

ONVIF™ Advanced Security Service Specification

Version 1.1
December 2014



© 2008-2014 by ONVIF: Open Network Video Interface Forum Inc.. All rights reserved.

Recipients of this document may copy, distribute, publish, or display this document so long as this copyright notice, license and disclaimer are retained with all copies of the document. No license is granted to modify this document.

THIS DOCUMENT IS PROVIDED "AS IS," AND THE CORPORATION AND ITS MEMBERS AND THEIR AFFILIATES, MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THIS DOCUMENT ARE SUITABLE FOR ANY PURPOSE; OR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

IN NO EVENT WILL THE CORPORATION OR ITS MEMBERS OR THEIR AFFILIATES BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE OR CONSEQUENTIAL DAMAGES, ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THIS DOCUMENT, WHETHER OR NOT (1) THE CORPORATION, MEMBERS OR THEIR AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR (2) SUCH DAMAGES WERE REASONABLY FORESEEABLE, AND ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THIS DOCUMENT. THE FOREGOING DISCLAIMER AND LIMITATION ON LIABILITY DO NOT APPLY TO, INVALIDATE, OR LIMIT REPRESENTATIONS AND WARRANTIES MADE BY THE MEMBERS AND THEIR RESPECTIVE AFFILIATES TO THE CORPORATION AND OTHER MEMBERS IN CERTAIN WRITTEN POLICIES OF THE CORPORATION.

CONTENTS

- 1 Scope 4**
- 2 Normative references 4**
- 3 Terms and Definitions 6**
 - 3.1 Definitions.....6
 - 3.2 Abbreviations6
 - 3.3 Namespace6
- 4 Overview 8**
- 5 Advanced Security Service 8**
 - 5.1 General Structure.....8
 - 5.2 Keystore8
 - 5.2.1 Elements of the Keystore.....8
 - 5.2.2 Unique Identifiers9
 - 5.2.3 Uniqueness of Objects in the Keystore9
 - 5.2.4 Referential Integrity.....9
 - 5.2.5 Key Status.....10
 - 5.2.6 Keystore Operations10
 - 5.3 TLS Server31
 - 5.3.1 Elements of the TLS Server.....31
 - 5.3.2 TLS Server Operations31
 - 5.4 Capabilities.....35
 - 5.4.1 Advanced Security Service Capabilities35
 - 5.4.2 Keystore Capabilities35
 - 5.4.3 TLS Server Capabilities36
 - 5.4.4 Capability-implied Requirements37
 - 5.5 Events40
 - 5.5.1 Key Status.....40
 - 5.6 Service specific data types.....40
 - 5.7 Service specific fault codes.....41
- 6 Security Considerations 43**
- 7 Design Rationale 44**
 - 7.1 General Design Goals.....44
 - 7.2 Keystore44
 - 7.3 TLS Server44
- Annex A. Revision History 45**

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

ONVIF Core Specification

<<http://www.onvif.org/specs/core/ONVIF-Core-Specification-v220.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 Request Syntax Specification 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 Cryptography 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.
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.
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.
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.

3.2 Abbreviations

CA	Certification Authority
CSR	Certificate Signing Request (also called Certification Request)
MAC	Message Authentication Code
ONVIF	Open Network Video Interface Forum
SHA	Secure Hashing Algorithm
TLS	Transport Layer Security

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/wsd
ter	http://www.onvif.org/ver10/error

tt <http://www.onvif.org/ver10/schema>
xs <http://www.w3.org/2001/XMLSchema>

4 Overview

This specification covers the following advanced security features:

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

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/wsd/advancedsecurity.wsd>.

