# ONVIF®

# Uplink Device Test Specification

Version 25.12

December 2025

# REVISION HISTORY

| Vers. | Date | Description |
|---|---|---|
| 23.06 | Jan 04, 2023 | Initial version |
| 25.06 | Mar 04, 2025 | The following new test cases were added as a part of #301 task:<br><br>UPLINK-3-1-1 Uplink over WebSocket with mTLS Authentication (new)<br><br>UPLINK-3-1-2 Uplink over WebSocket with access token authentication (OAuthClientCredentials, client_secret_basic) (new)<br><br>UPLINK-3-1-3 Uplink over WebSocket with access token authentication (OAuthClientCredentials, private_key_jwt) (new)<br><br>UPLINK-3-1-4 Uplink over WebSocket with mTLS Authentication - Invalid Server Certificate (new)<br><br>UPLINK-3-1-5 Uplink over WebSocket with access token authentication (OAuthClientCredentials, client_secret_basic) - Invalid Uplink Client Certificate (new)<br><br>UPLINK-3-1-6 Uplink over WebSocket with access token authentication (OAuthClientCredentials, client_secret_basic) - Invalid Authentication Server Certificate (new)<br><br>Annex HelperCreateKeyPairAndReceivePublicKey Create Key Pair and Receive Public Key (new)<br><br>Annex HelperGetAuthorizationServerConfigurations Get Authorization Server Configurations List (added)<br><br>Annex HelperEmptySpaceForOneAuthorizationServerConfiguration Make Sure That At Least One Authorization Server Configuration Could Be Created (added)<br><br>Annex HelperCreateCertPathValidationPolicyWithCertID Create a certification path validation policy with provided certificate identifier (added)<br><br>Annex HelperCreateSignedCertificate Provide certificate signed by private key of other certificate (added)<br><br>Annex HelperCreateCertPathValidationPolicyForAuthServer Create a certification path validation policy for authentication server configuration (added)<br><br>Introduction\Scope\Authentication and Authorization (new)<br><br>Test Overview\Test Policy\Authentication and Authorization (new) |
| 25.12 | Oct 02, 2025 | The following test and annexes were updated in the scope of #429 (Authorization server uri was replaced with Authentication server metadata endpoint):<br><br>Annex HelperConfigureAndStartAuthServer Configure Authorization Server On Device and Start It<br><br>UPLINK-3-1-2 Uplink over WebSocket with access token authentication (OAuthClientCredentials, client_secret_basic) |

| | | |
|---|---|---|
| | | UPLINK-3-1-3 Uplink over WebSocket with access token authentication (OAuthClientCredentials, private_key_jwt)<br><br>UPLINK-3-1-5 Uplink over WebSocket with access token authentication (OAuthClientCredentials, client_secret_basic) - Invalid Uplink Client Certificate<br><br>UPLINK-3-1-6 Uplink over WebSocket with access token authentication (OAuthClientCredentials, client_secret_basic) - Invalid Authentication Server Certificate |

**Table of Contents**

# 1 Introduction

The goal of the ONVIF test specification set is to make it possible to realize fully interoperable IP physical security implementation from different vendors. The set of ONVIF test specification describes the test cases need to verify the [ONVIF Uplink Specification] and [ONVIF Conformance] requirements. It also describes the test framework, test setup, pre-requisites, test policies needed for the execution of the described test cases.

This ONVIF Uplink Test Specification acts as a supplementary document to the [ONVIF Uplink Specification], illustrating test cases need to be executed and passed. And also this specification acts as an input document to the development of test tool which will be used to test the ONVIF device implementation conformance towards ONVIF standard. This test tool is referred as ONVIF Client hereafter.

## 1.1 Scope

This ONVIF Uplink Test Specification defines and regulates the conformance testing procedure for the ONVIF conformant devices. Conformance testing is meant to be functional black-box testing. The objective of this specification is to provide test cases to test individual requirements of ONVIF devices according to ONVIF Uplink which is defined in [ONVIF Uplink Specification].

The principal intended purposes are:

- Provide self-assessment tool for implementations.
- Provide comprehensive test suite coverage for [ONVIF Network Interface Specs].

This specification **does not** address the following:

- Product use cases and non-functional (performance and regression) testing.
- SOAP Implementation Interoperability test i.e. Web Service Interoperability Basic Profile version 2.0 (WS-I BP 2.0).
- Network protocol implementation Conformance test for HTTP, HTTPS, RTP and RTSP protocol.
- Poor streaming performance test (audio/video distortions, missing audio/video frames, incorrect lib synchronization etc.).

  Wi-Fi Conformance test

The set of ONVIF Test Specification will not cover the complete set of requirements as defined in [ONVIF Uplink Specification]; instead it would cover subset of it. The scope of this specification is to derive all the normative requirements of [ONVIF Uplink Specification] which are related to ONVIF Uplink and some of the optional requirements.

This ONVIF Uplink Test Specification covers Uplink service which is a functional block of [ONVIF Network Interface Specs]. The following sections give the brief overview of and scope of each functional block.

### 1.1.1 Uplink Connection

The Connection section covers the test cases needed for initiating an uplink from a device to a service endpoint. Once the connection is established the service endpoint acts as client.

The scope of this specification section is to cover the following functions:

- Setting device uplink configuration.
- Connection establishment and teardown.
- Reconnecting on communication timeout.
- Verification of authentication and user level.
- Responding to requests.
- Streaming of video.

### 1.1.2 Authentication and Authorization

The Authentication and Authorization section covers the test cases needed for authentication and authorization for uplink connection.

The scope of this specification section is to cover the following functions:

- mTLS authentication
- Access token authentication with following settings:
  - OAuthClientCredentials, client_secret_basic
  - OAuthClientCredentials, private_key_jwt
- Uplink authorization
- Uplink client certificate validation
- Authorization server certificate validation

### 1.1.3 Capabilities

The Capabilities section covers the test cases needed for getting capabilities from an ONVIF device.

The scope of this specification section is to cover the following functions:

- Getting Uplink service address with GetServices command via Device service
- Getting capabilities with GetServiceCapabilities command
- Getting capabilities with GetServices command via Device service

# 2 Normative references

- [ONVIF Conformance] ONVIF Conformance Process Specification:

  https://www.onvif.org/profiles/conformance/

- [ONVIF Profile Policy] ONVIF Profile Policy:

  https://www.onvif.org/profiles/

- [ONVIF Network Interface Specs] ONVIF Network Interface Specification documents:

  https://www.onvif.org/profiles/specifications/

- [ONVIF Core Specs] ONVIF Core Specifications:

  https://www.onvif.org/profiles/specifications/

- [ONVIF Uplink Specification] Uplink Specification:

  https://www.onvif.org/profiles/specifications/

- [ONVIF Base Test] ONVIF Base Device Test Specification:

  https://www.onvif.org/profiles/conformance/device-test/

- [ONVIF RTSP via Media2 Test] Real Time Streaming using Media2 Test Specification:

  https://www.onvif.org/profiles/specifications/

- [ISO/IEC Directives, Part 2] ISO/IEC Directives, Part 2, Annex H:

  http://www.iso.org/directives

- [ISO 16484-5] ISO 16484-5:2014-09 Annex P:

  https://www.iso.org/obp/ui/#!iso:std:63753:en

- [SOAP 1.2, Part 1] W3C SOAP 1.2, Part 1, Messaging Framework:

  http://www.w3.org/TR/soap12-part1/

- [XML-Schema, Part 1] W3C XML Schema Part 1: Structures Second Edition:

  http://www.w3.org/TR/xmlschema-1/

- [XML-Schema, Part 2] W3C XML Schema Part 2: Datatypes Second Edition:

  http://www.w3.org/TR/xmlschema-2/

- [WS-Security] "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)", OASIS Standard, February 2006.:

  http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

# 3 Terms and Definitions

## 3.1 Conventions

The key words "shall", "shall not", "should", "should not", "may", "need not", "can", "cannot" in this specification are to be interpreted as described in [ISO/IEC Directives Part 2].

## 3.2 Definitions

This section defines terms that are specific to the ONVIF Uplink Service and tests. For the list of applicable general terms and definitions, please see [ONVIF Base Test].

| | |
|---|---|
| **Uplink** | Doing something in advance to prepare for something else. |
| **Video Source** | Entity defined by [ONVIF Device I/O Specification] |
| **Video Source Token** | Token referencing a Device I/O Video Source |

## 3.3 Abbreviations

This section describes abbreviations used in this document.

| | |
|---|---|
| **HTTP** | Hyper Text Transport Protocol. |
| **AAC** | Advanced Audio Coding. |
| **URI** | Uniform Resource Identifier. |
| **WSDL** | Web Services Description Language. |
| **XML** | eXtensible Markup Language. |
| **JPEG** | Joint Photographic Experts Group. |
| **TTL** | Time To Live. |

# 4 Test Overview

This section provides information about the test setup procedure and required prerequisites, and the test policies that should be followed for test case execution.

## 4.1 Test Setup

### 4.1.1 Network Configuration for DUT

The generic test configuration for the execution of test cases defined in this document is as shown below (Figure 1).

**Figure 4.1. Test Configuration for DUT**

**DUT:** ONVIF device to be tested. Hereafter, this is referred to as DUT (Device Under Test).

**ONVIF Client (Test Tool):** Tests are executed by this system and it controls the behavior of the DUT. It handles both expected and unexpected behavior.

**Service Endpoint:** Cloud connection endpoint according to the ONVIF Uplink Specification to which the DUT can connect.

## 4.2 Prerequisites

The pre-requisites for executing the test cases described in this Test Specification are:

1. The DUT shall be configured with an IPv4 address.
2. The DUT shall be IP reachable [in the test configuration].
3. The DUT shall be able to be discovered by the Test Tool.
4. The DUT shall be configured with the time, i.e. manual configuration of UTC time and if NTP is supported by the DUT then NTP time shall be synchronized with NTP Server.
5. The DUT time and Test tool time shall be synchronized with each other either manually or by a common NTP server.

## 4.3 Test Policy

This section describes the test policies specific to the test case execution of each functional block.

The DUT shall adhere to the test policies defined in this section.

### 4.3.1 Uplink Connection

The test policies specific to the test case execution of all functional blocks:

- DUT shall give the Uplink Service entry point by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- DUT shall support the following commands:
  - GetUplinks
  - SetUplink
  - DeleteUplink
  - GetDeviceInformation (Device Mgmt Service)
  - GetUsers (Device Mgmt Service)
- A DUT supporting the Media2 service shall support the following commands:
  - GetProfiles (Media2)
  - GetStreamUri (Media2)

Please refer to Section 5.1 for Uplink Test Cases.

### 4.3.2 Authentication and Authorization

The test policies specific to the test case execution of Authentication and Authorization functional block:

- DUT shall give the ONVIF Uplink Service entry point by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- DUT shall give the ONVIF Security Configuration Service entry point by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- DUT shall support Uplink over WebSocket connection. Otherwise, these test cases will be skipped.
- If DUT supports access token authentication, it shall support Authorization Server Configuration. Otherwise, these test cases will be failed.

- The DUT shall support certification path validation policy.
- The following tests are performed about Authentication and Authorization
  - The DUT will not establish connection if uplink client certificate does not pass certification path validation policy configured at uplink connection.
  - The DUT will not require additional authentication for requests over uplink connection.
  - If DUT supports mTLS authentication:
    - The DUT will establish connection with mTLS authentication configured.
  - If DUT supports access token authentication:
    - The DUT will establish connection with access token authentication configured with the following configurations:
      - OAuthClientCredentials, client_secret_basic (if this combination is supported)
      - OAuthClientCredentials, private_key_jwt (if this combination is supported)
    - The DUT will re-use access token if it was not expired.
    - The DUT will renew access token if it was rejected by the server.
    - The DUT will not establish connection if authentication server does not pass certification path validation policy configured at authentication server configuration.

Please, refer to Section 5.3 for Authentication and Authorization Test Cases.

### 4.3.3 Capabilities

The test policies specific to the test case execution of Capabilities functional block:

- DUT shall give the Uplink Service entry point by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- DUT shall support the following commands:
  - GetServices
  - GetServiceCapabilities
- The following tests are performed
  - Getting capabilities with GetServiceCapabilities command

- Getting capabilities with GetServices command

Please refer to Section 5.4 for Uplink Test Cases.

# 5 Uplink Test Cases

## 5.1 Uplink Connection

### 5.1.1 UPLINK CONNECT AND DISCONNECT

**Test Case ID:** UPLINK-1-1-1

**Specification Coverage:** SetUplink (Uplink Specification), Connection Establishment (Uplink Specification), Connection Management (Uplink Specification), Authentication (Uplink Specification)

**Feature Under Test:** Connect, Disconnect

**WSDL Reference:** uplink.wsdl

**Test Purpose:** To verify conect and disconnect operations.

**Pre-Requisite:** Uplink Service is received from the DUT. Security Configuration Service is received from the DUT. Create self-signed certificate by the DUT as indicated by the SelfSignedCertificateCreationWithRSA or PCKS#10 supported by the DUT as indicated by the PKCS10ExternalCertificationWithRSAcapability or certificate along with an RSA private key in a PKCS#12 data structure upload is supported by the DUT as indicated by the PKCS12CertificateWithRSAPrivateKeyUpload capability.

