ONVIF™
Advanced SecurityServiceSpecification

Version 1.0.2
June2014

CONTENTS

1 Scope

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

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

2 Normative references

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 2986 PKCS #10 : Certification RequestSyntaxSpecification Version 1.7

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

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

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

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

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

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

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

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

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

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

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

Unified Modeling Language (UML)

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

3 Terms and Definitions

3.1 Definitions

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.
RSA key pair	A key pair that is accepted as input by the RSA algorithm.
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.
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 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.
Certification Authority	A certification authority is an entity that issues certificates to subject entities.
Alias	An alias is a name for an object on the device that is chosen by the client and treated transparently by the device.

3.2 Abbreviations

CA	Certification Authority
CSR	Certificate Signing Request (also called Certification Request)
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
ter	http://www.onvif.org/ver10/error
tt	http://www.onvif.org/ver10/schema
xs	http://www.w3.org/2001/XMLSchema
tas	http://www.onvif.org/ver10/advancedsecurity/wsdl

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/wsdl/advancedsecurity.wsdl>.

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.

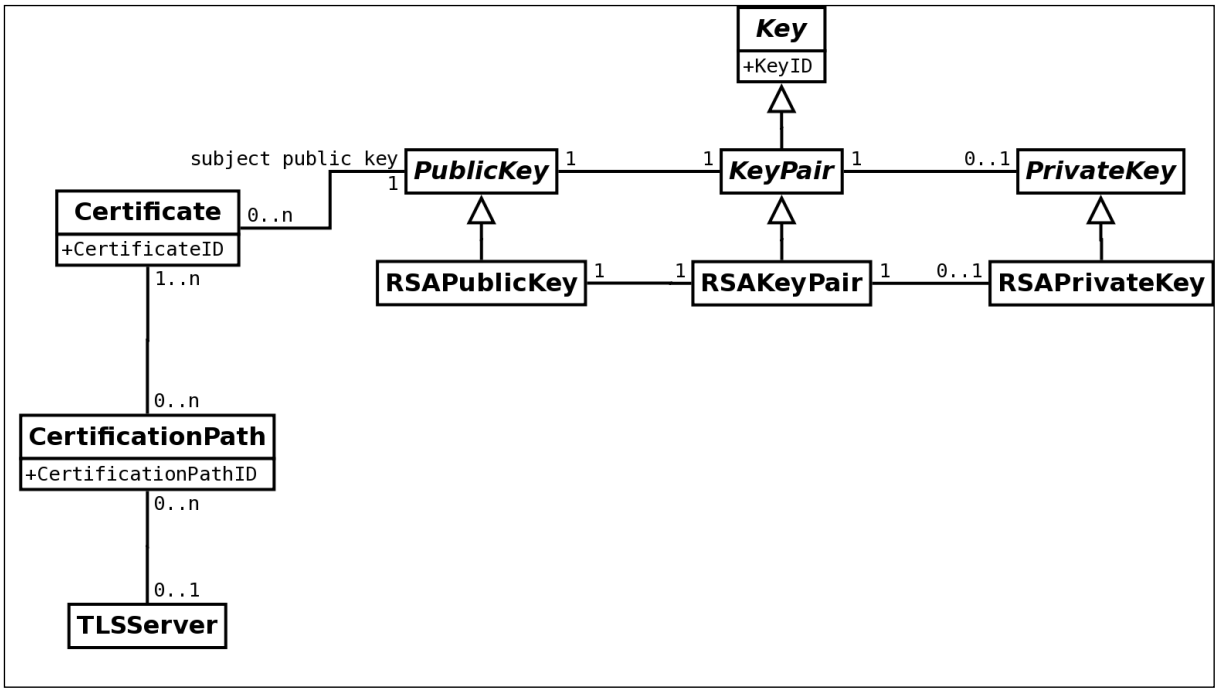

Figure 1ONVIF Advanced Security Service[UML]Class Diagram

5.2 Keystore

5.2.1 Elements of the Keystore

The keystoresecurity feature handles the storage and management of keys and certificates on an ONVIF device.

Thekeystore specified in this document supports 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.

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.

Keys in the keystoreshallbe uniquely identified by key IDs, certificates shall be uniquely identified by certificate IDs, and certification paths in the keystoreshall 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 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 certificate and multiple copies of the samecertification 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 objectis decremented. Deleting a key, certificate, or certification pathis 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.
- 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 deviceshall 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 Key Management

5.2.6.1.1 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.

---

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

A device signalling support for RSA key pair generation via the RSAKeyPairGeneration capability shall support this command.

Table 1:CreateRSAKeyPair command

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

5.2.6.1.2 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.

A device that indicates support for key handling via the MaximumNumberOfKeys capability shall support this command.

Table 2: GetKeyStatus command

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

5.2.6.1.3 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.

A device that indicates support for RSA key pairs via the RSAKeyPairGenerationcapability shall support this command.