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

5 Advanced Security Service

5.1 General Structure

This section covers the security features

- Keystore
- TLS server

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

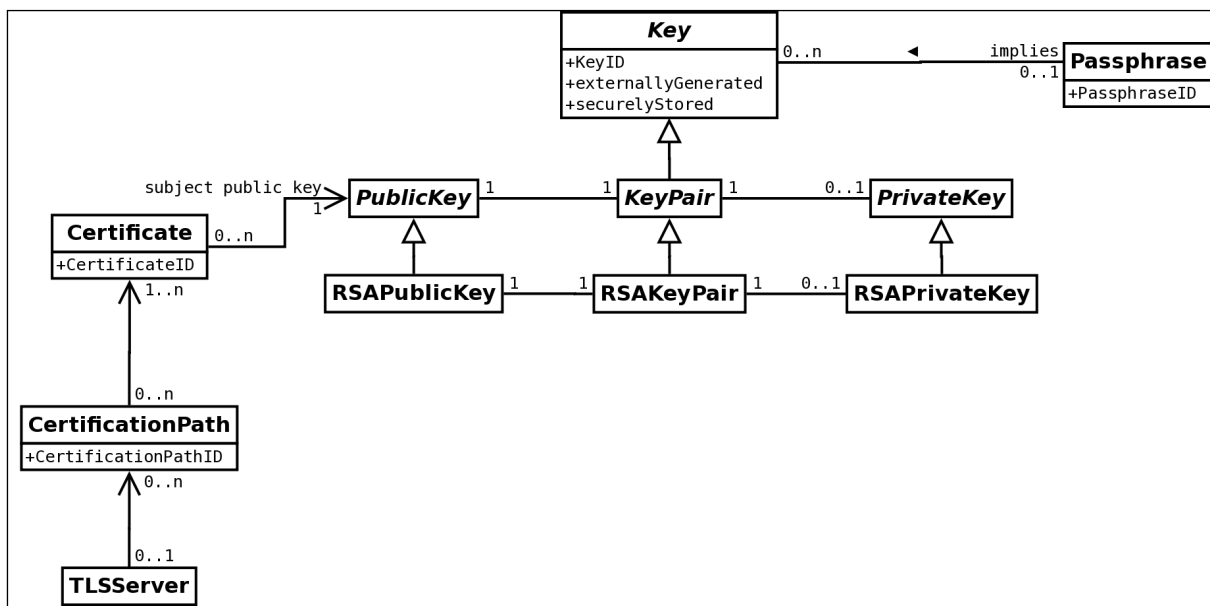


Figure 1 ONVIF Advanced Security Service [UML] Class Diagram

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, and certification paths.

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, and certification paths in the keystore shall be uniquely identified by certification path 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, and certification paths 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 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
- Certificates 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 or a security feature.
- A certification path 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 and certification paths. 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 or 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 1: UploadPassphrase command

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

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 2: GetAllPassphrases command

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

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 3: DeletePassphrase command

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

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.5.2.6.1.2) whether the generation has finished. The client may also subscribe to Key Status events (see Sect. 5.5.1) 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.

¹Implementors 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.

Table 4:CreateRSAKeyPair command

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

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 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 5: UploadKeyPairInPKCS8 command

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

env:Sender ter:InvalidArgVal ter:PublicPrivateKeyMismatch	<i>The supplied private key does not match the supplied public key.</i>
---	---

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 6: GetKeyStatus command

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

5.2.6.2.4 Get Private Key Status

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.

Table 7: GetPrivateKeyStatus command

GetPrivateKeyStatus		Access Class: READ_SYSTEM_SECRET
Message name	Description	
GetPrivateKeyStatusRequest	<i>This message contains a request for the device to return whether a key pair contains a private key.</i> tas:KeyID KeyID [1][1]	
GetPrivateKeyStatusResponse	<i>This message contains the status for the requested KeyID.</i> xs:boolean hasPrivateKey [1][1]	
Fault codes	Description	
env:Sender ter:InvalidArgVal	<i>No key is stored under the requested KeyID.</i>	

ter:KeyID	
env:Sender ter:InvalidArgVal ter:InvalidKeyType	<i>The key stored under the requested KeyID does not identify a key pair.</i>

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 8: GetAllKeys command

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

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 9: DeleteKey command

DeleteKey		Access Class: UNRECOVERABLE
Message name	Description	
DeleteKeyRequest	<i>This message contains a request for the device to delete a key from the keystore.</i> tas:KeyID KeyID [1][1]	
DeleteKeyResponse	<i>This is an empty message.</i>	

Fault codes	Description
env:Receiver ter:Action ter:KeyDeletionFailed	<i>Deleting the key with the requested KeyID failed.</i>
env:Sender ter:InvalidArgVal ter:KeyID	<i>No key is stored under the requested KeyID.</i>
env:Sender ter:InvalidArgVal ter:ReferenceExists	<i>A reference exists for the specified key.</i>

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. A device that supports this command shall as minimum support the sha1WithRSAEncryption signature algorithm as specified in [RFC 3279] and the sha256WithRSAEncryption signature algorithm as specified in [RFC 4055]. 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 10: CreatePKCS10CSR command

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

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. A device that supports this command shall as minimum support the sha1WithRSAEncryption signature algorithm as specified in [RFC 3279] and the sha256WithRSAEncryption signature algorithm as specified in [RFC 4055].