**Test Configuration:** ONVIF Client and DUT

**Test Procedure:**

1. Start an ONVIF Client.

2. Start the DUT.

3. ONVIF Client removes any existing uplink configurations by following the procedure mentioned in Annex A.1.

4. ONVIF Client configures certificate on a DUT that will be used for client authentication by following the procedure mentioned in Annex A.2

   - out *certificateDUT* - Certificate that uploaded to the DUT with private key.

   - out *certId* - Certificate Id for the certificatethat uploaded to the DUT with private key.

5. ONVIF Client starts service endpoint.

6. ONVIF Client invokes **SetUplink** request with parameters
    - RemoteAddress := IPv4 address and port of service endpoint
    - CertificateID := *certId*
    - UserLevel := "Administrator"
    - Status skipped
    - CertPathValidationPolicyID skipped
7. The DUT responds with **SetUplinkResponse** message.
8. ONVIF Client awaits device connecting to service endpoint by following the procedure mentioned in Annex A.19
    - in *certificate* - Client Certificate for TLS authentification.
9. ONVIF Client invokes **GetDeviceInformation** request via the service endpoint.
10. ONVIF Client responds with **GetDeviceInformationResponse** message via the service endpoint.
11. ONVIF Client removes any existing uplink configurations by following the procedure mentioned in Annex A.1.
12. ONVIF Client verifies that the device disconnects from the service endpoint.
13. ONVIF Client restores DUT configuration if requered.

**Test Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **SetUplinkResponse** message.
- DUT did not connect to the service endpoint.
- DUT did not send **GetDeviceInformationResponse** message.

## 5.2 Uplink Streaming

### 5.2.1 UPLINK CONNECT AND STREAM – H.264 (RTP-Unicast/RTSP/WebSockets)

**Test Case ID:** UPLINK-2-1-1

**Specification Coverage:** RTSP over WebSockets (Uplink Specification), WebSocket transport for RTP/RTSP/TCP (ONVIF Streaming Specification).

**Feature Under Test:** Uplink Streaming over WebSocket

**WSDL Reference:** uplink.wsdl

**Test Purpose:** To verify live WebSockets video streaming over the uplink operation.

**Pre-Requisite:** Uplink Service is received from the DUT. Media2 Service is received from the DUT. Media2 Service is received from the DUT. H.264 encoding is supported by DUT. Real-time streaming is supported by DUT. WebSockets is supported by DUT.

**Test Configuration:** ONVIF Client and DUT

**Test Procedure:**

1. Start an ONVIF Client.

2. Start the DUT.

3. ONVIF Client configures a media profile and retrieves a stream uri for video streaming by following the procedure mentioned in Annex A.20 with the following input and output parameters

   - in H264 - required video encoding
   - in RTSP - Transport Protocol
   - in IPv4 - IP version
   - out *streamUri* - Uri for media streaming

4. ONVIF Client set up uplink connection between the DUT and uplink service endpoint of the ONVIF Client by following the procedure mentioned in Annex A.26.

5. Uplink service endpoint of the ONVIF Client tries to start and decode media streaming over WebSocket by following the procedure mentioned in Annex A.27 with the following input and output parameters

  - in *streamUri* - Uri for media streaming
  - in video - media type
  - in H.264 - expected media stream encoding

6. ONVIF Client restores settings of Video Encoder Configuration and Media Profile changed at step [TODO] .

**Test Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send any of the required response messages.
- DUT did not connect to the service endpoint.
- DUT did not process the RTSP setup.
- DUT did not send H.264 Video.

## 5.3 Authentication and Authorization

### 5.3.1 Uplink over WebSocket with mTLS Authentication

**Test Case ID:** UPLINK-3-1-1

**Specification Coverage:** mTLS authentication (ONVIF Uplink Specification)

**Feature Under Test:** Uplink WebSocket Connection using mTLS authentication.

**WSDL Reference:** uplink.wsdl, advancedsecurity.wsdl

**Test Purpose:** To verify that DUT can establish connection to ONVIF Client via Uplink over WebSocket using mTLS authentication. To verify that DUT do not require additional authentication inside established Uplink connection.

**Pre-Requisite:** Security Configuration Service is received from the DUT. Uplink Service is received from the DUT. mTLS authentication is supported by the DUT as indicated by the AuthorizationModes = mTLS capability. Uplink WebSocket connection is supported by the DUT as indicated by the Protocols = wss capability.

**Test Configuration:** ONVIF Client and DUT

**Test Procedure:**

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client gets the security configuration service capabilities by following the procedure mentioned in Annex A.34 with the following input and output parameters:
   - out *cap* - Security Configuration Service Capabilities
4. Set:
   - *keyAlgorithm* := **"ECC"** if *cap*.KeystoreCapabilities.ECCKeyPairGeneration = true; **"RSA"** otherwise.
5. ONVIF Client configures uplink connection using the following steps:

   5.1. ONVIF Client creates certification path validation policy by following the procedure mentioned in Annex A.35 with the following input and output parameters

   - in *cap* - DUT capabilities
   - in *keyAlgorithm* - DUT capabilities
   - in "CN=ONVIF TT Uplink 1,C=US" - CA certificate subject
   - in "Test CertPathValidationPolicy Uplink Alias" - certification path validation policy alias
   - out *certPathValidationPolicyIDUplink* - certification path validation policy identifier
   - out *certIDUplink* - certificate identifier
   - out *keyIDUplink* - key pair identifier
   - out *CACertUplink* - CA certificate
   - out *privateKeyCACertUplink* - CA certificate private key

   5.2. ONVIF Client creates a certificate signed by private key of the CA-certificate with subject and a corresponding public key in the certificate along with the corresponding private key by following the procedure described in Annex A.37 with the following input and output parameters:

   - in *cap* - DUT capabilities
   - in "CN=ONVIF TT Uplink 2,C=US" - certificate subject

- in *CACertUplink* - CA-certificate
- in *privateKeyCACertUplink* - private key of CA-certificate for certificate signature
- out *certUplink* - certificate
- out *publicKeyUplink* - public key of certificate
- out *privateKeyUplink* - private key of certificate

5.3. ONVIF Client creates a CA certificate and a corresponding key pair by following the procedure described in Annex A.7 with the following input and output parameters:

- in *cap* - DUT capabilities
- in *keyAlgorithm* - key pair algorithm
- out *CAcert* - CA certificate
- out *CAkeyPair* - key pair with public and private keys

5.4. ONVIF Client creates and uploads a CA-signed certificate for key pair and associated CA certificate and a corresponding by following the procedure described in Annex A.16 with the following input and output parameters:

- in *cap* - DUT capabilities
- in *CAcert* - CA certificate
- in *CAkeyPair*.privateKey - CA certificate private key
- out *certID1* - CA-signed certificate identificator
- out *keyID1* - key pair
- out *cert1* - CA-signed certificate

5.5. ONVIF Client creates and uploads a CA-signed certificate for key pair and associated CA certificate and a corresponding by following the procedure described in Annex A.16 with the following input and output parameters:

- in *cap* - DUT capabilities
- in *CAcert* - CA certificate
- in *CAkeyPair*.privateKey - CA certificate private key

- out *certID2* - CA-signed certificate identificator
- out *keyID2* - key pair
- out *cert2* - CA-signed certificate

5.6. ONVIF Client configure endpoint for WebSocket uplink connection at address *wssUplinkAddress1* using *certUplink* certificate with the following certificates accepted from the client:

- *cert1*
- *cert2*

5.7. ONVIF Client invokes **SetUplink** request with parameters

- RemoteAddress := *wssUplinkAddress1*
- CertificateID := *certID1*
- UserLevel := **Administrator**
- Status is skipped
- CertPathValidationPolicyID := *certPathValidationPolicyIDUplink*
- AuthorizationServer is skipped
- Error is skipped

5.8. The DUT responds with **SetUplinkResponse** message.

6. ONVIF Client verifies initial connection establishment:

6.1. ONVIF Client awaits device connecting to *wssUplinkAddress1* endpoint.

6.2. DUT opens connection to *wssUplinkAddress1* with *cert1* used for client authentication.

6.3. DUT verifies certificate provided by ONVIF Client at uplink connection establishment based on *certPathValidationPolicyIDUplink* certification path validation policy.

6.4. ONVIF Client verify certificate received from the DUT. If received certificate is not *cert1*, FAIL the test, restore the DUT state, and skip other steps.

7. ONVIF Client verifies connection establishment after connection parameters were changed:

7.1. ONVIF Client invokes **SetUplink** request over uplink connection with parameters without any additional authentication with parameters

- RemoteAddress := *wssUplinkAddress1*
- CertificateID := *certID2*
- UserLevel := **Administrator**
- Status is skipped
- CertPathValidationPolicyID := *certPathValidationPolicyIDUplink*
- AuthorizationServer is skipped
- Error is skipped

7.2. The DUT responds with **SetUplinkResponse** message.

7.3. ONVIF Client close device connection to *wssUplinkAddress1* endpoint.

7.4. ONVIF Client awaits device connecting to *wssUplinkAddress1* endpoint.

7.5. DUT opens connection to *wssUplinkAddress1* with *cert2* used for client authentication.

7.6. DUT verifies certificate provided by ONVIF Client at uplink connection establishment based on *certPathValidationPolicyIDUplink* certification path validation policy.

7.7. ONVIF Client verify certificate received from the DUT. If received certificate is not *cert2*, FAIL the test, restore the DUT state, and skip other steps.

8. ONVIF Client restores the DUT state.

**Test Result:**

**PASS –**

- DUT passed all assertions.

**FAIL –**

- DUT did not send **SetUplinkResponse** message.
- DUT does not establish WebSocket Uplink connection at step 6.2.
- DUT requests additional authentication at steps 7.1 and 7.2.
- DUT does not establish WebSocket Uplink connection at step 7.5.

### 5.3.2 Uplink over WebSocket with access token authentication (OAuthClientCredentials, client_secret_basic)

**Test Case ID:** UPLINK-3-1-2

**Specification Coverage:** Device authentication and authorization (ONVIF Security Service Specification), Access token authentication (ONVIF Uplink Specification)

**Feature Under Test:** Uplink Connection using access token authentication using OAuthClientCredentials and client_secret_basic settings with authentication server certificate validation.

**WSDL Reference:** uplink.wsdl, advancedsecurity.wsdl

**Test Purpose:** To verify that DUT can receive access token from the authentication server using OAuthClientCredentials authentication method with client_secret_basic type. To verify that DUT can establish connection to ONVIF Client via Uplink using access token. To verify that DUT will reuse token if it is not expired. To verify that DUT will renew token when authentication parameters will be changed. To verify that DUT do not require additional authentication inside established Uplink connection.

**Pre-Requisite:** Security Configuration Service is received from the DUT. Uplink Service is received from the DUT. Authorization Server Configuration is supported by the DUT as indicated by the AuthorizationServer.MaxConfigurations capability. OAuthClientCredentials authentication method is supported by the DUT as indicated by the AuthorizationServer.ConfigurationTypesSupported capability. client_secret_basic authentication is supported by the DUT as indicated by the AuthorizationServer.ClientAuthenticationMethodsSupported capability. Access token authentication is supported by the DUT as indicated by the AuthorizationModes = AccessToken capability.

**Test Configuration:** ONVIF Client and DUT

**Test Procedure:**

1. Start an ONVIF Client.

2. Start the DUT.

3. ONVIF Client gets the security configuration service capabilities by following the procedure mentioned in Annex A.34 with the following input and output parameters:

   - out *cap* - Security Configuration Service Capabilities

4. ONVIF Client deletes one authorization server configuration if maximum is reached by following the procedure described in Annex A.38 with the following input and output parameters:

   - in *cap* - DUT capabilities
   - out *itemToRestore1* - deleted authorization server configuration if any

5. Set:

   - *keyAlgorithm* := **"ECC"** if *cap*.KeystoreCapabilities.ECCKeyPairGeneration = true; **"RSA"** otherwise.

6. ONVIF Client configures authentication server connection using the following steps:

   6.1. ONVIF Client creates certification path validation policy by following the procedure mentioned in Annex A.35 with the following input and output parameters

   - in *cap* - DUT capabilities
   - in *keyAlgorithm* - DUT capabilities
   - in "CN=ONVIF TT AuthServer 1,C=US" - CA certificate subject
   - in "Test CertPathValidationPolicy AuthServer Alias" - certification path validation policy alias
   - out *certPathValidationPolicyIDAuthServer* - certification path validation policy identifier
   - out *certIDAuthServer* - certificate identifier
   - out *keyIDAuthServer* - key pair identifier
   - out *CACertAuthServer* - CA certificate
   - out *privateKeyCACertAuthServer* - CA certificate private key

   6.2. ONVIF Client creates a certificate signed by private key of the CA-certificate with subject and a corresponding public key in the certificate along with the corresponding private key by following the procedure described in Annex A.37 with the following input and output parameters:

   - in *cap* - DUT capabilities
   - in "CN=ONVIF TT AuthServer 2,C=US" - certificate subject

- in *CACertAuthServer* - CA-certificate
- in *privateKeyCACertAuthServer* - private key of CA-certificate for certificate signature
- out *certAuthServer* - certificate
- out *publicKeyAuthServer* - public key of certificate
- out *privateKeyAuthServer* - private key of certificate

6.3. ONVIF Client starts Authorization server with metadata endpoint *authServerMetadataEndpoint1* conforming to RFC8414 with following settings:

- It is configured to *certAuthServer* as a server certificate.
- It is configured to accept OAuth2 client credentials grant flow per [RFC 6749] authentication method only.
- It is configured to accept client_secret_basic authentication method only.
- Client with client identifier *clientID1* with client secret *clientSecret1* and scope *scope1* is added to be authorized.
- Client with client identifier *clientID1* with client secret *clientSecret1* and scope *scope2* is added to be authorized.