Table 3: GetPrivateKeyStatus command

GetPrivateKeyStatus	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetPrivateKeyStatusRequest	*This message contains a request for the device to return whether a key pair contains a private key.* tas:KeyID**KeyID**[1][1]
GetPrivateKeyStatusResponse	*This message contains the status for the requested KeyID.* xs:boolean**hasPrivateKey**[1][1]
Fault codes	**Description**
env:Sender ter:InvalidArgVal ter:KeyID	*No key is stored under the requested KeyID.*
env:Sender ter:InvalidArgVal ter:InvalidKeyType	*The key stored under the requested KeyID does not identify a key pair.*

5.2.6.1.4 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.

A device that indicates support for key handling via the MaximumNumberOfKeys capability shall support this command.

Table 4: GetAllKeys command

GetAllKeys	Access Class: READ_SYSTEM_SECRET
Message name	**Description**
GetAllKeysRequest	*This message contains a request for the device to return information about all keys in the keystore.* *This is an empty message.*
GetAllKeysResponse	*This message contains information about all keys in the keystore.* tas:KeyAttribute**KeyAttribute** [0][unbounded]

Fault codes	**Description**
	No command-specific fault codes.

5.2.6.1.5 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 anInvalidArgVal 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 keystoreall data generated for this key.

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

A device that indicates support for key handling via the MaximumNumberOfKeys capability shall support this command.

Table 5: DeleteKey command

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

5.2.6.2 Certificate Management

5.2.6.2.1 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, thedevice shall encode the attribute inthe format given in the hexadecimal format.

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

A device signalling support for creating PKCS#10 certification requests via the PKCS10ExternalCertificationWithRSA capability shall support this command.

Table 6: CreatePKCS10CSR command

CreatePKCS10CSR	Access Class: READ_SYSTEM
Message name	**Description**
CreatePKCS10CSRRequest	*This message contains a request for the device to create a PKCS#10 certification request for one of its public keys.* tas:DistinguishedName**Subject** [1][1] tas:KeyID**KeyID**[1][1] tas:CSRAttribute**Attribute** [0][unbounded] tas:AlgorithmIdentifier**SignatureAlgorithm**[1][1]
CreatePKCS10CSRResponse	*This message contains the DER encoded PKCS#10 certification request.* tas:Base64DERencodedASN1Value**PKCS10CSR** [1][1]
Fault codes	**Description**
env:Receiver ter:Action ter:CSRCreationFailed	*The generation of the PKCS#10 certification request failed.*
env:Sender ter:InvalidArgVal ter:KeyID	*No key is stored under the requested KeyID or the key specified by the requested Key ID is not an asymmetric key pair.*

env:Sender ter: InvalidArgVal ter:UnsupportedSignatureAlgorithm	*The specified signature algorithm is not supported by the device.*
env:Sender ter: InvalidArgValter:KeySignatureAlgorithmMismatch	*The specified public key is an invalid input to the specified signature algorithm.*
env:Sender ter:InvalidArgVal ter:InvalidKeyStatus	*The key with the requested KeyID has an inappropriate status.*
env:Sender ter:InvalidArgVal ter:InvalidSubject	*The specified subject is invalid or incomplete.*
env:Sender ter:InvalidArgVal ter:InvalidAttribute	*The specified attribute is invalid or incomplete.*

5.2.6.2.2 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.ThenotValidAfter 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 inthe format given in the hexadecimal format.

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

A device signalling support for creating RSA-based self-signed certificates via the SelfSignedCertificateCreationWithRSA capability shall support this command.

Table 7: CreateSelfSignedCertificate command

CreateSelfSignedCertificate	Access Class: WRITE_SYSTEM
Message name	**Description**
CreateSelfSignedCertificateRequest	*This message contains a request for the device to create for a public key on the device a self-signed, RFC5280 compliant certificate.* xs:positive**X509Version** [0][1] tas:DistinguishedName**Subject** [1][1] tas:KeyID**KeyID**[1][1] xs:string**Alias** [0][1] xs:dateTime**notValidBefore** [0][1] xs:dateTime**notValidAfter**[0][1] tas:AlgorithmIdentifier**SignatureAlgorithm**[1][1] tas:X509v3Extension **Extension** [0][unbounded]
CreateSelfSignedCertificateResponse	*This message contains the certificate ID of the successfully created certificate.* tas:CertificateID**CertificateID** [1][1]
Fault codes	**Description**
env:Receiver ter:Action ter:CertificateCreationFailed	*The generation of the self-signed certificate failed.*
env:Receiver ter:Action ter:MaximumNumberOfCertificatesReached	*The device does not have enough storage space to store the certificate to be created.*
env:Sender ter:InvalidArgVal ter:UnsupportedX509Version	*The specified X.509 version is not supported by the device.*

