertificationPaths • DeleteCertificationPath • AddServerCertificateAssignment • RemoveServerCertificateAssignment • ReplaceServerCertificateAssignment • GetAssignedServerCertificates If not empty, MaximumNumberOfCertificationPaths>=2 and MaximumNumberOfTLSCertificationPaths>0 shall hold.
TLSServerSupported and PKCS10ExternalCertificationWithRSA	If TLSServerSupported is non-empty and PKCS10ExternalCertificationWithRSA is true, MaximumNumberOfCertificates>=3 shall hold.
MaximumNumberOfTLSCertificationPaths	If greater than zero, MaximumNumberOfCertificationPaths>0 shall hold.
X.509Versions	If X.509v3 is supported, the device shall support the distinguished name attribute types *country*, *organization*, *organizational unit*, *distinguished name qualifier*, *state or province name*, *common name*, and *serial number*.
MaximumNumberOfCRLs	If greater than zero, then the following commands shall be supported • UploadCRL • GetCRL

	• GetAllCRLs • DeleteCRL
MaximumNumberOfCertificationPathValidationPolicies	If greater than zero, then the following command shall be supported • CreateCertPathValidationPolicy • GetCertPathValidationPolicy • GetAllCertPathValidationPolicies • DeleteCertPathValidationPolicy If greater than zero, PKCS12CertificateWithRSAPrivateKeyUpload, PKCS10ExternalCertificationWithRSA or SelfSignedCertificateCreationWithRSA shall be true.
TLSClientAuthSupported	If true, the following commands shall be supported • SetClientAuthenticationRequired • GetClientAuthenticationRequired • AddCertPathValidationPolicyAssignment • RemoveCertPathValidationPolicyAssignment • ReplaceCertPathValidationPolicyAssignment • GetAssignedCertPathValidationPolicies If true, TLSServerSupported shall not be empty. If true, MaximumNumberOfCertificationPathValidationPolicies>=2 and MaximumNumberOfTLSCertificationPathValidationPolicies>0 shall hold. If true, MaximumNumberOfCRLs>0 shall hold. If true, the device shall support • validating certification paths containing X.509v3 certificates that are signed with signatures of type sha1-WithRSAEncryption or with sha256WithRSAEncrpytion • processing X.509 CRLs that are compliant to the CRL profile defined in [RFC 5280], Sect. 5 and that o are signed with signatures of type sha1-WithRSAEncryption or sha256WithRSAEncryption and o are complete direct CRLs as defined in [RFC 5280] that that are signed with the same signature key as the signature key that the CA uses to sign issued certificates
MaximumNumberOfTLSCertificationPathValidationPolici	If greater than zero, MaximumNumberOfCertificationPathValidationPolicies >0 shall

es	hold.
MaximumNumberOfDot1XConfigurations	If greater than zero, the following commands shall be supported: • AddDot1XConfiguration • GetDot1XConfigurations • GetDot1XConfiguration • DeleteDot1XConfiguration • SetNetworkInterfaceDot1XConfiguration • GetNetworkInterfaceDot1XConfiguration • DeleteNetworkInterfaceDot1XConfiguration If greater than zero, the Dot1XMethods capability shall be present and shall contain at least the EAP-PEAP MSCHAPv2 method.
Dot1XMethods	If not empty, shall contain at least the “EAP-PEAP/MSCHAPv2” method, and MaximumNumberOfDot1XConfigurations shall be greater than zero.

5.6 Events

5.6.1 Key Status

A device that indicates support for key handling via the MaximumNumberOfKeys capability shall provide information about key status changes through key status events.

A device shall include an OldStatus value unless NewStatus is generating.

```
Topic: tns1:Advancedsecurity/Keystore/KeyStatus
<tt:MessageDescription>
      <tt:Source>
            <tt:SimpleItemDescription Name="KeyID" Type="tas:KeyID"/>
      </tt:Source>
      <tt:Data>
            <tt:SimpleItemDescriptionminOccurs=”0” Name="OldStatus"
            Type="tas:KeyStatus"/>
            <tt:SimpleItemDescription Name="NewStatus"
            Type="tas:KeyStatus"/>
      </tt:Data>
</tt:MessageDescription>
```

5.7 Service specific data types

The service specific data types are defined in advancedsecurity.wsdl.