6.4. ONVIF Client invokes **CreateAuthorizationServerConfiguration** request with parameters

- Type := **"OAuthClientCredentials"**
- ClientAuth := **"client_secret_basic"**
- ServerUri := *authServerMetadataEndpoint1*
- ClientID := *clientID1*
- ClientSecret := *clientSecret1*
- Scope := *scope1*
- KeyID is skipped
- CertificateID is skipped
- CertPathValidationPolicyID := *certPathValidationPolicyIDAuthServer*

6.5. The DUT responds with **CreateAuthorizationServerConfigurationResponse** message with parameters

- Token =: *authServerToken1*

7. ONVIF Client configures uplink connection using the following steps:

7.1. ONVIF Client creates certification path validation policy by following the procedure mentioned in Annex A.35 with the following input and output parameters

- in *cap* - DUT capabilities
- in *keyAlgorithm* - DUT capabilities
- in "CN=ONVIF TT Uplink 1,C=US" - CA certificate subject
- in "Test CertPathValidationPolicy Uplink Alias" - certification path validation policy alias
- out *certPathValidationPolicyIDUplink* - certification path validation policy identifier
- out *certIDUplink* - certificate identifier
- out *keyIDUplink* - key pair identifier
- out *CACertUplink* - CA certificate
- out *privateKeyCACertUplink* - CA certificate private key

7.2. ONVIF Client creates a certificate signed by private key of the CA-certificate with subject and a corresponding public key in the certificate along with the corresponding private key by following the procedure described in Annex A.37 with the following input and output parameters:

- in *cap* - DUT capabilities
- in "CN=ONVIF TT Uplink 2,C=US" - certificate subject
- in *CACertUplink* - CA-certificate
- in *privateKeyCACertUplink* - private key of CA-certificate for certificate signature
- out *certUplink* - certificate
- out *publicKeyUplink* - public key of certificate
- out *privateKeyUplink* - private key of certificate

7.3. ONVIF Client configure and start service endpoint for uplink connection at address *uplinkAddress1* using *certUplink* certificate.

7.4. ONVIF Client invokes **SetUplink** request with parameters

- RemoteAddress := *uplinkAddress1*
- CertificateID is skipped
- UserLevel := **Administrator**
- Status is skipped
- CertPathValidationPolicyID := *certPathValidationPolicyIDUplink*
- AuthorizationServer := *authServerToken1*
- Error is skipped

7.5. The DUT responds with **SetUplinkResponse** message.

8. ONVIF Client verifies initial connection establishment:

8.1. The DUT opens connection to *authServerMetadataEndpoint1* with client identifier *clientID1* and client secret *clientSecret1* using client_secret_basic method authentication and OAuth2 client credentials grant flow.

8.2. The DUT verifies certificate provided by authentication server based on *certPathValidationPolicyIDAuthServer* certification path validation policy.

8.3. The DUT receives access token *accessToken1* from authorization server for scope *scope1*.

8.4. ONVIF Client awaits device connecting to *uplinkAddress1* endpoint.

8.5. DUT opens connection to *uplinkAddress1* with *accessToken1*.

8.6. DUT verifies certificate provided by ONVIF Client at uplink connection establishment based on *certPathValidationPolicyIDUplink* certification path validation policy.

8.7. ONVIF Client verify access token received from the DUT with Authentication Server. If received access token *accessToken1* is not valid for *scope1*, FAIL the test, restore the DUT state, and skip other steps.

9. ONVIF Client verifies that access token will be reused if it is not expired:

9.1. ONVIF Client close device connection to *uplinkAddress1* endpoint.

9.2. ONVIF Client awaits device connecting to *uplinkAddress1* endpoint.

9.3. DUT opens connection to *uplinkAddress1* with *accessToken1*.

9.4. DUT verifies certificate provided by ONVIF Client at uplink connection establishment based on *certPathValidationPolicyIDUplink* certification path validation policy.

9.5. ONVIF Client verify access token received from the DUT with Authentication Server. If received access token *accessToken1* is not the same as was used at step 8.5, FAIL the test, restore the DUT state, and skip other steps.

10. ONVIF Client verifies connection establishment after connection parameters were changed:

10.1. ONVIF Client invokes **SetAuthorizationServerConfiguration** request using uplink connection with parameters without any additional authentication

- @token := *authServerToken1*
- Data.Type := **"OAuthClientCredentials"**
- Data.ClientAuth := **"client_secret_basic"**
- Data.ServerUri := *authServerMetadataEndpoint1*
- Data.ClientID := *clientID1*
- Data.ClientSecret := *clientSecret1*
- Data.Scope := *scope2*
- Data.KeyID is skipped
- Data.CertificateID is skipped
- Data.CertPathValidationPolicyID := *certPathValidationPolicyIDAuthServer*

10.2. The DUT responds with **SetAuthorizationServerConfigurationResponse** message.

10.3. ONVIF Client close device connection to *uplinkAddress1* endpoint.

10.4. The DUT opens connection to *authServerMetadataEndpoint1* with client identifier *clientID1* and client secret *clientSecret1* using client_secret_basic method authentication and OAuth2 client credentials grant flow.

10.5. The DUT receives access token *accessToken2* from authorization server for scope *scope2*.

10.6. ONVIF Client awaits device connecting to *uplinkAddress1* endpoint.

10.7. DUT opens connection to *uplinkAddress1* with *accessToken2*.

10.8. DUT verifies certificate provided by ONVIF Client at uplink connection establishment based on *certPathValidationPolicyIDUplink* certification path validation policy.

10.9. ONVIF Client verify access token received from the DUT with Authentication Server. If received access token *accessToken2* is not valid for *scope2*, FAIL the test, restore the DUT state, and skip other steps.

11. ONVIF Client restores the DUT state.

**Test Result:**

**PASS –**

- DUT passed all assertions.

**FAIL –**

- DUT did not send **CreateAuthorizationServerConfigurationResponse** message.
- DUT did not send **SetUplinkResponse** message.
- DUT did not send **SetAuthorizationServerConfigurationResponse** message.
- DUT does not establish connection at step 8.5.
- DUT does not establish connection at step 9.3.
- DUT requests additional authentication at steps 10.1 and 10.2.
- DUT does not establish connection at step 10.7.

### 5.3.3 Uplink over WebSocket with access token authentication (OAuthClientCredentials, private_key_jwt)

**Test Case ID:** UPLINK-3-1-3

**Specification Coverage:** Device authentication and authorization (ONVIF Security Service Specification), Access token authentication (ONVIF Uplink Specification)

**Feature Under Test:** Uplink Connection using access token authentication using OAuthClientCredentials and private_key_jwt settings with authentication server certificate validation.

**WSDL Reference:** uplink.wsdl, advancedsecurity.wsdl

**Test Purpose:** To verify that DUT can receive access token from the authentication server using OAuthClientCredentials authentication method with private_key_jwt type. To verify that DUT can establish connection to ONVIF Client via Uplink using access token. To verify that DUT do not require additional authentication inside established Uplink connection.

**Pre-Requisite:** Security Configuration Service is received from the DUT. Uplink Service is received from the DUT. Authorization Server Configuration is supported by the DUT as indicated by the AuthorizationServer.MaxConfigurations capability. OAuthClientCredentials authentication method is supported by the DUT as indicated by the AuthorizationServer.ConfigurationTypesSupported capability. private_key_jwt authentication is supported by the DUT as indicated by the AuthorizationServer.ClientAuthenticationMethodsSupported capability. Access token authentication is supported by the DUT as indicated by the AuthorizationModes = AccessToken capability.

**Test Configuration:** ONVIF Client and DUT

**Test Procedure:**

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client gets the security configuration service capabilities by following the procedure mentioned in Annex A.34 with the following input and output parameters:
    - out *cap* - Security Configuration Service Capabilities
4. ONVIF Client deletes one authorization server configuration if maximum is reached by following the procedure described in Annex A.38 with the following input and output parameters:
    - in *cap* - DUT capabilities
    - out *itemToRestore1* - deleted authorization server configuration if any
5. Set:
    - *keyAlgorithm* := **"ECC"** if *cap*.KeystoreCapabilities.ECCKeyPairGeneration = true; **"RSA"** otherwise.
6. ONVIF Client configures authentication server connection using the following steps:

    6.1. ONVIF Client creates a key pair and get public key from device by following the procedure described in Annex A.40 with the following input and output parameters:

    - in *cap* - DUT capabilities

- in *keyAlgorithm* - CSR key pair algorithm
- out *keyID1Auth1* - key pair
- out *publicKeyAuth1* - public key

6.2. ONVIF Client creates certification path validation policy by following the procedure mentioned in Annex A.35 with the following input and output parameters

- in *cap* - DUT capabilities
- in *keyAlgorithm* - CSR key pair algorithm
- in "CN=ONVIF TT AuthServer 1,C=US" - CA certificate subject
- in "Test CertPathValidationPolicy AuthServer Alias" - certification path validation policy alias
- out *certPathValidationPolicyIDAuthServer* - certification path validation policy identifier
- out *certIDAuthServer* - certificate identifier
- out *keyIDAuthServer* - key pair identifier
- out *CACertAuthServer* - CA certificate
- out *privateKeyCACertAuthServer* - CA certificate private key

6.3. ONVIF Client creates a certificate signed by private key of the CA-certificate with subject and a corresponding public key in the certificate along with the corresponding private key by following the procedure described in Annex A.37 with the following input and output parameters:

- in *cap* - DUT capabilities
- in "CN=ONVIF TT AuthServer 2,C=US" - certificate subject
- in *CACertAuthServer* - CA-certificate
- in *privateKeyCACertAuthServer* - private key of CA-certificate for certificate signature
- out *certAuthServer* - certificate
- out *publicKeyAuthServer* - public key of certificate

- out *privateKeyAuthServer* - private key of certificate

6.4. ONVIF Client starts Authorization server with metadata endpoint *authServerMetadataEndpoint1* conforming to RFC8414 with following settings:

- It is configured to *certAuthServer* as a server certificate.
- It is configured to accept OAuth2 client credentials grant flow per [RFC 6749] authentication method only.
- It is configured to accept private_key_jwt authentication method only.
- Client with client identifier *clientID1* with client secret *clientSecret1*, *publicKeyAuth1* as key, and scope *scope1* is added to be authorized.
- Client with client identifier *clientID1* with client secret *clientSecret1*, *publicKeyAuth1* as key, and scope *scope2* is added to be authorized.

6.5. ONVIF Client invokes **CreateAuthorizationServerConfiguration** request with parameters

- Type := **"OAuthClientCredentials"**
- ClientAuth := **"private_key_jwt"**
- ServerUri := *authServerMetadataEndpoint1*
- ClientID := *clientID1*
- ClientSecret skipped
- Scope := *scope1*
- KeyID := *publicKeyAuth1*
- CertificateID is skipped
- CertPathValidationPolicyID := *certPathValidationPolicyIDAuthServer*

6.6. The DUT responds with **CreateAuthorizationServerConfigurationResponse** message with parameters

- Token =: *authServerToken1*

7. ONVIF Client configures uplink connection using the following steps:

7.1. ONVIF Client creates certification path validation policy by following the procedure mentioned in Annex A.35 with the following input and output parameters

- in *cap* - DUT capabilities
- in *keyAlgorithm* - DUT capabilities
- in "CN=ONVIF TT Uplink 1,C=US" - CA certificate subject
- in "Test CertPathValidationPolicy Uplink Alias" - certification path validation policy alias
- out *certPathValidationPolicyIDUplink* - certification path validation policy identifier
- out *certIDUplink* - certificate identifier
- out *keyIDUplink* - key pair identifier
- out *CACertUplink* - CA certificate
- out *privateKeyCACertUplink* - CA certificate private key

7.2. ONVIF Client creates a certificate signed by private key of the CA-certificate with subject and a corresponding public key in the certificate along with the corresponding private key by following the procedure described in Annex A.37 with the following input and output parameters:

- in *cap* - DUT capabilities
- in "CN=ONVIF TT Uplink 2,C=US" - certificate subject
- in *CACertUplink* - CA-certificate
- in *privateKeyCACertUplink* - private key of CA-certificate for certificate signature
- out *certUplink* - certificate
- out *publicKeyUplink* - public key of certificate
- out *privateKeyUplink* - private key of certificate

7.3. ONVIF Client configure and start service endpoint for uplink connection at address *uplinkAddress1* using *certUplink* certificate.

7.4. ONVIF Client invokes **SetUplink** request with parameters

- RemoteAddress := *uplinkAddress1*

- CertificateID is skipped
- UserLevel := **Administrator**
- Status is skipped
- CertPathValidationPolicyID := *certPathValidationPolicyIDUplink*
- AuthorizationServer := *authServerToken1*
- Error is skipped

7.5. The DUT responds with **SetUplinkResponse** message.

8. ONVIF Client verifies initial connection establishment:

8.1. The DUT opens connection to *authServerMetadataEndpoint1* with client identifier *clientID1* and client secret *clientSecret1* using client_secret_basic method authentication and OAuth2 client credentials grant flow.

8.2. The DUT verifies certificate provided by authentication server based on *certPathValidationPolicyIDAuthServer* certification path validation policy.