env:Sender ter:InvalidArgVal ter:KeyID	*No key is stored under the requested KeyID or the key specified by the requested Key ID is not an asymmetric key pair.*
env:Sender ter: InvalidArgVal ter:UnsupportedSignatureAlgorithm	*The specified signature algorithm is not supported by the device.*
env:Sender ter: InvalidArgValter:KeySignatureAlgorithmMismatch	*The specified public key is an invalid input to the specified signature algorithm.*
env:Sender ter: InvalidArgVal ter:X509VersionExtensionsMismatch	*The request contains extensions which are not supported by the X509Versionin the request.*
env:Sender ter:InvalidArgVal ter:InvalidKeyStatus	*The key with the requested KeyID has an inappropriate status.*
env:Sender ter:InvalidArgVal ter:InvalidSubject	*The specified subject is invalid or incomplete.*
env:Sender ter:InvalidArgVal ter:InvalidDateTime	*A specified dateTime is invalid.*

5.2.6.2.3 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,andkey 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.

Ifno private key is required for the public key in the certificate and a key pair exists in the keystorewith 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 keypair 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.

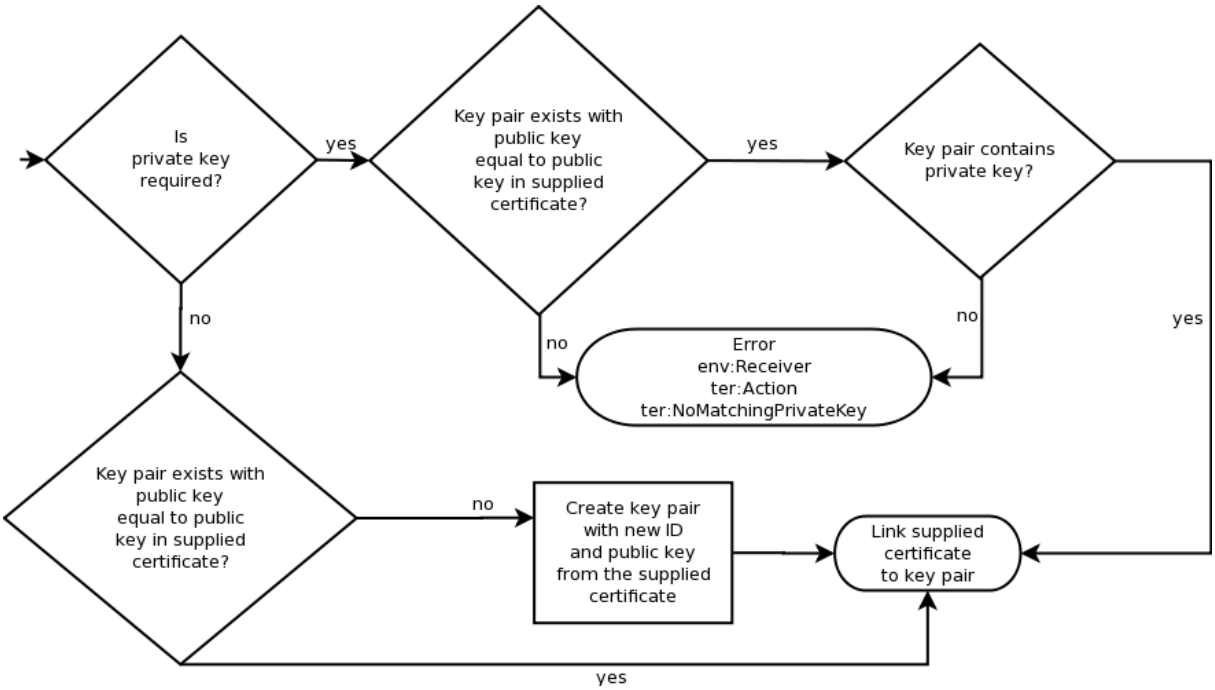

Figure 2Link establishment between certificate and key pair for Upload Certificate

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.

A device signalling support for PKCS#10 External Certification with the PKCS10ExternalCertificationWithRSAcapability shall support this command.

Table 8: Upload Certificate command

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

5.2.6.2.4 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.

A device that supports the Create Self Signed Certificate command or the Upload certificate command shall support this command.

Table 9: GetCertificate command

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

5.2.6.2.5 Get All Certificates

This operation returns the IDs of 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.

A device that supports the Create Self Signed Certificate command or the Upload certificate command shall support this command.

Table 10: GetAllCertificates command

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

5.2.6.2.6 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.

A device that supports the Create Self Signed Certificate command or the Upload certificate command shall support this command.

Table 11: DeleteCertificate command

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

5.2.6.2.7 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.

A device signalling support for TLS via the TLSServerSupported capability shall support this command.

Table 12: CreateCertificationPath command

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

5.2.6.2.8 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.

A device signalling support for TLS via theTLSServerSupported capability shall support this command.

Table 13: GetCertificationPath command

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

ter:CertificationPathID	*path ID.*

5.2.6.2.9 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.

A device signalling support for TLS via theTLSServerSupported capability shall support this command.