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 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.

[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 11: CreateSelfSignedCertificate command

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

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. A device that supports this command shall as minimum support the sha1WithRSAEncryption signature algorithm as specified in [RFC 3279] and the sha256WithRSAEncryption signature algorithm as specified in [RFC 4055].

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.

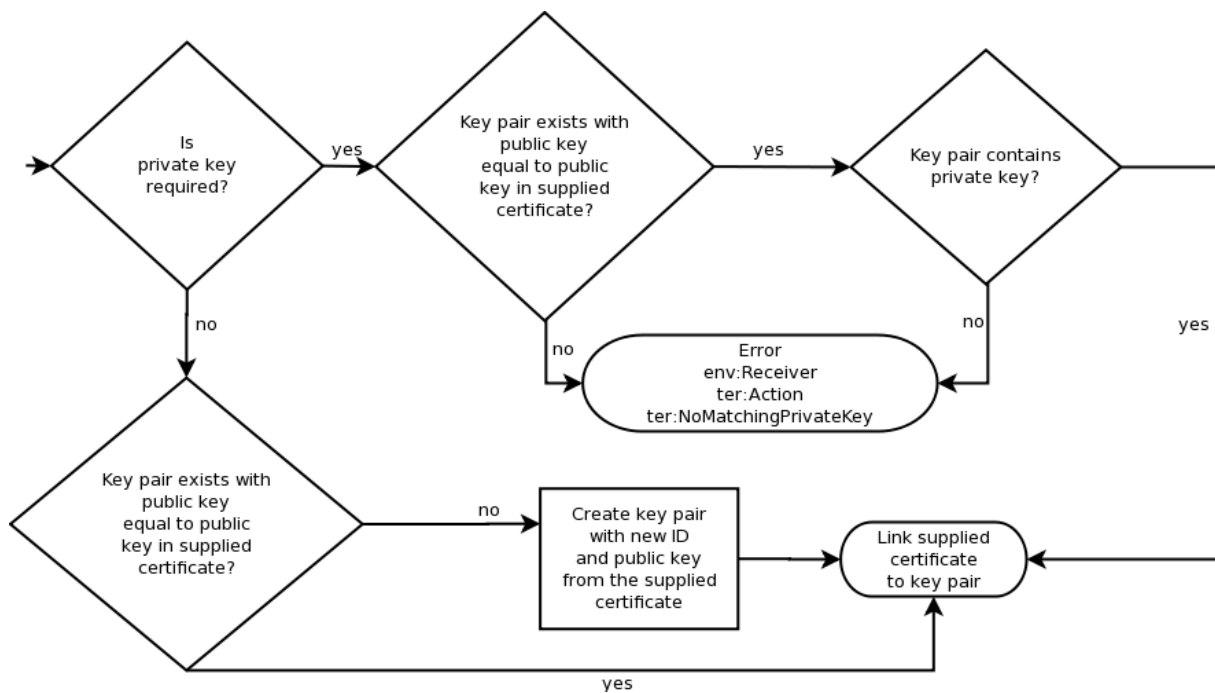


Figure 2 Link establishment between certificate and key pair for Upload Certificate

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 12: Upload Certificate command

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

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 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.