8.3. The DUT receives access token *accessToken1* from authorization server for scope *scope1*.

8.4. ONVIF Client awaits device connecting to *uplinkAddress1* endpoint.

8.5. DUT opens connection to *uplinkAddress1* with *accessToken1*.

8.6. DUT verifies certificate provided by ONVIF Client at uplink connection establishment based on *certPathValidationPolicyIDUplink* certification path validation policy.

8.7. ONVIF Client verify access token received from the DUT with Authentication Server. If received access token *accessToken1* is not valid for *scope1*, FAIL the test, restore the DUT state, and skip other steps.

8.8. ONVIF Client invokes **SetAuthorizationServerConfiguration** request using uplink connection with parameters without any additional authentication

- @token := *authServerToken1*
- Data.Type := **"OAuthClientCredentials"**
- Data.ClientAuth := **"client_secret_basic"**

- Data.ServerUri := *authServerMetadataEndpoint1*
- Data.ClientID := *clientID1*
- Data.ClientSecret := *clientSecret1*
- Data.Scope := *scope2*
- Data.KeyID is skipped
- Data.CertificateID is skipped
- Data.CertPathValidationPolicyID := *certPathValidationPolicyIDAuthServer*

8.9. The DUT responds with **SetAuthorizationServerConfigurationResponse** message.

9. ONVIF Client restores the DUT state.

**Test Result:**

**PASS –**

- DUT passed all assertions.

**FAIL –**

- DUT did not send **CreateAuthorizationServerConfigurationResponse** message.
- DUT did not send **SetUplinkResponse** message.
- DUT did not send **SetAuthorizationServerConfigurationResponse** message.
- DUT does not establish connection at step 8.5.

### 5.3.4 Uplink over WebSocket with mTLS Authentication - Invalid Server Certificate

**Test Case ID:** UPLINK-3-1-4

**Specification Coverage:** mTLS authentication (ONVIF Uplink Specification)

**Feature Under Test:** Uplink Connection using mTLS authentication.

**WSDL Reference:** uplink.wsdl, advancedsecurity.wsdl

**Test Purpose:** To verify that DUT will not establish connection with invalid server certificate.

**Pre-Requisite:** Security Configuration Service is received from the DUT. Uplink Service is received from the DUT. mTLS authentication is supported by the DUT as indicated by the AuthorizationModes = mTLS capability.

**Test Configuration:** ONVIF Client and DUT

**Test Procedure:**

1. Start an ONVIF Client.

2. Start the DUT.

3. ONVIF Client gets the security configuration service capabilities by following the procedure mentioned in Annex A.34 with the following input and output parameters:

   - out *cap* - Security Configuration Service Capabilities

4. Set:

   - *keyAlgorithm* := **"ECC"** if *cap*.KeystoreCapabilities.ECCKeyPairGeneration = true; **"RSA"** otherwise.

5. ONVIF Client configures uplink connection using the following steps:

   5.1. ONVIF Client creates certification path validation policy by following the procedure mentioned in Annex A.35 with the following input and output parameters

   - in *cap* - DUT capabilities
   - in *keyAlgorithm* - DUT capabilities
   - in "CN=ONVIF TT Uplink 1,C=US" - CA certificate subject
   - in "Test CertPathValidationPolicy Uplink Alias" - certification path validation policy alias
   - out *certPathValidationPolicyIDUplink* - certification path validation policy identifier
   - out *certIDUplink* - certificate identifier
   - out *keyIDUplink* - key pair identifier
   - out *CACertUplink* - CA certificate
   - out *privateKeyCACertUplink* - CA certificate private key

   5.2. ONVIF Client creates a CA certificate and a corresponding key pair by following the procedure described in Annex A.7 with the following input and output parameters:

- in *cap* - DUT capabilities
- in *keyAlgorithm* - key pair algorithm
- out *CAcert* - CA certificate
- out *CAkeyPair* - key pair with public and private keys

5.3. ONVIF Client creates and uploads a CA-signed certificate for key pair and associated CA certificate and a corresponding by following the procedure described in Annex A.16 with the following input and output parameters:

- in *cap* - DUT capabilities
- in *CAcert* - CA certificate
- in *CAkeyPair*.privateKey - CA certificate private key
- out *certID1* - CA-signed certificate identificator
- out *keyID1* - key pair
- out *cert1* - CA-signed certificate

5.4. ONVIF Client configure endpoint for uplink connection at address *uplinkAddress1* using any certificate that will not pass *certPathValidationPolicyIDUplink* with the following certificates accepted from the client:

- *cert1*

5.5. ONVIF Client invokes **SetUplink** request with parameters

- RemoteAddress := *uplinkAddress1*
- CertificateID := *certID1*
- UserLevel := **Administrator**
- Status is skipped
- CertPathValidationPolicyID := *certPathValidationPolicyIDUplink*
- AuthorizationServer is skipped
- Error is skipped

5.6. The DUT responds with **SetUplinkResponse** message.

6. ONVIF Client verifies connection will not be establishment:

   6.1. ONVIF Client awaits device connecting to *uplinkAddress1* endpoint.

   6.2. DUT tries to open connection to *uplinkAddress1* with *cert1* used for client authentication.

   6.3. DUT verifies certificate provided by ONVIF Client at uplink connection establishment based on *certPathValidationPolicyIDUplink* certification path validation policy and prevent connection.

7. ONVIF Client restores the DUT state.

**Test Result:**

**PASS –**

- DUT passed all assertions.

**FAIL –**

- DUT did not send **SetUplinkResponse** message.
- DUT establishes connection at step 6.3 during *operationDelay* timeout.

**Note:** *operationDelay* will be taken from Operation Delay field of ONVIF Device Test Tool.

### 5.3.5 Uplink over WebSocket with access token authentication (OAuthClientCredentials, client_secret_basic) - Invalid Uplink Client Certificate

**Test Case ID:** UPLINK-3-1-5

**Specification Coverage:** Device authentication and authorization (ONVIF Security Service Specification), Access token authentication (ONVIF Uplink Specification)

**Feature Under Test:** Uplink Connection using access token authentication when token received from authentication server using OAuthClientCredentials and client_secret_basic settings with authentication server certificate validation.

**WSDL Reference:** uplink.wsdl, advancedsecurity.wsdl

**Test Purpose:** To verify that DUT will not establish connection with uplink client which has invalid certificate.

**Pre-Requisite:** Security Configuration Service is received from the DUT. Uplink Service is received from the DUT. Authorization Server Configuration is supported by the DUT as indicated by the

AuthorizationServer.MaxConfigurations capability. OAuthClientCredentials authentication method is supported by the DUT as indicated by the AuthorizationServer.ConfigurationTypesSupported capability. client_secret_basic authentication is supported by the DUT as indicated by the AuthorizationServer.ClientAuthenticationMethodsSupported capability. Access token authentication is supported by the DUT as indicated by the AuthorizationModes = AccessToken capability.

**Test Configuration:** ONVIF Client and DUT

**Test Procedure:**

1. Start an ONVIF Client.

2. Start the DUT.

3. ONVIF Client gets the security configuration service capabilities by following the procedure mentioned in Annex A.34 with the following input and output parameters:

   - out *cap* - Security Configuration Service Capabilities

4. ONVIF Client deletes one authorization server configuration if maximum is reached by following the procedure described in Annex A.38 with the following input and output parameters:

   - in *cap* - DUT capabilities

   - out *itemToRestore1* - deleted authorization server configuration if any

5. Set:

   - *keyAlgorithm* := **"ECC"** if *cap*.KeystoreCapabilities.ECCKeyPairGeneration = true; **"RSA"** otherwise.

6. ONVIF Client configures authentication server connection using the following steps:

   6.1. ONVIF Client creates certification path validation policy by following the procedure mentioned in Annex A.35 with the following input and output parameters

   - in *cap* - DUT capabilities

   - in *keyAlgorithm* - DUT capabilities

   - in "CN=ONVIF TT AuthServer 1,C=US" - CA certificate subject

   - in "Test CertPathValidationPolicy AuthServer Alias" - certification path validation policy alias

- out *certPathValidationPolicyIDAuthServer* - certification path validation policy identifier
- out *certIDAuthServer* - certificate identifier
- out *keyIDAuthServer* - key pair identifier
- out *CACertAuthServer* - CA certificate
- out *privateKeyCACertAuthServer* - CA certificate private key

6.2. ONVIF Client creates a certificate signed by private key of the CA-certificate with subject and a corresponding public key in the certificate along with the corresponding private key by following the procedure described in Annex A.37 with the following input and output parameters:

- in *cap* - DUT capabilities
- in "CN=ONVIF TT AuthServer 2,C=US" - certificate subject
- in *CACertAuthServer* - CA-certificate
- in *privateKeyCACertAuthServer* - private key of CA-certificate for certificate signature
- out *certAuthServer* - certificate
- out *publicKeyAuthServer* - public key of certificate
- out *privateKeyAuthServer* - private key of certificate

6.3. ONVIF Client starts Authorization server with metadata endpoint *authServerMetadataEndpoint1* conforming to RFC8414 with following settings:

- It is configured to *certAuthServer* as a server certificate.
- It is configured to accept OAuth2 client credentials grant flow per [RFC 6749] authentication method only.
- It is configured to accept client_secret_basic authentication method only.
- Client with client identifier *clientID1* with client secret *clientSecret1* and scope *scope1* is added to be authorized.
- Client with client identifier *clientID1* with client secret *clientSecret1* and scope *scope2* is added to be authorized.

6.4. ONVIF Client invokes **CreateAuthorizationServerConfiguration** request with parameters

- Type := **"OAuthClientCredentials"**
- ClientAuth := **"client_secret_basic"**
- ServerUri := *authServerMetadataEndpoint1*
- ClientID := *clientID1*
- ClientSecret := *clientSecret1*
- Scope := *scope1*
- KeyID is skipped
- CertificateID is skipped
- CertPathValidationPolicyID := *certPathValidationPolicyIDAuthServer*

6.5. The DUT responds with **CreateAuthorizationServerConfigurationResponse** message with parameters

- Token =: *authServerToken1*

7. ONVIF Client configures uplink connection using the following steps:

7.1. ONVIF Client creates certification path validation policy by following the procedure mentioned in Annex A.35 with the following input and output parameters

- in *cap* - DUT capabilities
- in *keyAlgorithm* - DUT capabilities
- in "CN=ONVIF TT Uplink 1,C=US" - CA certificate subject
- in "Test CertPathValidationPolicy Uplink Alias" - certification path validation policy alias
- out *certPathValidationPolicyIDUplink* - certification path validation policy identifier
- out *certIDUplink* - certificate identifier
- out *keyIDUplink* - key pair identifier
- out *CACertUplink* - CA certificate

- out *privateKeyCACertUplink* - CA certificate private key

7.2. ONVIF Client configure and start service endpoint for uplink connection at address *uplinkAddress1* using any certificate that will **not** pass *certPathValidationPolicyIDUplink* certification path validation policy.

7.3. ONVIF Client invokes **SetUplink** request with parameters

- RemoteAddress := *uplinkAddress1*
- CertificateID is skipped
- UserLevel := **Administrator**
- Status is skipped
- CertPathValidationPolicyID := *certPathValidationPolicyIDUplink*
- AuthorizationServer := *authServerToken1*
- Error is skipped

7.4. The DUT responds with **SetUplinkResponse** message.

8. ONVIF Client verifies initial connection establishment:

8.1. The DUT opens connection to *authServerMetadataEndpoint1* with client identifier *clientID1* and client secret *clientSecret1* using client_secret_basic method authentication and OAuth2 client credentials grant flow.

8.2. The DUT verifies certificate provided by authentication server based on *certPathValidationPolicyIDAuthServer* certification path validation policy.

8.3. The DUT receives access token *accessToken1* from authorization server for scope *scope1*.

8.4. ONVIF Client awaits device connecting to *uplinkAddress1* endpoint.

8.5. DUT opens connection to *uplinkAddress1* with *accessToken1*.

8.6. DUT verifies certificate provided by ONVIF Client at uplink connection establishment based on *certPathValidationPolicyIDUplink* certification path validation policy and prevent connection.

9. ONVIF Client restores the DUT state.

**Test Result:**

**PASS –**

- DUT passed all assertions.

**FAIL –**

- DUT did not send **CreateAuthorizationServerConfigurationResponse** message.
- DUT did not send **SetUplinkResponse** message.
- DUT establishes connection at step 8.6 during *operationDelay* timeout.

**Note:** *operationDelay* will be taken from Operation Delay field of ONVIF Device Test Tool.

### 5.3.6 Uplink over WebSocket with access token authentication (OAuthClientCredentials, client_secret_basic) - Invalid Authentication Server Certificate

**Test Case ID:** UPLINK-3-1-6

**Specification Coverage:** Device authentication and authorization (ONVIF Security Service Specification), Access token authentication (ONVIF Uplink Specification)

**Feature Under Test:** Uplink Connection using access token authentication when token received from authentication server using OAuthClientCredentials and client_secret_basic settings with authentication server certificate validation.