Table 14: GetAllCertificationPaths command

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

5.2.6.2.10 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.

A device signalling support for TLS via theTLSServerSupported capability shall support this command.

Table 15: DeleteCertificationPath command

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

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.

A device signalling a TLS server implementation via the TLSServerSupported capability shall support this command.

Table 16: AddServerCertificateAssignment command

AddServerCertificateAssignment	Access Class:WRITE_SYSTEM
Message name	**Description**
AddServerCertificateAssignmentRequest	*This message contains a request for the device to assign a certificate along with a certification path to the TLS server.* tas:CertificationPathID**CertificationPathID**[1][1]
AddServerCertificateAssignmentResponse	*This is an empty message.*
Fault codes	**Description**
env:Receiver ter:InvalidArgVal ter:CertificationPathID	*No certification path is stored in the keystore under the given certification path ID.*
env:Receiver ter:InvalidArgVal ter:NoPrivateKey	*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.*
env: Receiver ter: Action ter:MaximumNumberOfTLSCertificationPathsReached	*The maximum number of certification paths that may be assigned to the TLS server simultaneously is reached.*

5.3.2.2 Remove Server Certificate Assignment

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

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

A device signalling a TLS server implementation via the TLSServerSupported capability shall support this command.

Table 17: RemoveServerCertificateAssignment command

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

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.

A device signalling a TLS server implementation via the TLSServerSupported capability shall support this command.

Table 18: ReplaceServerCertificateAssignment command

ReplaceServerCertificateAssignment	Access Class:WRITE_SYSTEM
Message name	**Description**
ReplaceServerCertificateAssignmentRequest	*This message contains a request for the device to replace a TLS server certificate assignment to the TLS server by a new key pair and certificate assignment.* tas:CertificationPathID**OldCertificationPathID**[1][1] tas:CertificationPathID**NewCertificationPathID**[1][1]
ReplaceServerCertificateAssignmentResponse	*This is an empty message.*
Fault codes	**Description**
env:Receiver ter:InvalidArgVal ter:OldCertificationPathID	*No certification path under the given certification path ID is associatedwith the TLS server.*
env:Receiver ter:InvalidArgVal ter:NewCertificationPathID	*No certification path is stored in the keystore under the given certification path ID.*
env:Receiver ter:InvalidArgVal ter:NoPrivateKey	*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.*

5.3.2.4 Get Assigned Server Certificates

This operation returns the IDs of all key pairs and certificates (including 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.

A device signalling a TLS server implementation via the TLSServerSupported capability shall support this command.

Table 19: GetAssignedServerCertificates command

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

5.4 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 20: GetServiceCapabilitites command

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

5.4.2 Keystore Capabilities

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

Table 21: Keystore Capabilities

Capability Name	**Capability Semantics**
MaximumNumberOfKeys	Indicates the maximum number of keys that the device is able store simultaneously.
MaximumNumberOfCertificates	Indicates the maximum number of certificates that the device is able to store simultaneously.
MaximumNumberOfCertificationPaths	Indicates the maximum number of certificate paths that the device is able to store simultaneously.
RSAKeyPairGeneration	Indicates support for on-board RSA key pair generation.
RSAKeyLengths	Indicates which RSA key lengths are supported by the device.
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.
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 capabilites are available:

Table 22: 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 23 summarizes for each capability the minimum requirements that a device signaling this capability shall satisfy; it should not be seen as a recommendation.

Table 23: Requirements implied by Capabilities

Capability	Implied Requirements
MaximumNumberOfKeys	If greater than zero, then the following commands shall be supported: • GetKeyStatus • GetAllKeys • DeleteKey
MaximumNumberOfCertificates	If greater than zero, then MaximumNumberOfKeys>0 shall hold.
MaximumNumberOfCertificationPaths	If greater than zero, MaximumNumberOfCertificates>=2 shall hold.
RSAKeyPairGeneration	If true, the following commands shall be supported: • CreateRSAKeyPair • GetPrivateKeyStatus If true, the list of supported RSA key lengths as indicated by the RSAKeyLenghts capability shall not be empty. If true,MaximumNumberOfKeys>0 shall hold.
PKCS10ExternalCertificationWithRSA	If true, the following operations shall be supported: • RSA key pair generation as signaled by the RSAKeyPairGeneration capability • Creating a CSR with the CreatePKCS10CSR command. • Uploading the certificate created for the CSR as well as the certificate of the created certificate's signer with the UploadCertificate command. If true, SignatureAlgorithmsshall not be empty. If true, MaximumNumberOfCertificates>=2 and MaximumNumberOfCertificationPaths>0 shall hold. If true,MaximumNumberOfKeys>=2shall hold.
SelfSignedCertificateCreationWithRSA	If true, the following commands shall be supported: • CreateSelfSignedCertificate • GetCertificate