5.8 Service specific fault codes

The table below lists the advanced security service specific fault codes. Additionally, each command can also generate a generic fault as defined in the [ONVIF Core specification].

Table 54: Advanced security service specific fault codes

Fault Code	Parent Subcode Subcode	Fault Reason	Description
env:Sender	ter:InvalidArgVal ter:KeyID	KeyID not appropriate	No key is stored under the requested KeyID.
env:Sender	ter:InvalidArgVal ter:InvalidKeyType	Key type invalid	The key stored in the keystore under the requested KeyID is of an invalid type.
env:Receiver	ter:Action ter:KeyDeletionFailed	Deletion of a key failed.	Deleting the key with the requested KeyID failed.
env:Receiver	ter:Action ter:CSRCreationFailed	Failure to create a CSR	The generation of the PKCS#10 certification request failed.
env:Sender	ter:InvalidArgVal ter:UnsupportedSignatureAlgorithm	Signature algorithm not supported	The specified signature algorithm is not supported by the device.
env:Sender	ter:InvalidArgVal ter:KeySignatureAlgorithmMismatch	Mismatch of key and signature algorithm	The specified public key is an invalid input to the specified signature algorithm.
env:Sender	ter:InvalidArgVal ter:InvalidKeyStatus	KeyStatus invalid	The key with the requested KeyID has an inappropriate status.
env:Sender	ter:InvalidArgVal ter:InvalidSubject	Subject invalid	The specified subject is invalid or incomplete.
env:Sender	ter:InvalidArgVal ter:InvalidAttribute	Attribute invalid	The specified attribute is invalid or incomplete.
env:Sender	ter:InvalidArgVal ter:InvalidDateTime	dateTime invalid	The specified dateTime is invalid.
env:Receiver	ter:Action ter:CertificateCreationFailed	Certificate creation failed.	The generation of a certificate failed.
env:Receiver	ter:Action ter:MaximumNumberOfCertificatesReached	Maximum number of certificates reached	The device does not have enough storage space to store the certificate to be created.
ter:Sender	ter:InvalidArgVal ter:UnsupportedX509Version	X509 version not supported	The specified X.509 version is not supported by the device.
env:Sender	ter:InvalidArgVal ter:X509VersionExtensionsMismatch	Extensions not supported	The request contains extensions that are not supported by the X.509 version specified in the request.
env:Receiver	ter:Action ter: MaximumNumberOfKeysReached	Maximum number of keys reached	The keystore does not have enough storage space to store the key pair that has to be generated.
env:Sender	ter:InvalidArgVal ter:BadCertificate	Certificate bad	The supplied certificate cannot be processed by the device.

env:Sender	ter:InvalidArgVal ter:UnsupportedPublicKeyAlgorithm	Public key algorithm not supported	The public key algorithm of the public key in the certificate is not supported by the device.
env:Receiver	ter:Action ter:NoMatchingPrivateKey	Matching private key not found.	The keystore does not contain a key pair with a private key that matches the public key in the uploaded certificate.
env:Sender	ter:InvalidArgVal ter:CertificateID	CertificateID not appropriate	No certificate is stored under the requested CertificateID.
env:Receiver	ter:Action ter:CertificateDeletionFailed	Deletion of a certificate failed.	Deleting the certificate with the requested CertificateID failed.
env:Sender	ter:InvalidArgVal ter:ReferenceExists	ReferenceExists	A reference exists for the object that is to be deleted.
env:Sender	ter:InvalidArgVal ter:InvalidCertificationPath	CertificationPath invalid	At least one certificate in the certification path is not correctly signed with the public key in the next certificate in the path.
env:Receiver	ter:Action ter:CertificationPathCreationFailed	Certification path creation failed.	Creating the certification path failed.
env:Sender	ter:InvalidArgVal ter:CertificationPathID	Certification Path ID invalid	No certification path is stored under the requested certification path ID.
env:Receiver	ter:Action ter:CertificationPathDeletionFailed	Certification path deletion failed	Deleting the certification path with the requested certification path ID failed.
env:Sender	ter:InvalidArgVal ter:NoPrivateKey	The key pair does not contain a private key.	The key pair that is associated with the first certificate in the certificate chain does not have an associated private key.
env:Receiver	ter:Action ter:MaximumNumberOfCertificationPathsReached	Maximum number of certification paths received.	The maximum number of certification paths that may be assigned to the TLS server simultaneously is reached.
env:Sender	ter:InvalidArgVal ter:OldCertificationPathID	Invalid old certification path ID	No certification path under the given old certification path ID is associated with the TLS server.
env:Sender	ter:InvalidArgVal ter:NewCertificationPathID	Invalid new certification path ID	No certification path is stored in the keystore under the given certification path ID.
env:Receiver	ter:Action ter: MaximumNumberOfTLSCertificationPathsReached	Maximum number of TLS certification paths reached	The maximum number of certification paths that may be assigned to the TLS server simultaneously is reached.
env:Sender	ter:InvalidArgVal ter:PassphraseID	PassphraseID not appropriate	No passphrase is stored under the requested PassphraseID.
env:Sender	ter:InvalidArgVal	Invalid PKCS#8 File	The PKCS#8 data structure