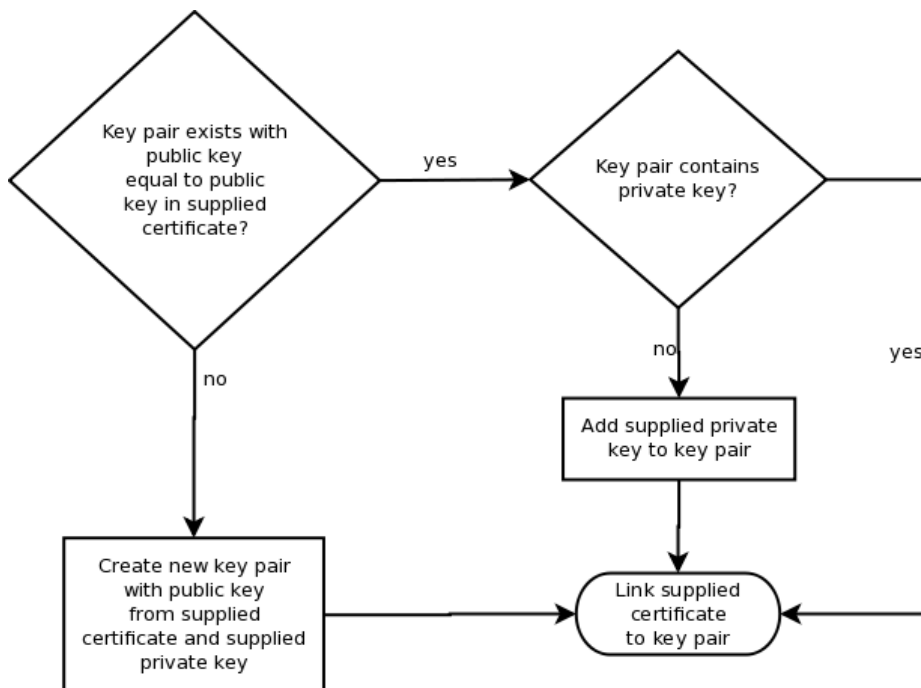


Figure 3 Link establishment between certificates and key pair for Upload Certificate with Private Key in PKCS#12

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 13: UploadCertificateWithPrivateKeyInPKCS12 command

UploadCertificateWithPrivateKeyInPKCS12		Access Class: WRITE_SYSTEM
Message name	Description	
UploadCertificateWithPrivateKeyInPKCS12Request	<p><i>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.</i></p> <p>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]</p>	
UploadCertificateWithPrivateKeyInPKCS12Response	<p><i>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.</i></p> <p>tas:CertificationPathID CertificationPathID [1][1] tas:KeyID KeyID [1][1]</p>	
Fault codes	Description	
env:Receiver ter:Action ter:MaximumNumberOfCertificatesReached	<i>The device does not have enough storage space to store the certificate to be uploaded.</i>	
env:Receiver ter:Action ter:MaximumNumberOfKeysReached	<i>The device does not have enough storage space to store the key pair that has to be generated.</i>	
env:Receiver ter:Action ter:MaximumNumberOfCertificationPathsReached	<i>The device does not have enough storage space to store the certification path to be uploaded.</i>	
env:Sender ter:InvalidArgVal ter:PassphraseID	<i>No passphrase is stored under the requested PassphraseID.</i>	
env:Sender ter:InvalidArgVal ter:DecryptionFailed	<i>The given data could not be decrypted.</i>	
env:Sender ter:InvalidArgVal ter:BadCertificate	<i>The supplied certificate file cannot be processed by the device.</i>	

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

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 14: GetCertificate command

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

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 15: GetAllCertificates command

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

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 16: DeleteCertificate command

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

env:Sender ter:InvalidArgVal ter:ReferenceExists	<i>A reference exists for the specified certificate.</i>
--	--

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 17: CreateCertificationPath command

CreateCertificationPath		Access Class: WRITE_SYSTEM
Message name	Description	
CreateCertificationPathRequest	<i>This message contains a request for the device to create a certification path.</i> tas:CertificateIDs CertificateIDs [1][1] xs:string Alias [0][1]	
CreateCertificationPathResponse	<i>This message contains the ID of the newly generated certification path.</i> tas:CertificationPathID CertificationPathID [1][1]	
Fault codes	Description	
env:Receiver ter:Action ter:MaximumNumberOfCertificationPathsReached	<i>The device does not have enough storage space to store the certification path to be created.</i>	
env:Sender ter:InvalidArgVal ter:CertificateID	<i>For at least one of the supplied certificate IDs, there exists no certificate in the device's keystore.</i>	
env:Sender ter:InvalidArgVal ter:InvalidCertificationPath	<i>At least one certificate in the certification path is not correctly signed with the public key in the next certificate in the path.</i>	
env:Receiver ter:Action ter:CertificationPathCreationFailed	<i>Creating the certification path failed.</i>	

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 18: GetCertificationPath command

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

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 19: GetAllCertificationPaths command

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

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 20: DeleteCertificationPath command

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

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 21: AddServerCertificateAssignment command

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

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.