**WSDL Reference:** uplink.wsdl, advancedsecurity.wsdl

**Test Purpose:** To verify that DUT will not establish connection with invalid authentication server certificate.

**Pre-Requisite:** Security Configuration Service is received from the DUT. Uplink Service is received from the DUT. Authorization Server Configuration is supported by the DUT as indicated by the AuthorizationServer.MaxConfigurations capability. OAuthClientCredentials authentication method is supported by the DUT as indicated by the AuthorizationServer.ConfigurationTypesSupported capability. client_secret_basic authentication is supported by the DUT as indicated by the AuthorizationServer.ClientAuthenticationMethodsSupported capability. Access token authentication is supported by the DUT as indicated by the AuthorizationModes = AccessToken capability.

**Test Configuration:** ONVIF Client and DUT

**Test Procedure:**

1. Start an ONVIF Client.

2. Start the DUT.

3. ONVIF Client gets the security configuration service capabilities by following the procedure mentioned in Annex A.34 with the following input and output parameters:

   - out *cap* - Security Configuration Service Capabilities

4. ONVIF Client deletes one authorization server configuration if maximum is reached by following the procedure described in Annex A.38 with the following input and output parameters:

   - in *cap* - DUT capabilities

   - out *itemToRestore1* - deleted authorization server configuration if any

5. Set:

   - *keyAlgorithm* := **"ECC"** if *cap*.KeystoreCapabilities.ECCKeyPairGeneration = true; **"RSA"** otherwise.

6. ONVIF Client configures authentication server connection using the following steps:

   6.1. ONVIF Client creates certification path validation policy by following the procedure mentioned in Annex A.35 with the following input and output parameters

   - in *cap* - DUT capabilities

   - in *keyAlgorithm* - DUT capabilities

   - in "CN=ONVIF TT AuthServer 1,C=US" - CA certificate subject

   - in "Test CertPathValidationPolicy AuthServer Alias" - certification path validation policy alias

   - out *certPathValidationPolicyIDAuthServer* - certification path validation policy identifier

   - out *certIDAuthServer* - certificate identifier

   - out *keyIDAuthServer* - key pair identifier

   - out *CACertAuthServer* - CA certificate

   - out *privateKeyCACertAuthServer* - CA certificate private key

   6.2. ONVIF Client starts Authorization server with metadata endpoint *authServerMetadataEndpoint1* conforming to RFC8414 with following settings:

- It is configured to any certificate that will **not** pass *certPathValidationPolicyIDAuthServer* validation as a server certificate.
- It is configured to accept OAuth2 client credentials grant flow per [RFC 6749] authentication method only.
- It is configured to accept client_secret_basic authentication method only.
- Client with client identifier *clientID1* with client secret *clientSecret1* and scope *scope1* is added to be authorized.
- Client with client identifier *clientID1* with client secret *clientSecret1* and scope *scope2* is added to be authorized.

6.3. ONVIF Client invokes **CreateAuthorizationServerConfiguration** request with parameters

- Type := **"OAuthClientCredentials"**
- ClientAuth := **"client_secret_basic"**
- ServerUri := *authServerMetadataEndpoint1*
- ClientID := *clientID1*
- ClientSecret := *clientSecret1*
- Scope := *scope1*
- KeyID is skipped
- CertificateID is skipped
- CertPathValidationPolicyID := *certPathValidationPolicyIDAuthServer*

6.4. The DUT responds with **CreateAuthorizationServerConfigurationResponse** message with parameters

- Token =: *authServerToken1*

7. ONVIF Client configures uplink connection using the following steps:

7.1. ONVIF Client creates certification path validation policy by following the procedure mentioned in Annex A.35 with the following input and output parameters

- in *cap* - DUT capabilities

- in *keyAlgorithm* - DUT capabilities
- in "CN=ONVIF TT Uplink 1,C=US" - CA certificate subject
- in "Test CertPathValidationPolicy Uplink Alias" - certification path validation policy alias
- out *certPathValidationPolicyIDUplink* - certification path validation policy identifier
- out *certIDUplink* - certificate identifier
- out *keyIDUplink* - key pair identifier
- out *CACertUplink* - CA certificate
- out *privateKeyCACertUplink* - CA certificate private key

7.2. ONVIF Client creates a certificate signed by private key of the CA-certificate with subject and a corresponding public key in the certificate along with the corresponding private key by following the procedure described in Annex A.37 with the following input and output parameters:

- in *cap* - DUT capabilities
- in "CN=ONVIF TT Uplink 2,C=US" - certificate subject
- in *CACertUplink* - CA-certificate
- in *privateKeyCACertUplink* - private key of CA-certificate for certificate signature
- out *certUplink* - certificate
- out *publicKeyUplink* - public key of certificate
- out *privateKeyUplink* - private key of certificate

7.3. ONVIF Client configure and start service endpoint for uplink connection at address *uplinkAddress1* using *certUplink* certificate.

7.4. ONVIF Client invokes **SetUplink** request with parameters

- RemoteAddress := *uplinkAddress1*
- CertificateID is skipped
- UserLevel := **Administrator**

- Status is skipped
- CertPathValidationPolicyID := *certPathValidationPolicyIDUplink*
- AuthorizationServer := *authServerToken1*
- Error is skipped

7.5. The DUT responds with **SetUplinkResponse** message.

8. ONVIF Client verifies initial connection establishment:

8.1. The DUT opens connection to *authServerMetadataEndpoint1* with client identifier *clientID1* and client secret *clientSecret1* using client_secret_basic method authentication and OAuth2 client credentials grant flow.

8.2. The DUT verifies certificate provided by authentication server based on *certPathValidationPolicyIDAuthServer* certification path validation policy and prevent connection.

9. ONVIF Client restores the DUT state.

**Test Result:**

**PASS –**

- DUT passed all assertions.

**FAIL –**

- DUT did not send **CreateAuthorizationServerConfigurationResponse** message.
- DUT did not send **SetUplinkResponse** message.
- DUT establishes connection at step 8.2 during *operationDelay* timeout.

**Note:** *operationDelay* will be taken from Operation Delay field of ONVIF Device Test Tool.

## 5.4 Capabilities

### 5.4.1 GET SERVICES AND GET UPLINK SERVICE CAPABILITIES CONSISTENCY

**Test Case ID:** UPLINK-4-1-1

**Specification Coverage:** Capability exchange (ONVIF Core Specification), Capabilities (Uplink Service Specification)

**Feature Under Test:** GetServices, GetServiceCapabilities (Uplink)

**WSDL Reference:** devicemgmt.wsdl, uplink.wsdl

**Test Purpose:** To verify getting Uplink Service using GetServices request. To verify Get Services and Uplink Service Capabilities consistency.

**Pre-Requisite:** Uplink Service is received from the DUT.

**Test Configuration:** ONVIF Client and DUT

**Test Procedure:**

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetServices** message with parameters:
   - IncludeCapability := false
4. The DUT responds with a **GetServicesResponse** message with parameters:
   - Service list =: *listOfServicesWithoutCapabilities*
5. If *listOfServicesWithoutCapabilities* does not contain item with Namespace = "http://www.onvif.org/ver10/uplink/wsdl", FAIL the test and skip other steps.
6. Set *uplinkServ* := item from *listOfServicesWithoutCapabilities* list with Namespace = "http://www.onvif.org/ver10/uplink/wsdl".
7. If *uplinkServ*.Capabilities is specified, FAIL the test and skip other steps.
8. ONVIF Client invokes **GetServices** message with parameters:
   - IncludeCapability := true
9. The DUT responds with a **GetServicesResponse** message with parameters:
   - Service list =: *listOfServicesWithCapabilities*
10. If *listOfServicesWithCapabilities* does not contain item with Namespace = "http://www.onvif.org/ver10/uplink/wsdl", FAIL the test and skip other steps.
11. Set *uplinkServ* := item from *listOfServicesWithCapabilities* list with Namespace = "http://www.onvif.org/ver10/uplink/wsdl".

12. If *uplinkServ*.Capabilities is not specified, FAIL the test and skip other steps.
13. If *uplinkServ*.Capabilities does not contain valid Capabilities element for Uplink service from "http://www.onvif.org/ver10/uplink/wsdl" namespace, FAIL the test and skip other steps.
14. ONVIF Client invokes **GetServiceCapabilities** (Uplink) message.
15. The DUT responds with **GetServiceCapabilitiesResponse** message with parameters
    - Capabilities =: *cap*
16. If *cap* differs from *uplinkServ*.Capabilities.Capabilities, FAIL the test.

**Test Result:**

**PASS –**

- DUT passed all assertions.

**FAIL –**

- The DUT did not send **GetServicesResponse** message.
- The DUT did not send **GetServiceCapabilitiesResponse** message.

**Note:** The following fields are compared at step 16:

- MaxUplinks

# Annex A Helper Procedures and Additional Notes

## A.1 Clean Up Uplink Configurations

**Name:** HelperCleanUpUplinkConfigurations

**Procedure Purpose:** Helper procedure to remove all uplink configurations from the DUT.

**Pre-requisite:** Uplink Service is received from the DUT.

**Input:** None

**Returns:** None.

**Procedure:**

1. ONVIF Client invokes **GetUplinks** request.
2. The DUT responds with **GetUplinksResponse** message with parameters
   - Uplink Configurations list =: *uplinkConfigList*
3. For each uplink configuration (*uplinkConfig*) from *uplinkConfigList* repeat the following steps:

   3.1. ONVIF Client invokes **DeleteUplink** request with parameters

   - RemoteAddress := *uplinkConfig*.RemoteAddress

   3.2. The DUT responds with **DeleteUplinkResponse** message.

**Procedure Result:**

**PASS –**

- DUT passed all assertions.

**FAIL –**

- DUT did not send **GetUplinksResponse** message.
- DUT did not send **DeleteUplinkResponse** message.

## A.2 Configure Client Certificate

**Name:** HelperConfigureClientCertificate

**Procedure Purpose:** Helper procedure to choose or create a certificate with private key.

**Pre-requisite:** Security Configuration Service is received from the DUT. Create self-signed certificate by the DUT as indicated by the SelfSignedCertificateCreationWithRSA or PCKS#10 supported by the DUT as indicated by the PKCS10ExternalCertificationWithRSAcapability or certificate along with an RSA private key in a PKCS#12 data structure upload is supported by the DUT as indicated by the PKCS12CertificateWithRSAPrivateKeyUpload capability.

**Input:** None

**Returns:** Certificate that has private key and uploaded to the DUT (*certDUT*, *certId*).

**Procedure:**

1. ONVIF Client tries to find existing certificate with private key by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out (optional) *certDUT* - Certificate that has private key.
   - out (optional) *certId* - Certificate Id that has private key.
2. If certificate was returned on step 1, skip other steps and return to test procedure.
3. ONVIF Client invokes **GetServiceCapabilities** for Security Configuration Service.
4. The DUT responds with **GetServiceCapabilitiesResponse** message with parameters
   - Capabilities =: *cap*
5. If certificate along with an RSA private key in a PKCS#12 data structure upload is supported by the DUT as indicated by the PKCS12CertificateWithRSAPrivateKeyUpload capability:

   5.1. ONVIF Client creates a CA certificate (out *certDUT*) and a corresponding public key (out *publicKey*) in the certificate along with the corresponding private key (out *privateKey*) in the form of a PKCS#12 file (out *PKCS12data*) and uploads it with certification path ID (out *certificationPathID*) and key pair ID (out *keyID*) by following the procedure described in Annex A.4.

   5.2. ONVIF Client invokes **GetCertificationPath** with parameters
   - CertificationPathID =: *certificationPathID*

   5.3. The DUT responds with a **GetCertificationPathResponse** message with parameters
   - CertificationPath.CertificateID[0] =: *certId*
   - CertificationPath.Alias

   5.4. Skip other steps and return to test procedure.

6. If self-signed certificate is supported by the DUT as indicated by the SelfSignedCertificateCreationWithRSA capability:

   6.1. ONVIF Client creates a self-signed certificate (out *certId*) and related RSA key pair (out *keyID*) by following the procedure mentioned in Annex A.12.

   6.2. ONVIF Client invokes **GetCertificate** message with parameters

   - CertificateID := *certID*

   6.3. The DUT responds with a **GetCertificateResponse** message with parameters

   - Certificate =: *certDUT*

   6.4. Skip other steps and return to test procedure.

7. If PKCS#10 supported by the DUT as indicated by the PKCS10ExternalCertificationWithRSA capability:

   7.1. ONVIF Client creates a CA certificate (out *certCA*) and a corresponding private key (out *privateKey*) by following the procedure described in Annex A.7.

   7.2. ONVIF Client creates and uploads a CA-signed certificate (out *certId*, out *certDUT*) for RSA key pair (out *keyID1*) based on associated CA certificate (in *certCA*) and a corresponding private key (in *privateKey*) by following the procedure described in Annex A.16.

   7.3. Skip other steps and return to test procedure.

8. FAIL the test and restore DUT configuration if needed.

**Procedure Result:**

**PASS –**

- DUT passed all assertions.

**FAIL –**

- DUT did not send **GetServiceCapabilitiesResponse** message.
- DUT did not send **GetCertificationPathResponse** message.
- DUT did not send **GetCertificateResponse** message.

## A.3 Choose Client Certificate With Private Key

**Name:** HelperChooseClientCertificateWithPrivateKey

**Procedure Purpose:** Helper procedure to choose one certificate that has private key.

**Pre-requisite:** Security Configuration Service is received from the DUT. RSA key pair generation is supported by the DUT as indicated by the RSAKeyPairGeneration capability.