	ter:BadPKCS8File		cannot be processed by the device.
env:Sender	ter:InvalidArgVal ter:DecryptionFailed	Decryption failed	The given data could not be decrypted.
env:Sender	ter:InvalidArgVal ter:PublicPrivateKeyMismatch	Public and private key do not match	The supplied private key does not match the supplied public key.
env:Sender	ter:InvalidArgVal ter:BadPKCS12File	Invalid PKCS#12 File	The PKCS#12 data structure cannot be processed by the device.
env:Receiver	ter:Action ter:KeyLength	Key length not supported.	The specified key length is not supported by the device.
env:Receiver	ter:Action ter:MaximumNumberOfCRLs Reached	Maximum number of CRLs reached	The device does not have enough storage space to store the CRL to be uploaded.
env:Sender	ter:InvalidArgVal ter:BadCRL	CRL bad	The supplied CRL cannot be processed by the device.
env:Sender	ter:InvalidArgVal ter:CRLID	CRL ID not appropriate	No CRL is stored under the requested CRL ID.
env:Receiver	ter:Action ter:MaximumNumberOfCertP athValidationPoliciesReached	Maximum number of certification path validation policies reached	The device does not have enough storage space to store the certification path validation policy to be created.
env:Sender	ter:InvalidArgVal ter:CertPathValidationParame ters	Bad certification path validation parameters	The specified certification path validation parameters are invalid.
env:Sender	ter:InvalidArgVal ter:CertPathValidationPolicyID	Certification path validation policy ID not appropriate	No certification path validation policy is stored under the requested certification path validation policy ID.
env:Receiver	ter:ActionNotSupported ter:EnablingClientAuthenticati onFailed	Enabling client authentication failed.	The device does not support TLS client authentication, or TLS client authentication is not configured appropriately.
env:Receiver	ter:Action ter:MaxinumNumberOfTLSCe rtPathValidationPoliciesReach ed	Maximum number of TLS certification path validation policies reached.	The maximum number of certification path validation policies that may be assigned to the TLS server simultaneously is reached.
env:Sender	ter:InvalidArgVal ter:OldCertPathValidationPoli cyID	Invalid old certification path validation policy ID	No certification path validation policy under the given OldCertPathValidationPolicyI D is associated with the TLS server.
env:Sender	ter:InvalidArgVal	Invalid new	No certification path validation

	ter:NewCertPathValidationPolicyID	certification path validation policy ID	policy under the given NewCertPathValidationPolicyID is stored in the device's keystore.
env:Receiver	ter:Action ter:MaximumNumberOfDot1XConfigurationsReached	Maximum configurations reached	The device already has the number of configurations specified by MaximumNumberOfDot1XConfigurations..
env:Sender	ter:InvalidArgVal ter:Dot1XMethod	Invalid method	A supplied IEEE 802.1X authentication method cannot be processed by the device.
env:Sender	ter:InvalidArgVal ter: Dot1XMethodCombination	Invalid combination of methods	The combination of IEEE 802.1X authentication methods cannot be processed by the device.
env:Sender	ter:InvalidArgVal ter:Dot1XConfigurationID	Invalid configuration ID	The supplied IEEE 802.1X configuration ID cannot be processed by the device.