Table 22: RemoveServerCertificateAssignment command

RemoveServerCertificateAssignment		Access Class:WRITE_SYSTEM
Message name	Description	
RemoveServerCertificateAssignmentRequest	<p><i>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.</i></p> <p>tas:CertificationPathID CertificationPathID[1][1]</p>	
RemoveServerCertificateAssignmentResponse	<p><i>This is an empty message.</i></p>	
Fault codes	Description	
env:Sender ter:InvalidArgVal ter:OldCertificationPathID	<p><i>No certification path under the given certification path ID is associated with the TLS server.</i></p>	

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 23: ReplaceServerCertificateAssignment command

ReplaceServerCertificateAssignment		Access Class:WRITE_SYSTEM
Message name	Description	
ReplaceServerCertificateAssignmentRequest	<p><i>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.</i></p> <p>tas:CertificationPathID OldCertificationPathID[1][1] tas:CertificationPathID NewCertificationPathID[1][1]</p>	
ReplaceServerCertificateAssignmentResponse	<p><i>This is an empty message.</i></p>	
Fault codes	Description	
env:Sender ter:InvalidArgVal ter:OldCertificationPathID	<p><i>No certification path under the given certification path ID is associated with the TLS server.</i></p>	
env:Sender ter:InvalidArgVal ter:NewCertificationPathID	<p><i>No certification path is stored in the keystore under the given certification path ID.</i></p>	
env:Sender ter:InvalidArgVal ter:NoPrivateKey	<p><i>The key pair that is associated with the first certificate in the new certification path (i.e., the server certificate), does not have an associated private key.</i></p>	

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 24: GetAssignedServerCertificates command

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

5.4 Capabilities

5.4.1 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 25: GetServiceCapabilities command

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

5.4.2 Keystore Capabilities

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

Table 26: 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 to 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.
RSAPKeyPairGeneration	Indicates support for on-board RSA key pair generation.
RSAPKeyLengths	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.

5.4.3 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 capabilities are available:

Table 27: 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.
MaximumNumberOfTLSCertificationPaths	Indicates the maximum number of certification paths that may be assigned to the TLS server simultaneously.

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

5.4.4 Capability-implied Requirements

Table 28 summarizes for each capability the minimum requirements that a device signaling this capability shall satisfy; it should not be seen as a recommendation.

Table 28: Requirements implied by Capabilities

Capability	Implied Requirements
MaximumNumberOfPassphrases	<p>If greater than zero, the following commands shall be supported:</p> <ul style="list-style-type: none"> • UploadPassphrase • GetAllPassphrases • DeletePassphrase <p>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.</p>
MaximumNumberOfKeys	<p>If greater than zero, then the following commands shall be supported:</p> <ul style="list-style-type: none"> • GetKeyStatus • GetAllKeys • DeleteKey
MaximumNumberOfCertificates	<p>If greater than zero, then MaximumNumberOfKeys>0 shall hold.</p>
MaximumNumberOfCertificationPaths	<p>If greater than zero, MaximumNumberOfCertificates>=2 shall hold.</p>
RSAPrivateKeyGeneration	<p>If true, the following commands shall be supported:</p> <ul style="list-style-type: none"> • CreateRSAPrivateKeyPair • GetPrivateKeyStatus <p>If true, the list of supported RSA key lengths as indicated by the RSAPrivateKeyLengths capability shall not be empty.</p> <p>If true, MaximumNumberOfKeys>0 shall hold.</p>
PKCS8RSAPrivateKeyUpload	<p>If true, the following commands shall be supported:</p> <ul style="list-style-type: none"> • UploadKeyPairInPKCS8 • GetPrivateKeyStatus <p>If true, MaximumNumberOfPassphrases >0 shall hold.</p> <p>If true, MaximumNumberOfKeys > 0 shall hold.</p> <p>If true, the list of supported RSA key lengths as indicated by</p>

	<p>the RSAKeyLenghts capability shall not be empty.</p> <p>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.</p>
<p>PKCS12CertificateWithRSA PrivateKeyUpload</p>	<p>If true, the following commands shall be supported:</p> <ul style="list-style-type: none"> • UploadCertificateWithPrivateKeyInPKCS12 • GetPrivateKeyStatus • GetCertificate • GetAllCertificates • DeleteCertificate • GetCertificationPath • GetAllCertificationPaths • DeleteCertificationPath <p>If true, MaximumNumberOfPassphrases >0 shall hold.</p> <p>If true, MaximumNumberOfKeys >=2 shall hold.</p> <p>If true, MaximumNumberOfCertificates >=2 shall hold.</p> <p>If true, MaximumNumberOfCertificationPaths >0 shall hold.</p> <p>If true, SignatureAlgorithms shall not be empty.</p> <p>If true, the list of supported RSA key lengths as indicated by the RSAKeyLenghts capability shall not be empty.</p> <p>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.</p> <p>If true, the list of supported password-based MAC algorithms as indicated by the PasswordBasedMACAlgorithms capability shall contain at least the algorithm hmacWithSHA256.</p> <p>If true, the list of supported X.509 versions as indicated by the X.509Versions capability shall contain at least the value 3.</p> <p>If true, the list of supported signature algorithms as indicated by the SignatureAlgorithms capability shall contain at least the algorithms sha1-WithRSAEncryption and sha256WithRSAEncryption.</p>
<p>PKCS10ExternalCertificatio nWithRSA</p>	<p>If true, the following operations shall be supported:</p> <ul style="list-style-type: none"> • 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