**Input:** None

**Returns:** (opional) Certificate (*certDUT*) that has private key.

**Procedure:**

1. ONVIF Client invokes **GetAllCertificates**.
2. The DUT responds with a **GetAllCertificatesResponse** message with parameters
   - Certificate list =: *certificateList*
3. ONVIF Client invokes **GetAllKeys**.
4. The DUT responds with a **GetAllKeysResponse** message with parameters
   - KeyAttribute list =: *keyAttributeList*
5. For each *certificate* from *certificateList*:

   5.1. If *keyAttributeList*[KeyID = *certificate*.KeyID].hasPrivateKey = true, return *certificate* to test procedure and skip other steps.

**Procedure Result:**

**PASS –**

- DUT passed all assertions.

**FAIL –**

- DUT did not send **GetAllCertificates** message.
- DUT did not send **GetAllKeysResponse** message.

## A.4 Upload PKCS#12 – no key pair exists

**Name:** HelperUploadPKCS12

**Procedure Purpose:** Helper procedure to create and upload PKCS#12 data structure with new public key and private key.

**Pre-requisite:** Security Configuration Service is received from the DUT. Certificate along with an ECC or RSA private key in a PKCS#12 data structure upload is supported by the DUT as indicated by the PKCS12 or PKCS12CertificateWithRSAPrivateKeyUpload capability. The DUT shall have enough free storage capacity for one additional key pair. The DUT shall have enough free storage capacity for one additional certificate. The DUT shall have enough free storage capacity for one additional certification path.

**Input:** The subject (*subject*) of CA certificate (optional input parameter, could be skipped). The service capabilities (*cap*). The key pair algorithm (*keyAlgorithm*).

**Returns:** A [PKCS#12] compliant PKCS#12 data structure (*PKCS12data*) with CA certificate (*CAcert*) and a corresponding key pair (*keyPair*) with a corresponding public key in the certificate along with the corresponding private key and certification path ID (*certificationPathID*) with corresponding key pair ID (*keyID*) for uploaded PKCS#12 data structure.

**Procedure:**

1. ONVIF Client generates an encryption passphrase *passphrase1* (see Annex A.5).
2. ONVIF Client creates an CA certificate in a form of PKCS#12 file with with given passphrase by following the procedure mentioned in Annex A.6 with the following input and output parameters
   - in *cap* - DUT capabilities
   - in *keyAlgorithm* - key pair algorithm
   - in *subject* - CA certificate subject
   - in *passphrase1* - passphrase
   - out *PKCS12data* - PKCS#12 file
   - out *CAcert* - CA certificate
   - out *keyPair* - key pair for the CA certificate
3. ONVIF Client invokes **UploadCertificateWithPrivateKeyInPKCS12** with parameters
   - CertWithPrivateKey := *PKCS12data*
   - CertificationPathAlias := "ONVIF_Certification_Path_Test"
   - KeyAlias := "ONVIF_Key_Test"
   - IgnoreAdditionalCertificates skipped
   - IntegrityPassphraseID skipped
   - EncryptionPassphraseID skipped

- Passphrase := *passphrase1*

4. The DUT responds with a **UploadCertificateWithPrivateKeyInPKCS12Response** message with parameters
    - CertificationPathID =: *certificationPathID*
    - KeyID =: *keyID*

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **UploadCertificateWithPrivateKeyInPKCS12Response** message.

## A.5 Passphrases for test cases

Use the following passphrases for test cases (20 ASCII characters):

- *passphrase1* := "Passphrase for ONVIF"
- *passphrase2* := "AdditionalPassphrase"

## A.6 Creating a PKCS#12 data structure with new CA-signed certificate signed by new public key and private key with passphrase

**Name:** HelperCreatePKCS12WithNewCACertWithPassphrase

**Procedure Purpose:** Helper procedure to create CA certificate and a corresponding public key in the certificate along with the corresponding private key and encryption passphrase in the form of a PKCS#12 file.

**Pre-requisite:** None.

**Input:** The passphrase (*passphrase*) to use in encryption. The subject (*subject*) of CA certificate (optional input parameter, could be skipped). The service capabilities (*cap*). The key pair algorithm (*keyAlgorithm*).

**Returns:** A [PKCS#12] compliant PKCS#12 data structure (*PKCS12data*) with CA certificate (*CAcert*) and a corresponding key pair (*keyPair*) with a corresponding public key in the certificate along with the corresponding private key encrypted with passphrase (*passphrase*).

**Procedure:**

1. ONVIF Client creates an CA certificate by following the procedure mentioned in Annex A.7 with the following input and output parameters
   - in *cap* - DUT capabilities
   - in *keyAlgorithm* - key pair algorithm
   - in *subject* - CA certificate subject
   - out *keyPair* - key pair for the CA certificate
2. ONVIF Client creates a PKCS#12 file by following the procedure mentioned in Annex A.11 with the following input and output parameters
   - in *CAcert* - CA certificate
   - in *keyPair* - key pair
   - in *passphrase* - passphrase
   - out *PKCS12data* - PKCS#12 file

## A.7 Provide CA certificate

**Name:** HelperCreateCACertificate

**Notes:** Annex is used at:

- ONVIF Real Time Streaming using Media2 Device Test Specification
- ONVIF Security Configuration Device Test Specification
- ONVIF Base Device Test Specification

**Procedure Purpose:** Helper procedure to create an X.509 CA certificate.

**Pre-requisite:** None.

**Input:** The subject (*subject*) of certificate (optional input parameter, could be skipped). The service capabilities (*cap*). The key pair algorithm (*keyAlgorithm*).

**Returns:** An X.509 CA certificate (*CAcert*) that is compliant to [RFC 5280] and a corresponding key pair (*keyPair*) with private key and public key.

**Procedure:**

1. ONVIF Client selects signature algorithm by following the procedure mentioned in Annex A.8 with the following input and output parameters:
   - in *cap* - DUT capabilities
   - in *keyAlgorithm* - key pair algorithm
   - out *signatureAlgorithm* - signature algorithm
2. ONVIF Client generates a key pair by following the procedure mentioned in Annex A.9 with the following input and output parameters:
   - in *cap* - DUT capabilities
   - in *keyAlgorithm* - key pair algorithm
   - out *keyPair* - key pair
3. If *subject* is skipped set:
   - *subject* := "CN=ONVIF TT,C=US"
4. ONVIF Client creates an X.509 self-signed CA certificate that is compliant to [RFC 5280] and has the following properties:
   - version := v3
   - signature := *signatureAlgorithm*
   - validity := not before 19700101000000Z and not after 99991231235959Z
   - subject := *subject*
   - public key := *keyPair* .publicKey
   - private key to be used := *keyPair*.privateKey

**Note:** ONVIF Client may return the same CA certificate in subsequent invocations of this procedure for the same subject.

## A.8 Signature Algorithm Selection

**Name:** HelperSignatureAlgorithmSelection

**Notes:** Annex is used at:

- ONVIF Real Time Streaming using Media2 Device Test Specification
- ONVIF Security Configuration Device Test Specification
- ONVIF Base Device Test Specification

**Procedure Purpose:** Helper procedure to select signature algorithm wich will be used for tests based on Device capabilities.

**Pre-requisite:** Security Configuration Service is received from the DUT.

**Input:** The service capabilities (*cap*). The key pair algorithm (*keyAlgorithm*).

**Returns:** The signature algorithm (*signatureAlgorithm*).

**Procedure:**

1. If *keyAlgorithm* = RSA: ONVIF Client selects signature algorithm (*signatureAlgorithm*) that will be used for the test from the list provided by DUT at *cap*.KeystoreCapabilities.SignatureAlgorithms. Selection is done among the following list of signature algorithms supported by the Client by priority from first to last:
   - 1.2.840.113549.1.1.13 (OID of SHA-512 with RSA Encryption algorithm)
   - 1.2.840.113549.1.1.12 (OID of SHA-384 with RSA Encryption algorithm)
   - 1.2.840.113549.1.1.11 (OID of SHA-256 with RSA Encryption algorithm)
   - 1.2.840.113549.1.1.14 (OID of SHA-224 with RSA Encryption algorithm)
   - 1.2.840.113549.1.1.5 (OID of SHA-1 with RSA Encryption algorithm)
2. If *keyAlgorithm* = ECC: ONVIF Client selects signature algorithm (*signatureAlgorithm*) that will be used for the test from the list provided by DUT at *cap*.KeystoreCapabilities.SignatureAlgorithms. Selection is done among the following list of signature algorithms supported by the Client by priority from first to last:
   - 1.2.840.10045.4.3.4 (OID of SHA-512 with ECC Encryption algorithm)
   - 1.2.840.10045.4.3.3 (OID of SHA-384 with ECC Encryption algorithm)
   - 1.2.840.10045.4.3.2 (OID of SHA-256 with ECC Encryption algorithm)
   - 1.2.840.10045.4.3.1 (OID of SHA-224 with ECC Encryption algorithm)
   - 1.2.840.10045.4.1 (OID of SHA-1 with ECC Encryption algorithm)

3. If the previous steps is done with empty signature algorithm (*signatureAlgorithm*): FAIL the procedure.

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not return any of signature algorithms listed at step 1 or 2.

## A.9 Generate a key pair

**Name:** HelperGenerateKeyPair

**Notes:** Annex is used at:

- ONVIF Real Time Streaming using Media2 Device Test Specification
- ONVIF Security Configuration Device Test Specification
- ONVIF Base Device Test Specification

**Procedure Purpose:** Helper procedure to generate a key pair.

**Pre-requisite:** None.

**Input:** The service capabilities (*cap*). The key pair algorithm (*keyAlgorithm*).

**Returns:** A [RFC 3447] compliant RSA or [RFC 5480, RFC 5915] compliant ECC key pair (*keyPair*) with new public key and private key.

**Procedure:**

1. ONVIF Client determines the key pair generation params by following the procedure mentioned in Annex A.10 with the following input and output parameters:
    - in *cap* - DUT capabilities
    - in *keyAlgorithm* - key pair algorithm
    - out *keyGenParams* - key pair generation params
2. If *keyGenParams*.algorithm = RSA:

a. Create an [RFC 3447] compliant RSA key pair (out *keyPair*) with new public key and private key with the following properties:

   - KeyLength := *keyGenParams*.keyLength

3. If *keyGenParams*.algorithm = ECC:

   a. Create an [RFC 5480, RFC 5915] compliant ECC key pair (out *keyPair*) with new public key and private key with the following properties:

      - EllipticCurve := *keyGenParams*.ellipticCurve

## A.10 Determine key pair generation params

**Name:** HelperDetermineKeyPairGenerationParams

**Notes:** Annex is used at:

- ONVIF Real Time Streaming using Media2 Device Test Specification
- ONVIF Security Configuration Device Test Specification
- ONVIF Base Device Test Specification

**Procedure Purpose:** Helper procedure to determine the key pair generation params to use during testing.

**Pre-requisite:** Security Configuration Service is received from the DUT. On-board ECC or RSA key pair generation is supported by the DUT as indicated by the ECCKeyPairGeneration or RSAKeyPairGeneration capability.

**Input:** The service capabilities (*cap*). The key pair algorithm (*keyAlgorithm*).

**Returns:** The key pair generation params (*keyGenParams*).

**Procedure:**

1. If *keyAlgorithm* = RSA:

   a. ONVIF Client loops through the supported Key length list (*cap*.KeystoreCapabilities .RSAKeyLengths) and selects the smallest supported key length (*keyLength*).

   b. ONVIF Client creates the RSA key generation params (*keyGenParams*) with the key length (*keyLength*).

2. If *keyAlgorithm* = ECC:

    a. ONVIF Client loops through the supported elliptic curves list (*cap*.KeystoreCapabilities.EllipticCurves) and selects the simplest elliptic curve (*ellipticCurve*).

    b. ONVIF Client creates the ECC key generation params (*keyGenParams*) with the elliptic curve (*ellipticCurve*).

3. If previous steps is done with empty key pair generation params (*keyGenParams*): FAIL the procedure.

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- No supported RSA key length was found at step 1.1 or no supported ECC elliptic curves was found at step 2.1.

## A.11 Creating a PKCS#12 data structure with existing CA-signed certificate and a corresponding public key and private key with passphrase

**Name:** HelperCreatePKCS12WithPassphrase

**Procedure Purpose:** Helper procedure to create a PKCS#12 data structure with existing CA-signed certificate and a corresponding public key and private key with passphrase.

**Pre-requisite:** None.

**Input:** An X.509 CA certificate (*CAcert*) that is compliant to [RFC 5280] and a corresponding key pair (*keyPair*) with private key and public key, and passphrase (*passphrase*).

**Returns:** A [PKCS#12] compliant PKCS#12 data structure (*PKCS12data*).

**Procedure:**

1. Use the current PrivateKeyInfo data:

    - PrivateKeyInfo

        - version := v2
        - privateKeyAlgorithm := *keyPair*.algorithm
        - privateKey := *keyPair* .privateKey
        - publicKey := *keyPair* .publicKey

2. Create an EncryptedPrivateKeyInf data structure with the following properties:
    - EncryptedPrivateKeyInfo
        - encryptionAlgorithm := pbeWithSHAAnd3-KeyTripleDES-CBC
        - encryptedData := encrypted with *passphrase* PrivateKeyInfo data
3. Create an [PKCS#12] compliant PKCS#12 data structure (*PKCS12data*) with the following properties:
    - version := v3
    - authSafe
        - SafeBag
            - Pkcs-12-PKCS9ShroudedKeyBag := EncryptedPrivateKeyInfo
            - PKCS12AttrSet
                - friendlyName := "testAlias"
        - SafeBag
            - Pkcs-12-CertBag := *CAcert*
            - PKCS12AttrSet
                - friendlyName := "testAlias"

## A.12 Create a self-signed certificate

**Name:** HelperCreateSelfSignedCertificate

**Notes:** Annex is used at:

- ONVIF Real Time Streaming using Media2 Device Test Specification
- ONVIF Security Configuration Device Test Specification

- ONVIF Base Device Test Specification

**Procedure Purpose:** Helper procedure to create a self-signed certificate.

**Pre-requisite:** Security Configuration Service is received from the DUT. Create self-signed certificate supported by the DUT as indicated by the SelfSignedCertificateCreation or SelfSignedCertificateCreationWithRSA capability. On-board ECC or RSA key pair generation is supported by the DUT as indicated by the ECCKeyPairGeneration or RSAKeyPairGeneration capability.