6 Security Considerations

This section is informative.

- Faults and their types shall not disclose sensitive information to an attacker that he could not obtain otherwise.

- For interoperability reasons, sha1WithRSAEncryption as specified in [RFC3279] is mandated as default signature algorithm. However, since the security of the SHA-1 algorithm is under question, this specification mandates that a signature algorithm based on SHA-256, particularly sha256WithRSAEncryption as specified in [RFC 4055], be supported in addition.

- Operations with arguments that need protection against eavesdropping or manipulation shall only be executed over sufficiently protected communication channels.

- It is good practice not to use the same key for different purposes. In order to prevent the device from using the same key for different purposes unnoticedly, this specification mandates that all keys in the keystore be distinct.

- Private keys must be protected against disclosure to unauthorized parties. If a private key is uploaded in an encrypted PKCS#8 or PKCS#12 structure, the passphrase that is used to encrypt the structure must be uploaded to the device over a communication channel that is protected against eavesdropping in order to preserve the confidentiality of the private key. Moreover, the confidentiality of the uploaded private key depends on the strength of the encryption passphrase. It is therefore strongly recommended to use random passwords with sufficient length.

- In general, externally generated keys must be regarded less trustworthy than keys that are generated by the device because the probability of being disclosed to an attacker is higher for an externally generated key than for an internally generated key. A client may determine whether a key was generated by the device from the *externallyGenerated* attribute of the key.

- While new specifications should be based on [PKCS#5 v2.0] or higher, adoption of this standard is still limited. Therefore, this specification intends to balance security and interoperability by mandating cryptographic algorithms based on [PKCS#5 v1.5] as interoperability baseline while strongly encouraging the use of [PKCS#5 v2.0] or higher. Future versions of this specification or specifications referring to this specification may mandate additional cryptographic algorithms.

- Although PKCS#8 [RFC 5208] is widely used for exchanging cryptographic keys, this specification is based on the successor standard [RFC 5958], particularly in order to incorporate both private key and public key in the same data structure.

- The default certification path validation policy is designed as a permissive interoperability baseline based on the certification path validation algorithm defined in [RFC 5280].

- CRLs can be expected to be available from virtually any CA as a source of revocation information. The benefit of OCSP [RFC 6960] as a means to obtain revocation information is increasingly under question, since a man-in-the-middle attacker blocking OCSP traffic combined with a permissive validator that silently accepts certificates for which no revocation is available limits the effective security gain of using OCSP. Therefore, this specification mandates support for CRLs as interoperability baseline and leaves other revocation information sources to future versions.

- Devices may be required to use different trust anchors for different security features. Therefore, trust in a certificate is indicated as part of a certification path validation policy rather than globally, e.g., with an attribute of the X509Certificate data type.

- [RFC 5280], Sect. 6.1.4 (k) mandates that every certificate in a certification path except for the end entity certificate must be verified to be a CA certificate. For X.509 version 3 certificates, this is verified through the CA attribute in the basic constraints extension. For X.509 version 1 and 2 certificates, this information must be supplied by out-of-band mechanisms. Within the scope of this specification, the only means to obtain this information is the trust anchor information contained in the certification path validation algorithm. Therefore, the default certification path validation policy mandates that the certification path validation algorithm defined in Sect. 4.2.4 consider all certificates that are not marked as trust anchor as non-CA certificates.

- When configuring IEEE 802.1X, it is usually necessary to upload a password to the device's keystore. This should be done either on a private network (e.g., using a direct network connection between a laptop and a device) or using TLS (SSL) on the device to encrypt client / device traffic.

7 Design Rationale

This section is informative.

7.1 General Design Goals

The Advanced Security Service is designed for modularity and extensibility. Therefore, each security feature is encapsulated in a separate port type within the service. Later revisions of this specification may add port types to enhance the Advanced Security Service by additional security features.

Within a security feature, capabilities indicate support for sub-features and configuration options. Later revisions of this specification may add additional sub-features to existing features and identify them by additional capabilities.