	<p>PKCS12CertificateWithRSAPrivateKeyUpload capability</p> <ul style="list-style-type: none"> • 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. <p>If true, SignatureAlgorithms shall not be empty.</p> <p>If true, MaximumNumberOfCertificates\geq2 and MaximumNumberOfCertificationPaths$>$0 shall hold.</p> <p>If true, MaximumNumberOfKeys\geq2 shall hold.</p> <p>If true, the list of supported signature algorithms as indicated by the SignatureAlgorithms capability shall contain at least the algorithms sha1-WithRSAEncryption and sha256WithRSAEncryption.</p>
SelfSignedCertificateCreationWithRSA	<p>If true, the following commands shall be supported:</p> <ul style="list-style-type: none"> • CreateSelfSignedCertificate • GetCertificate • GetAllCertificates • DeleteCertificate <p>If true, the following operations shall be supported:</p> <ul style="list-style-type: none"> • RSA key pair generation as signaled by the RSAKeyPairGeneration capability or RSA key pair upload as signaled by the PKCS8RSAKeyPairUpload capability or RSA key pair upload as signaled by the PKCS12CertificateWithRSAPrivateKeyUpload capability <p>If true, MaximumNumberOfCertificates$>$ 0 shall hold.</p> <p>If true, SignatureAlgorithms shall not be empty</p> <p>If true, the list of supported signature algorithms as indicated by the SignatureAlgorithms capability shall contain at least the algorithms sha1-WithRSAEncryption and sha256WithRSAEncryption.</p>
TLSServerSupported	<p>If not empty, the value 1.0 shall be contained in the list of supported TLS versions.</p> <p>If not empty, PKCS10ExternalCertificationWithRSA shall be true or SelfSignedCertificateCreationWithRSA shall be true.</p> <p>If not empty, the following commands shall be supported:</p> <ul style="list-style-type: none"> • CreateCertificationPath

	<ul style="list-style-type: none"> • GetCertificationPath • GetAllCertificationPaths • DeleteCertificationPath • AddTLSServerCertificateAssignment • RemoveTLSServerCertificateAssignment • ReplaceTLSServerCertificateAssignment • GetAssignedServerCertificates <p>If not empty, MaximumNumberOfCertificationPaths\geq2 and MaximumNumberOfTLSCertificationPaths$>$0 shall hold.</p>
TLSServerSupported and PKCS10ExternalCertificationWithRSA	If both TLSServerSupported and PKCS10ExternalCertificationWithRSA are true, MaximumNumberOfCertificates \geq 3 shall hold.
MaximumNumberOfTLSCertificationPaths	If greater than zero, MaximumNumberOfCertificationPaths $>$ 0 shall hold.

5.5 Events

5.5.1 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:SimpleItemDescription minOccurs="0" Name="OldStatus"
      Type="tas:KeyStatus"/>
    <tt:SimpleItemDescription Name="NewStatus"
      Type="tas:KeyStatus"/>
  </tt>Data>
</tt:MessageDescription>

```

5.6 Service specific data types

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

5.7 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 29: Advanced security service specific fault codes

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

	ter: MaximumNumberOfKeysReached	keys reached	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	PassphraseID not appropriate	No passphrase is stored under the requested PassphraseID.
	ter:PassphraseID		
env:Sender	ter:InvalidArgVal	Invalid PKCS#8 File	The PKCS#8 data structure cannot be processed by the device.
	ter:BadPKCS8File		
env:Sender	ter:InvalidArgVal	Decryption failed	The given data could not be decrypted.
	ter:DecryptionFailed		
env:Sender	ter:InvalidArgVal	Public and private key do not match	The supplied private key does not match the supplied public key.
	ter:PublicPrivateKeyMismatch		
env:Sender	ter:InvalidArgVal	Invalid PKCS#12 File	The PKCS#12 data structure cannot be processed by the device.
	ter:BadPKCS12File		

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.

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. 5.2.6.1.5) 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.

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