The DUT shall have enough free storage capacity for one additional key pair. The DUT shall have enough free storage capacity for one additional certificate.

**Input:** The service capabilities (*cap*). The key pair algorithm (*keyAlgorithm*).

**Returns:** The identifier of the new certificate (*certID*) and key pair (*keyID*).

**Procedure:**

1. ONVIF Client creates a key pair by following the procedure mentioned in Annex A.13 with the following input and output parameters:
   - in *cap* - DUT capabilities
   - in *keyAlgorithm* - key pair algorithm
   - out *keyID* - key pair ID
2. ONVIF Client selects signature algorithm by following the procedure mentioned in Annex A.8 with the following input and output parameters:
   - in *cap* - DUT capabilities
   - in *keyAlgorithm* - Key Pair Algorithm
   - out *signatureAlgorithm* - signature algorithm
3. ONVIF Client invokes **CreateSelfSignedCertificate** with parameters
   - X509Version skipped
   - KeyID := *keyID*
   - Subject := subject (see Annex A.15)
   - Alias skipped
   - notValidBefore skipped

- notValidAfter skipped
- SignatureAlgorithm := *signatureAlgorithm*
- SignatureAlgorithm.parameters skipped
- SignatureAlgorithm.anyParameters skipped
- Extension skipped

4. The DUT responds with **CreateSelfSignedCertificateResponse** message with parameters
   - CertificateID =: *certID*

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **CreateSelfSignedCertificateResponse** message.

## A.13 Create a key pair

**Name:** HelperCreateKeyPair

**Notes:** Annex is used at:

- ONVIF Real Time Streaming using Media2 Device Test Specification
- ONVIF Security Configuration Device Test Specification
- ONVIF Base Device Test Specification

**Procedure Purpose:** Helper procedure to create ECC or RSA key pair

**Pre-requisite:** Security Configuration Service is received from the DUT. On-board ECC or RSA key pair generation is supported by the DUT as indicated by the ECCKeyPairGeneration or RSAKeyPairGeneration capability. The DUT shall have enough free storage capacity for one additional key pair.

**Input:** The service capabilities (*cap*). The key pair algorithm (*keyAlgorithm*).

**Returns:** The identifier of the new key pair (*keyID*).

**Procedure:**

1. ONVIF Client determines the key pair generation params by following the procedure mentioned in Annex A.10 with the following input and output parameters:
    - in *cap* - DUT capabilities
    - in *keyAlgorithm* - key pair algorithm
    - out *keyGenParams* - key pair generation params
2. If *keyGenParams*.algorithm = RSA:
    a. ONVIF Client invokes **CreateRSAKeyPair** with parameter
        - KeyLength := *keyGenParams*.keyLength

    b. The DUT responds with **CreateRSAKeyPairResponse** message with parameters
        - KeyID =: *keyID*
        - EstimatedCreationTime =: *duration*
3. If *keyGenParams*.algorithm = ECC:
    a. ONVIF Client invokes **CreateECCKeyPair** with parameter
        - EllipticCurve := *keyGenParams*.ellipticCurve

    b. The DUT responds with **CreateECCKeyPairResponse** message with parameters
        - KeyID =: *keyID*
        - EstimatedCreationTime =: *duration*
4. Until *operationDelay* + *duration* expires repeat the following steps:

    4.1. ONVIF Client waits for 5 seconds.

    4.2. ONVIF Client invokes **GetKeyStatus** with parameters
        - KeyID := *keyID*

    4.3. The DUT responds with **GetKeyStatusResponse** message with parameters
        - KeyStatus =: *keyStatus*

    4.4. If *keyStatus* is equal to "ok", *keyID* will be return as a result of the procedure, other steps will be skipped.

   4.5. If *keyStatus* is equal to "corrupt", FAIL the procedure and deletes the key pair (*keyID*) by following the procedure mentioned in Annex A.14.

5. If *operationDelay* + *duration* expires for step 4 and the last *keyStatus* is other than "ok", FAIL the procedure and deletes the key pair (*keyID*) by following the procedure mentioned in Annex A.14.

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **CreateRSAKeyPairResponse** or **CreateECCKeyPairResponse** message.
- DUT did not send **GetKeyStatusResponse** message(s).

**Note:** *operationDelay* will be taken from Operation Delay field of ONVIF Device Test Tool.

## A.14 Delete a key pair

**Name:** HelperDeleteKeyPair

**Notes:** Annex is used at:

- ONVIF Real Time Streaming using Media2 Device Test Specification
- ONVIF Security Configuration Device Test Specification
- ONVIF Base Device Test Specification

**Procedure Purpose:** Helper procedure to delete a key pair.

**Pre-requisite:** Security Configuration Service is received from the DUT. On-board RSA or ECC key pair generation is supported by the DUT as indicated by the RSAKeyPairGeneration or ECCKeyPairGeneration capability.

**Input:** The identifier of the key pair (*keyID*) to delete.

**Returns:** None

**Procedure:**

1. ONVIF Client invokes **DeleteKey** with parameters
   - KeyID := *keyID*

2. DUT responds with a **DeleteKeyResponse** message.

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **DeleteKeyResponse** message.

## A.15 Subject for a server certificate

**Name:** HelperSubjectForServerCertificate

**Notes:** Annex is used at:

- ONVIF Real Time Streaming using Media2 Device Test Specification
- ONVIF Security Configuration Device Test Specification
- ONVIF Base Device Test Specification

Use the following subject for test cases:

- Subject.Country := "US"
- Subject.CommonName := <DUT IP-address>

## A.16 Create and upload a CA-signed certificate for private key

**Name:** HelperUploadCASignedCertificate

**Procedure Purpose:** Helper procedure to create and upload a CA-signed certificate for private key

**Pre-requisite:** Security Configuration Service is received from the DUT. Create PCKS#10 supported by the DUT as indicated by the PKCS10 or PKCS10ExternalCertificationWithRSA capability. On-board ECC or RSA key pair generation is supported by the DUT as indicated by the ECCKeyPairGeneration or RSAKeyPairGeneration capability. The DUT shall have enough free storage capacity for one additional key pair. The DUT shall have enough free storage capacity for one additional certificate. Current time of the DUT shall be at least Jan 01, 1970.

**Input:** CA certificate (*CAcert*) and a corresponding private key (*caPrivateKey*). The service capabilities (*cap*). The key pair algorithm of the key which will be in the result certificate (*certKeyAlgorithm*).

**Returns:** The identifier of the new key pair (*keyID*), a certificate identifier (*certID*), a certificate (*cert*).

**Procedure:**

1. ONVIF Client creates a certificate from the PKCS#10 request with key pair and associated CA certificate and a corresponding private key by following the procedure described in Annex A.17 with the following input and output parameters:
   - in *cap* - DUT capabilities
   - in *CAcert* - CA certificate
   - in *caPrivateKey* - private key
   - in *certKeyAlgorithm* - key pair algorithm
   - out *cert* - certificate
   - out *keyID1* - key pair ID
2. ONVIF Client invokes **UploadCertificate** with parameters
   - Certificate := *cert*
   - Alias := "ONVIF_Test1"
   - PrivateKeyRequired := true
3. The DUT responds with **UploadCertificateResponse** with parameters
   - CertificateID =: *certID*
   - KeyID =: *keyID*

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **UploadCertificateResponse** message.

## A.17 Create a CA-signed certificate for the key pair

**Name:** HelperCreateCASignedCertificate

**Procedure Purpose:** Helper procedure to create a CA-signed certificate for key pair.

**Pre-requisite:** Security Configuration Service is received from the DUT. Create PCKS#10 supported by the DUT as indicated by the PKCS10 or PKCS10ExternalCertificationWithRSA capability. On-board ECC or RSA key pair generation is supported by the DUT as indicated by the ECCKeyPairGeneration or RSAKeyPairGeneration capability. The DUT shall have enough free storage capacity for one additional key pair. Current time of the DUT shall be at least Jan 01, 1970.

**Input:** CA certificate (*CAcert*) and a corresponding private key (*caPrivateKey*). The service capabilities (*cap*). The CSR key pair algorithm (*csrKeyAlgorithm*).

**Returns:** The identifier of the new key pair (*keyID*), a CA-signed certificate (*cert*).

**Procedure:**

1. ONVIF Client creates a key pair by following the procedure mentioned in Annex A.13 with the following input and output parameters:
   - in *cap* - DUT capabilities
   - in *csrKeyAlgorithm* - key pair algorithm
   - out *csrKeyID* - key pair ID
2. ONVIF Client selects signature algorithm by following the procedure mentioned in Annex A.8 with the following input and output parameters:
   - in *cap* - DUT capabilities
   - in csrKeyAlgorithm - key pair algorithm
   - out *caSignatureAlgorithm* - signature algorithm
3. ONVIF Client invokes **CreatePKCS10CSR** with parameters
   - Subject := *subject* (see Annex A.15)
   - KeyID := *csrKeyID*
   - CSRAttribute skipped
   - SignatureAlgorithm.algorithm := *signatureAlgorithm*
4. The DUT responds with **CreatePKCS10CSRResponse** message with parameters
   - PKCS10CSR =: *PKCS10request*
5. ONVIF Client creates a certificate from the PKCS#10 request and an associated CA certificate with related private key by following the procedure described in Annex A.18 with the following input and output parameters:

- in *cap* - DUT capabilities
- in *PKCS10request* - PKCS#10 request
- in *CAcert* - CA certificate
- out *privateKey* - private key
- out *cert* - certificate

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **CreatePKCS10CSRResponse** message.

## A.18 Creating a certificate from a PCKS#10 request

**Name:** HelperCreateCertificateFromPKCS10CSR

**Procedure Purpose:** Helper procedure to create an X.509 certificate from a PKCS#10 certification request.

**Pre-requisite:** None.

**Input:** PKCS#10 request (*pkcs10*) and associated CA certificate (*CAcert*) and a corresponding private key (*privateKey*). The service capabilities (*cap*).

**Returns:** An [RFC 5280] compliant X.509 certificate (*certResult*) from the PKCS#10 request signed with the private key of the CA certificate.

**Procedure:**

1. ONVIF Client selects signature algorithm by following the procedure mentioned in Annex A.8 with the following input and output parameters:
   - in *cap* - DUT capabilities
   - in *privateKey*.algorithm - key pair algorithm
   - out *signatureAlgorithm* - signature algorithm
2. Create an [RFC 5280] compliant X.509 certificate (*certResult*) from the PKCS#10 request (*pkcs10*) with the following properties:

- version:= v3
- signature := *signatureAlgorithm*
- subject := subject from the PKCS#10 request (*pkcs10*)
- subject public key := subject public key in the PKCS#10 request (*pkcs10*)
- validity := not before 19700101000000Z and not after 99991231235959Z
- certificate signature is generated with the private key (*privateKey*) of the CA certificate (*CAcert*)
- certificate extensions := the X.509v3 extensions from the PKCS#10 request (*pkcs10*)

## A.19 Uplink Connection Establishment

**Name:** HelperUplinkConnectionEstablishment

**Procedure Purpose:** Helper procedure to execute Uplink connection initiated by the DUT to service endpoint.

**Pre-requisite:** Uplink is supported by the DUT.

**Input:** Client Certificate set for Client TLC Authentification (*clientAuthentificationCertificate*).

**Returns:** None.

**Procedure:**

1. The DUT invokes **ClientHello** with parameters
   - ClientVersion =: *tlcVersion*
   - Random number =: *ClientRandom[32]*
   - CipherSuites
   - Compression methods list
   - SessionID skipped
   - Extension: server_name
2. The Service Endpoint TLS server responds with a **ServerHello** message with parameters
   - Version := *tlcVersion*

- Random number := *ServerRandom[32]*, that is 4-byte number that consists of the client’s date and time plus a 28-byte randomly generated number
- CipherSuite := the strongest cipher that both the client and server support
- Compression method := NONE
- Session ID := *sessionID*

3. The Service Endpoint TLS server responds with **Certificate** message with parameters
   - Certificate.CertificateID := Service Endpoint Certificate ID
   - Certificate.KeyID := *KeyID* Service Endpoint Key ID
4. The Service Endpoint TLS server responds with a **ServerHelloDone** message.
5. The Service Endpoint sends a **ClientCertificateRequest** message.
6. The DUT sends Certificate =: *clientCertificate*.
7. The Service Endpoint validates *clientCertificate*. If *clientCertificate* does not correspond to *clientAuthentificationCertificate*, FAIL the test and skip other steps.
8. The DUT invokes **ClientKeyExchange** message with parameters
   - Premaster Secret =: *PreMasterSecret* encrypted with *KeyID*
9. The DUT computes *MasterSecret* using ClientRandom[32], ServerRandom[32] and PreMasterSecret.
10. The Service Endpoint TLS server computes *MasterSecret* using *ClientRandom[32]*, *ServerRandom[32]* and *PreMasterSecret*.
11. The DUT invokes **ChangeCipherSpec** message.
12. The DUT invokes encrypted **Finished** message, containing a hash and MAC over the previous handshake messages.
13. The Service Endpoint TLS server decrypts the client's **Finished** message and verify the hash and MAC.
14. The Service Endpoint TLS server responds its encrypted **Finished** message, containing a hash and MAC over the previous handshake messages.
15. The DUT invokes **HTTP GET** request to to switch connection to HTTP/2 with the following header

- Upgrade = "h2"
- Connection = "Upgrade"

16. The Service Endpoint responds with **HTTP 101 Switching Protocols** message with parameters
    - Upgrade = "h2"
    - Connection = "Upgrade"

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- The DUT did not send **ClientHello** message.
- The DUT did not send **ClientKeyExchange** message.
- The DUT did not send **ChangeCipherSpec** message.
- The DUT TLS server did not send **Finished** message.