	• GetAllCertificates • DeleteCertificate If true, the following operations shall be supported: • RSA key pair generation as signaled by the RSAKeyPairGeneration capability If true, MaximumNumberOfCertificates> 0 shall hold. If true, SignatureAlgorithmsshall not be empty
TLSServerSupported	If not empty, the value 1.0 shall be contained in the list of supported TLS versions. If not empty, the following commands shall be supported: • CreateCertificationPath • GetCertificationPath • GetAllCertificationPaths • DeleteCertificationPath • AddTLSServerCertificateAssignment • RemoveTLSServerCertificateAssignment • ReplaceTLSServerCertificateAssignment • GetAssignedServerCertificates If true, MaximumNumberOfCertificationPaths>=2and MaximumNumberOfTLSCertificationPaths>0 shall hold.
TLSServerSupported and PKCS10ExternalCertificationWithRSA	If both TLSServerSupported and PKCS10ExternalCertificationWithRSA are true, MaximumNumberOfCertificates>=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:SimpleItemDescriptionminOccurs=”0” Name="OldStatus"
            Type="tas:KeyStatus"/>
            <tt:SimpleItemDescription Name="NewStatus"
            Type="tas:KeyStatus"/>
      </tt:Data>
</tt:MessageDescription>
```

5.6 Service specific data types

```
<xs:simpleType name="ID">
<xs:restriction base="xs:token">
<xs:annotation>
<xs:documentation>Unique identifier for objects in the key
store.</xs:documentation>
</xs:annotation>
</xs:restriction>
</xs:simpleType>
<!--================================-->
<xs:simpleType name="KeyID">
<xs:restriction base="xs:ID">
<xs:annotation>
<xs:documentation>Unique identifier for keys in the key
store.</xs:documentation>
</xs:annotation>
</xs:restriction>
</xs:simpleType>
<!--================================-->
<xs:simpleType name="CertificateID">
<xs:restriction base="xs:ID">
<xs:annotation>
<xs:documentation>Unique identifier for certificates in the key
store.</xs:documentation>
</xs:annotation>
</xs:restriction>
</xs:simpleType>
<!--================================-->
<xs:simpleType name="CertificationPathID">
<xs:restriction base="xs:ID">
<xs:annotation>
<xs:documentation>Unique identifier for certification paths in the key
store.</xs:documentation>
</xs:annotation>
</xs:restriction>
</xs:simpleType>
<!--================================-->
<xs:simpleType name="KeyStatus">
<xs:restriction base="xs:string">
<xs:enumeration value="ok">
<xs:annotation>
<xs:documentation>Key is ready for use</xs:documentation>
</xs:annotation>
</xs:enumeration>
<xs:enumeration value="generating">
<xs:annotation>
<xs:documentation>Key is being generated</xs:documentation>
</xs:annotation>
</xs:enumeration>
<xs:enumeration value="corrupt">
<xs:annotation>
<xs:documentation>Key has not been successfully generated and cannot be
used.</xs:documentation>
</xs:annotation>
```

```
</xs:enumeration>
</xs:restriction>
</xs:simpleType>
<!--================================-->
<xs:simpleType name="DotDecimalOID">
<xs:restriction base="xs:string">
<xs:pattern value="[0-9]+(.[0-9]+)*">
<xs:annotation>
<xs:documentation>An object identifier (OID) in dot-decimal form as
specified in RFC4512.</xs:documentation>
</xs:annotation>
</xs:pattern>
</xs:restriction>
</xs:simpleType>
<!--================================-->
<xs:simpleType name="DNAttributeType">
<xs:restriction base="xs:string">
<xs:annotation>
<xs:documentation>The distinguished name attribute type shall be encoded as
specified in RFC 4514.</xs:documentation>
</xs:annotation>
</xs:restriction>
</xs:simpleType>
<!--================================-->
<xs:simpleType name="DNAttributeValue">
<xs:restriction base="xs:string">
<xs:annotation>
<xs:documentation>The distinguished name attribute values are encoded in
UTF-8 or in hexadecimal form as specified in RFC 4514.
</xs:documentation>
</xs:annotation>
</xs:restriction>
</xs:simpleType>
<!--================================-->
<xs:complexType name="KeyAttribute">
<xs:sequence>
<xs:element name="KeyID" type="tas:KeyID">
<xs:annotation>
<xs:documentation>The ID of the key.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Alias" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>The alias of the key</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="hasPrivateKey" type="xs:boolean" minOccurs="0">
<xs:annotation>
<xs:documentation>Absent if the key is not a key pair. True if and only if
the key is a key pair and contains a private key. False if and only if the
key is a key pair and does not contain a private key.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="KeyStatus" type="tas:KeyStatus">
<xs:annotation>
<xs:documentation>The status of the key.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:any minOccurs=”0” maxOccurs=”unbounded” namespace="##any"
processContents="lax"/>
</xs:sequence>
<xs:anyAttribute processContents="lax"/>
```