Port types and capabilities enable devices to support well-defined subsets of this specification and to communicate this information to clients effectively.

7.2 Keystore

The keystore design assumes that passphrases are chosen by clients. Therefore, an operation for retrieving passphrases from a device is deliberately omitted. If client loses a previously uploaded passphrase, the client should create a new passphrase, upload the new passphrase to the device, and delete the old passphrase from the device.

This specification deliberately deviates from the terminology in PKCS#8 and PKCS#12 by using the term 'passphrase' instead of 'password' in order to avoid confusion with the

password that is assigned to ONVIF device users and the corresponding API in the ONVIF Device Management Service.

The keystore design is based on the rationale that an RSA key pair is a special type of key pair and a key pair is a special type of key. Therefore, key-related operations in the keystore deliberately refer to the most generic possible type in this hierarchy. For example, the DeleteKey operation (see Sect. 5.2.6.2.6) refers to a key instead of a key pair or even an RSAKeyPair because it is applicable to all keys. On the other hand, the GetPrivateKeyStatus command refers to a key pair instead of a key, since this command is not meaningful for a key that is not a key pair, e.g., a symmetric key.

While this revision of the keystore specification only supports RSA key pairs as key pairs, later revisions of this specification may add other types of key pairs or symmetric keys as special types of keys.

Some interactions with the keystore, e.g., retrieving the private key for a public key that is contained in a certificate, are required device-internally, but need not be accessible to clients and may even, as in the above example, imply a security risk when made available outside the device. Such operations are therefore deliberately omitted from this specification.

The certificate-based client authentication specification intends to balance security concerns, interoperability, and implementation effort in order to facilitate adoption. Therefore, the certification path validation algorithm defined in [RFC 5280] serves as interoperability baseline. The parameter values in the default certification path validation policy have been selected such that widely used implementations of the certification path validation algorithm can be used in their default configurations as much as permitted by the objective to provide an acceptable security baseline.

At the same time, more fine grained customization of the default certification path validation behavior in future versions of this specification is enabled by an extensible CertValidationPolicyParameters data type and capabilities that indicate which configuration options a device supports. As an example, checking for the TLS WWW client authentication key usage extension in client certificates is included in this specification, which can be implemented with moderate effort on the device side (e.g., with the OpenSSL option –purpose sslclient). Customization options for other parameters of the certification path validation algorithm are deliberately left to future versions of this specification in order to limit the required initial implementation effort.

In order to facilitate future extensions of this specification, the number of certification path validation policies that may be assigned to the TLS server simultaneously is not limited, but the certification path validation behavior is unspecified if more than one policy is assigned to the TLS server at the same time. Therefore, devices implementing this specification should limit the number of simultaneously assigned policies to one.

7.3 TLS Server

This revision of the Advanced Security Service Specification allows to manage assignments of certification paths to the TLS server on a device. It is permitted that a TLS server presents different certification paths to different clients, therefore more than one certification path may be assigned simultaneously to the TLS server to use as a server certificate.

All other configuration of the TLS server on a device is outside the scope of this specification revision and may be addressed by later revisions of this document.

Annex A. Revision History

Rev.	Date	Editor	Changes
1.0	Aug - 2013	Dirk Stegemann	Initial version
1.0.1	Dec - 2013	Michio Hirai, Dirk Stegemann	Change Request 1219, 1220 1222, 1267, 1271, 1272, 1277
1.0.2	June - 2014	Dirk Stegemann, Stefan Andersson	Change Request 1268, 1276, 1349, 1350, 1351, 1352, 1376, 1377, 1378, 1379, 1380, 1381, 1382, 1390
1.1	Dec - 2014	Dirk Stegemann	Change Request 1528, 1529, 1530, 1531, 1532, 1533, 1534, 1535, 1536, 1543, 1554
1.2	Jun - 2015	Dirk Stegemann	Change Request 1552, 1555, 1565, 1580, 1583, 1590, 1615, 1616, 1617, 1618, 1619 Added certificate-based client authentication
1.3	Feb-2016	Stefan Andersson Steve Wolf	Added IEEE 802.1X configuration