```
</xs:complexType>
<!--================================-->
<xs:complexType name="DNAttributeTypeAndValue">
<xs:sequence>
<xs:element name="Type" type="tas:DNAttributeType"/>
<xs:element name="Value" type="tas:DNAttributeValue"/>
</xs:sequence>
</xs:complexType>
<!--================================-->
<xs:complexType name="MultiValuedRDN">
<xs:annotation>
<xs:documentation>A multi-valued RDN</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element minOccurs="0" maxOccurs="unbounded" name="Attribute"
type="tas:DNAttributeTypeAndValue">
<xs:annotation>
<xs:documentation>A list of types and values defining a multi-valued
RDN</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<!--================================-->
<xs:complexType name="DistinguishedName">
<xs:sequence>
<xs:element minOccurs="0" maxOccurs="unbounded" name="Country"
type="tas:DNAttributeValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded" name="Organization"
type="tas:DNAttributeValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded" name="OrganizationalUnit"
type="tas:DNAttributeValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded"
name="DistinguishedNameQualifier" type="tas:DNAttributeValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded" name="StateOrProvinceName"
type="tas:DNAttributeValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded" name="CommonName"
type="tas:DNAttributeValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded" name="SerialNumber"
type="tas:DNAttributeValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded" name="Locality"
type="tas:DNAttributeValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded" name="Title"
type="tas:DNAttributeValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded" name="Surname"
type="tas:DNAttributeValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded" name="GivenName"
type="tas:DNAttributeValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded" name="Initials"
type="tas:DNAttributeValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded" name="Pseudonym"
type="tas:DNAttributeValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded" name="GenerationQualifier"
type="tas:DNAttributeValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded" name="GenericAttribute"
type="tas:DNAttributeTypeAndValue"/>
<xs:element minOccurs="0" maxOccurs="unbounded" name="MultiValuedRDN"
type="tas:MultiValuedRDN">
<xs:annotation>
<xs:documentation>A multi-valued RDN</xs:documentation>
</xs:annotation>
</xs:element>
```

```
<xs:element minOccurs="0" name="anyAttribute">
<xs:complexType>
<xs:sequence>
<xs:any minOccurs=”0” maxOccurs="unbounded" namespace="##any"
processContents="lax"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<!--================================-->
<xs:complexType name="AlgorithmIdentifier">
<xs:sequence>
<xs:element name="algorithm" type="tas:DotDecimalOID">
<xs:annotation>
<xs:documentation>OID of the algorithm in dot-decimal
form</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element minOccurs="0" name="parameters"
type="tas:Base64DERencodedASN1Value">
<xs:annotation>
<xs:documentation>Optional parameters of the algorithm</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element minOccurs="0" name="anyParameters">
<xs:complexType>
<xs:sequence>
<xs:any minOccurs=”0” maxOccurs="unbounded" namespace="##any"
processContents="lax"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<!--================================-->
<xs:complexTypename="BasicRequestAttribute">
<xs:annotation>
<xs:documentation>A CSR attribute as specified in RFC
2986.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:elementname="OID" type="tas:DotDecimalOID">
<xs:annotation>
<xs:documentation>The OID of the attribute.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="value" type="tas:Base64DERencodedASN1Value">
<xs:annotation>
<xs:documentation>The value of the attribute as a base64-encoded DER
representation of an ASN.1 value.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:anyminOccurs="0" maxOccurs="unbounded" namespace="##any"
processContents="lax"/>
</xs:sequence>
<xs:anyAttributeprocessContents="lax"/>
</xs:complexType>
<!--================================-->
<xs:complexType name="CSRAttribute">
<xs:annotation>
```

```
<xs:documentation>A CSR attribute as specified in
PKCS#10</xs:documentation>
</xs:annotation>
<xs:choice>
<xs:element name="X509v3Extension" type="tas:X509v3Extension">
<xs:annotation>
<xs:documentation>Extension request</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementname="BasicRequestAttribute" type="tas:BasicRequestAttribute">
<xs:annotation>
<xs:documentation>A basic CSR attribute.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element minOccurs="0" name="anyAttribute">
<xs:complexType>
<xs:sequence>
<xs:any minOccurs=”0” maxOccurs="unbounded" namespace="##any"
processContents="lax"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
<!--================================-->
<xs:simpleType name="Base64DERencodedASN1Value">
<xs:restriction base="xs:base64Binary"/>
</xs:simpleType>
<!--================================-->
<xs:complexType name="X509v3Extension">
<xs:annotation>
<xs:documentation>An X.509v3 extension field as specified in RFC
5280</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="extnOID" type="tas:DotDecimalOID">
<xs:annotation>
<xs:documentation>The OID of the extension field.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element default="false" name="critical" type="xs:boolean">
<xs:annotation>
<xs:documentation>True if and only if the extension is
critical.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="extnValue" type="tas:Base64DERencodedASN1Value">
<xs:annotation>
<xs:documentation>The value of the extension field as a base64-encoded DER
representation of an ASN.1 value.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:anyminOccurs="0" maxOccurs="unbounded" namespace="##any"
processContents="lax"/>
</xs:sequence>
</xs:complexType>
<!--================================-->
<xs:complexType name="X509Certificate">
<xs:sequence>
<xs:element name="CertificateID" type="tas:CertificateID">
<xs:annotation>
<xs:documentation>The ID of the certificate</xs:documentation>
```

```
</xs:annotation>
</xs:element>
<xs:element name="KeyID" type="tas:KeyID">
<xs:annotation>
<xs:documentation>The ID of the key that this certificate associates to the
certificate subject.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Alias" type="xs:string" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation>The alias of the certificate</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="CertificateContent" type="tas:Base64DERencodedASN1Value">
<xs:annotation>
<xs:documentation>The base64-encoded DER representation of the X.509
certificate</xs:documentation>
</xs:annotation>
</xs:element>
<xs:anyminOccurs="0" maxOccurs="unbounded" namespace="##any"
processContents="lax"/>
</xs:sequence>
</xs:complexType>
<!--================================-->
<xs:complexType name="CertificateIDs">
<xs:sequence>
<xs:element maxOccurs="unbounded" name="CertificateID"
type="tas:CertificateID">
<xs:annotation>
<xs:documentation>A certificate in the list of certificate
IDs</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<!--================================-->
<xs:complexType name="CertificationPath">
<xs:sequence>
<xs:element maxOccurs="unbounded" name="CertificateID"
type="tas:CertificateID">
<xs:annotation>
<xs:documentation>A certificate in the certification
path</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Alias" type="xs:string" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation>The alias of the certification path</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<!--================================-->
<xs:simpleType name="RSAKeyLengths">
<xs:list itemType="xs:int"/>
</xs:simpleType>
<xs:simpleType name="X509Versions">
<xs:list itemType="xs:int"/>
</xs:simpleType>
<xs:simpleType name="TLSVersions">
<xs:list itemType="xs:string"/>
</xs:simpleType>
```

```
<!--================================-->
<xs:complexType name="KeystoreCapabilities">
<xs:sequence>
<xs:element minOccurs="0" maxOccurs="unbounded" name="SignatureAlgorithms"
type="tas:AlgorithmIdentifier"/>
</xs:sequence>
<xs:attribute name="MaximumNumberOfKeys" type="xs:positiveInteger">
<xs:annotation>
<xs:documentation>Indicates the maximum number of keys that the device can
store simultaneously.</xs:documentation>
</xs:annotation>
</xs:attribute>
<xs:attribute name="MaximumNumberOfCertificates" type="xs:positiveInteger">
<xs:annotation>
<xs:documentation>Indicates the maximum number of certificates that the
device can store simultaneously.</xs:documentation>
</xs:annotation>
</xs:attribute>
<xs:attribute name="MaximumNumberOfCertificationPaths"
type="xs:positiveInteger">
<xs:annotation>
<xs:documentation>Indicates the maximum number of certification paths that
the device can store simultaneously.</xs:documentation>
</xs:annotation>
</xs:attribute>
<xs:attribute name="RSAKeyPairGeneration" type="xs:boolean">
<xs:annotation>
<xs:documentation>Indication that the device supports on-board RSA key pair
generation.</xs:documentation>
</xs:annotation>
</xs:attribute>
<xs:attribute name="RSAKeyLengths" type="tas:RSAKeyLengths">
<xs:annotation>
<xs:documentation>Indicates which RSA key lengths are supported by the
device.</xs:documentation>
</xs:annotation>
</xs:attribute>
<xs:attribute name="PKCS10ExternalCertificationWithRSA" type="xs:boolean">
<xs:annotation>
<xs:documentation>Indication support for generating PKCS#10
requests.</xs:documentation>
</xs:annotation>
</xs:attribute>
<xs:attribute name="SelfSignedCertificateCreationWithRSA"
type="xs:boolean">
<xs:annotation>
<xs:documentation>Indication support for creating self-signed
certificates.</xs:documentation>
</xs:annotation>
</xs:attribute>

<xs:attribute name="X509Versions" type="tas:X509Versions">
<xs:annotation>
<xs:documentation>Indicates which X.509 versions are supported by the
device.</xs:documentation>
</xs:annotation>
</xs:attribute>
<xs:anyAttribute processContents="lax"/>
</xs:complexType>
<!--================================-->
<xs:complexType name="TLSServerCapabilities">
<xs:sequence>
```

```
<xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="TLSServerSupported" type="tas:TLSVersions">
<xs:annotation>
<xs:documentation>Indicates which TLS versions are supported by the
device.</xs:documentation>
</xs:annotation>
</xs:attribute>
<xs:attribute name="MaximumNumberOfTLSCertificationPaths"
type="xs:positiveInteger">
<xs:annotation>
<xs:documentation>Indicates the maximum number of certification paths that
may be assigned to the TLS server simultaneously.</xs:documentation>
</xs:annotation>
</xs:attribute>
<xs:anyAttribute processContents="lax"/>
</xs:complexType>
<!--================================-->
<xs:complexType name="Capabilities">
<xs:sequence>
<xs:element name="KeystoreCapabilities" type="tas:KeystoreCapabilities"/>
<xs:element name="TLSServerCapabilities" type="tas:TLSServerCapabilities"/>
<xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
</xs:sequence>
<xs:anyAttribute processContents="lax"/>
</xs:complexType>

<xs:element name="Capabilities" type="tas:Capabilities"/>
```

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 24: Advanced security service specific fault codes

Fault Code	Parent Subcode Subcode	Fault Reason	Description
env:Sender	ter:InvalidArgVal ter:KeyID	KeyID not appropriate	No key is stored under the requested KeyID.
env:Sender	ter:InvalidArgVal ter:InvalidKeyType	Key type invalid	The key stored in the keystore under the requested KeyID is of an invalid type.
env:Receiver	ter:Action ter:KeyDeletionFailed	Deletion of a key failed.	Deleting the key with the requested KeyID failed.
env:Receiver	ter:Action ter:CSRCreationFailed	Failure to create a CSR	The generation of the PKCS#10 certification request failed.
env:Sender	ter:InvalidArgVal ter:UnsupportedSignatureAlgorithm	Signature algorithm not supported	The specified signature algorithm is not supported by the device.
env:Sender	ter:InvalidArgVal	Mismatch of key and	The specified public key is an

	ter:KeySignatureAlgorithmMismatch	signature algorithm	invalid input to the specified signature algorithm.
env:Sender	ter:InvalidArgVal ter:InvalidKeyStatus	KeyStatus invalid	The key with the requested KeyID has an inappropriate status.
env:Sender	ter:InvalidArgVal ter:InvalidSubject	Subject invalid	The specified subject is invalid or incomplete.
env:Sender	ter:InvalidArgVal ter:InvalidAttribute	Attribute invalid	The specified attribute is invalid or incomplete.
env:Sender	ter:InvalidArgVal ter:InvalidDateTime	dateTime invalid	The specified dateTime is invalid.
env:Receiver	ter:Action ter:CertificateCreationFailed	Certificate creation failed.	The generation of a certificate failed.
env:Receiver	ter:Action ter:MaximumNumberOfCertificatesReached	Maximum number of certificates reached	The device does not have enough storage space to store the certificate to be created.
ter:Sender	ter:InvalidArgVal ter:UnsupportedX509Version	X509 version not supported	The specified X.509 version is not supported by the device.
env:Sender	ter:InvalidArgVal ter:X509VersionExtensionsMismatch	Extensions not supported	The request contains extensions that are not supported by the X.509 version specified in the request.
env:Receiver	ter:Action ter: MaximumNumberOfKeysReached	Maximum number of keys reached	The keystore does not have enough storage space to store the key pair that has to be generated.
env:Sender	ter:InvalidArgVal ter:BadCertificate	Certificate bad	The supplied certificate cannot be processed by the device.
env:Sender	ter:InvalidArgVal ter:UnsupportedPublicKeyAlgorithm	Public key algorithm not supported	The public key algorithm of the public key in the certificate is not supported by the device.
env:Receiver	ter:Action ter:NoMatchingPrivateKey	Matching private key not found.	The keystore does not contain a key pair with a private key that matches the public key in the uploaded certificate.
env:Sender	ter:InvalidArgVal ter:CertificateID	CertificateID not appropriate	No certificate is stored under the requested CertificateID.
env:Receiver	ter:Action ter:CertificateDeletionFailed	Deletion of a certificate failed.	Deleting the certificate with the requested CertificateID failed.
env:Sender	ter:InvalidArgVal ter:ReferenceExists	ReferenceExists	A reference exists for the object that is to be deleted.
env:Sender	ter:InvalidArgVal ter:InvalidCertificationPath	CertificationPath invalid	At least one certificate in the certification path is not correctly signed with the public key in the next certificate in the path.
env:Receiver	ter:Action	Certification path	Creating the certification path

	ter:CertificationPathCreationFailed	creation failed.	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.

5.8 Protocol Options

This section summarizes in Table 25 mandatory configurations of cryptographic protocols that are used by the ONVIF advanced security service.

Table 25 Configuration options of cryptographic protocols

Operation	Protocol	Mandatory configuration options
CreatePKCS10CSR	PKCS#10	sha-1WithRSAEncryption sha256WithRSAEncryption
CreateSelfSignedCertificate	X.509v3	sha-1WithRSAEncryption sha256WithRSAEncryption
UploadCertificate	X.509v3	sha-1WithRSAEncryption sha256WithRSAEncryption

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 to distinct.

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-featuresto 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 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