## A.20 Device Configuration for Video Streaming

**Name:** HelperDeviceConfigurationForVideoStreaming

**Procedure Purpose:** Helper procedure to configure Media profile, Video Encoder Configuration, and get stream URI from the DUT for video streaming.

**Pre-requisite:** Media2 Service is received from the DUT.

**Input:** Required video encoding (*requiredVideoEncoding*), Transport protocol (*protocol*), IP version (*ipVersion*).

**Returns:** Stream Uri (*streamUri*).

**Procedure:**

1. ONVIF Client selects a Media Profile with required video encoding support by following the procedure mentioned in Annex A.21 with the following input and output parameters
    - in *requiredVideoEncoding* - required video encoding

- out *profile* - Media Profile with Video Source Configuration and Video Encoder Configuration with the required video encoding
- out *vecOptions* - Video Encoder Configuration Options for the Media Profile

2. if *protocol* = RtspMulticast:

   2.1. ONVIF Client removes Audio Encoder Configuration and Metadata Configuration from media profile by following the procedure mentioned in Annex A.23 with the following input and output parameters

   - in *profile* - Media Profile

3. ONVIF Client invokes **SetVideoEncoderConfiguration** request with parameters

- Configuration.@token := *profile*.Configurations.VideoEncoder.@token
- Configuration.Name := *profile*.Configurations.VideoEncoder.Name
- Configuration.UseCount := *profile*.Configurations.VideoEncoder.UseCount
- Configuration.@GovLength := minimum item from *vecOptions*.@GovLengthRange list (or skipped if *vecOptions*.@GovLengthRange skipped)
- Configuration.@Profile := highest value from *vecOptions*.@ProfilesSupported list as the order is High/Extended/Main/Baseline (or skipped if *vecOptions*.@ProfilesSupported skipped)
- Configuration.Encoding := *requiredVideoEncoding*
- Configuration.Resolution := resolution closest to 640x480 from *vecOptions*.ResolutionsAvailable list
- if *vecOptions*.@FrameRatesSupported skipped and *profile*.Configurations.VideoEncoder.RateControl skipped:
  - Configuration.RateControl skipped
- if *vecOptions*.@FrameRatesSupported or *profile*.Configurations.VideoEncoder.RateControl is not skipped:
  - Configuration.RateControl.@ConstantBitRate := *vecOptions*.@ConstantBitRateSupported
  - Configuration.RateControl.FrameRateLimit := value closest to 25 but greater than 1 from *vecOptions*.@FrameRatesSupported

list (or *profile*.Configurations.VideoEncoder.RateControl.FrameRateLimit if *vecOptions*.@FrameRatesSupported skipped)

- Configuration.RateControl.BitrateLimit := min {max {*profile*.Configurations.VideoEncoder.RateControl.BitrateLimit, *vecOptions*.BitrateRange.Min}, *vecOptions*.BitrateRange.Max}

- if *protocol* is not equal to RtspMulticast:
  - Configuration.Multicast := *profile*.Configurations.VideoEncoder.Multicast
- if *protocol* = RtspMulticast and *ipVersion* = IPv4:
  - Configuration.Multicast.Address.Type := IPv4
  - Configuration.Multicast.Address.IPv4Address := multicast IPv4 address
  - Configuration.Multicast.Address.IPv6Address skipped
  - Configuration.Multicast.Port := port for multicast streaming
  - Configuration.Multicast.TTL := 1
  - Configuration.Multicast.AutoStart := false
- if *protocol* = RtspMulticast and *ipVersion* = IPv6:
  - Configuration.Multicast.Address.Type := IPv6
  - Configuration.Multicast.Address.IPv4Address skipped
  - Configuration.Multicast.Address.IPv6Address := multicast IPv6 address
  - Configuration.Multicast.Port := port for multicast streaming
  - Configuration.Multicast.TTL := 1
  - Configuration.Multicast.AutoStart := false
- Configuration.Quality := *vecOptions*.QualityRange.Min

4. The DUT responds with **SetVideoEncoderConfigurationResponse** message.

5. ONVIF Client retrieves a stream uri for Media Profile for required transport protocol by following the procedure mentioned in Annex A.24 with the following input and output parameters

   - in *protocol* - Transport protocol

    - in *ipVersion* - IP Type
    - in *profile*.@token - Media profile token
    - out *uri* - Stream URI

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **SetVideoEncoderConfigurationResponse** message.

**Note:** See Annex A.25 for Name and Token Parameters Length limitations.

## A.21 Media2 Service Profile Configuration for Video Streaming

**Name:** HelperFindMediaProfileForVideoStreaming

**Notes:** Annex is used at:

- ONVIF Real Time Streaming using Media2 Device Test Specification

**Procedure Purpose:** Helper procedure to configure Media Profile to contain Video Source Configuration and Video Encoder Configuration with the required video encoding.

**Pre-requisite:** Media2 Service is received from the DUT.

**Input:** Required video encoding (*requiredVideoEncoding*)

**Returns:** Media Profile (*profile*) containing Video Source Configuration and Video Encoder Configuration with the required video encoding. Video Encoder Configuration Options for the Media Profile (*vecOptions*).

**Procedure:**

1. ONVIF Client invokes **GetProfiles** request with parameters
    - Token skipped
    - Type[0] := VideoSource
    - Type[1] := VideoEncoder
2. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

3. For each Media Profile *profile1* in *profileList* with both Configuration.VideoSource and Configuration.VideoEncoder repeat the following steps:

   3.1. ONVIF Client invokes **GetVideoEncoderConfigurationOptions** request with parameters

   - ConfigurationToken := *profile1*.Configuration.VideoEncoder.@token
   - ProfileToken := *profile1*.@token

   3.2. DUT responds with **GetVideoEncoderConfigurationOptionsResponse** message with parameters

   - Options list =: *optionsList*

   3.3. If *optionsList* list contains item with Encoding = *requiredVideoEncoding*:

      3.3.1. Set *profile* := *profile1*.

      3.3.2. Set *vecOptions* := item with Encoding = *requiredVideoEncoding* from *optionsList* list.

      3.3.3. Skip other steps in procedure.

4. For each Media Profile *profile1* in *profileList* that contains VideoSource configuration repeat the following steps:

   4.1. If *profile1*.Configurations.VideoSource.@token is different from video source configuration token of previous profiles in cycle:

      4.1.1. ONVIF Client invokes **GetVideoEncoderConfigurations** request with parameters

      - ConfigurationToken skipped
      - ProfileToken := *profile1*.@token

      4.1.2. The DUT responds with **GetVideoEncoderConfigurationsResponse** with parameters

      - Configurations list =: *videoEncoderConfList*

      4.1.3. For each Video Encoder Configuration *videoEncoderConfiguration1* in *videoEncoderConfList* repeat the following steps:

4.1.3.1. ONVIF Client invokes **GetVideoEncoderConfigurationOptions** request with parameters

- ConfigurationToken := *videoEncoderConfiguration1*.@token
- ProfileToken := *profile1*.@token

4.1.3.2. DUT responds with **GetVideoEncoderConfigurationOptionsResponse** message with parameters

- Options list =: *optionsList*

4.1.3.3. If *optionsList* list contains item with Encoding = *requiredVideoEncoding*:

4.1.3.3.1. ONVIF Client invokes **AddConfiguration** request with parameters

- ProfileToken := *profile1*.@token
- Name skipped
- Configuration[0].Type := VideoEncoder
- Configuration[0].Token := *videoEncoderConfiguration1*.@token

4.1.3.3.2. The DUT responds with **AddConfigurationResponse** message.

4.1.3.3.3. Set *profile* := *profile1*.

4.1.3.3.4. Set *vecOptions* := item with Encoding = *requiredVideoEncoding* from *optionsList* list.

4.1.3.3.5. Skip other steps in procedure.

5. Set *profile1* := *profileList*[0]

6. Set *confTypeList* := (configurations that are contained in profile *profile1*)

7. ONVIF Client removes all configurations from the Media Profile by following the procedure mentioned in Annex A.22 with the following input and output parameters

- in *confTypeList* - list of configuration type to remove from Media Profile

- in *profile1* - Media Profile to update

8. ONVIF Client invokes **GetVideoSourceConfigurations** request with parameters
    - ConfigurationToken skipped
    - ProfileToken := *profile1*.@token
9. The DUT responds with **GetVideoSourceConfigurationsResponse** with parameters
    - Configurations list =: *videoSourceConfList*
10. For each Video Source Configuration *videoSourceConfiguration1* in *videoSourceConfList* repeat the following steps:

    10.1. ONVIF Client invokes **AddConfiguration** request with parameters

    - ProfileToken := *profile1*.@token
    - Name skipped
    - Configuration[0].Type := VideoSource
    - Configuration[0].Token := *videoSourceConfiguration1*.@token

    10.2. The DUT responds with **AddConfigurationResponse** message.

    10.3. ONVIF Client invokes **GetVideoEncoderConfigurations** request with parameters

    - ConfigurationToken skipped
    - ProfileToken := *profile1*.@token

    10.4. The DUT responds with **GetVideoEncoderConfigurationsResponse** with parameters

    - Configurations list =: *videoEncoderConfList*

    10.5. For each Video Encoder Configuration *videoEncoderConfiguration1* in *videoEncoderConfList* repeat the following steps:

    10.5.1. ONVIF Client invokes **GetVideoEncoderConfigurationOptions** request with parameters

    - ConfigurationToken := *videoEncoderConfiguration1*.@token
    - ProfileToken := *profile1*.@token

10.5.2. DUT responds with **GetVideoEncoderConfigurationOptionsResponse** message with parameters

- Options list =: *optionsList*

10.5.3. If *optionsList* list contains item with Encoding = *requiredVideoEncoding*:

10.5.3.1. ONVIF Client invokes **AddConfiguration** request with parameters

- ProfileToken := *profile1*.@token
- Name skipped
- Configuration[0].Type := VideoEncoder
- Configuration[0].Token := *videoEncoderConfiguration1*.@token

10.5.3.2. The DUT responds with **AddConfigurationResponse** message.

10.5.3.3. Set *profile* := *profile1*.

10.5.3.4. Set *vecOptions* := item with Encoding = *requiredVideoEncoding* from *optionsList* list.

10.5.3.5. Skip other steps in procedure.

11. FAIL the test and skip other steps.

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **GetVideoEncoderConfigurationOptionsResponse** message.
- DUT did not send **GetVideoEncoderConfigurationsResponse** message.
- DUT did not send **AddConfigurationResponse** message.
- DUT did not send **GetVideoSourceConfigurationsResponse** message.

## A.22 Removing Configurations from Media Profile

**Name:** HelperRemoveConfigurationsFromMediaProfile

**Notes:** Annex is used at:

- ONVIF Real Time Streaming using Media2 Device Test Specification

**Procedure Purpose:** Helper Procedure to remove configuartions from Media Profile.

**Pre-requisite:** Media2 Service is received from the DUT.

**Input:** Media Profile (*profile*). List of configuration type to remove from profile (*confTypeList*).

**Returns:** None.

**Procedure:**

1. ONVIF Client invokes **GetProfiles** request with parameters
   - Token := *profile*.@token
   - Type[0] := All
2. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
3. If *profileList*[0] contains at least one Configuration with type equals to configuration type from *confTypeList*:

   3.1. ONVIF Client invokes **RemoveConfiguration** request with parameters
   - ProfileToken := *profile*.@token
   - If *profileList*[0] contains Configuration.VideoSource and *confTypeList* contains VideoSource:
     - Configuration[0].Type := VideoSource
     - Configuration[0].Token skipped
   - If *profileList*[0] contains Configuration.VideoEncoder and *confTypeList* contains VideoEncoder:
     - Configuration[1].Type := VideoEncoder
     - Configuration[1].Token skipped

- If *profileList*[0] contains Configuration.AudioSource and *confTypeList* contains AudioSource:
  - Configuration[2].Type := AudioSource
  - Configuration[2].Token skipped
- If *profileList*[0] contains Configuration.AudioEncoder and *confTypeList* contains AudioEncoder:
  - Configuration[3].Type := AudioEncoder
  - Configuration[3].Token skipped
- If *profileList*[0] contains Configuration.AudioOutput and *confTypeList* contains AudioOutput:
  - Configuration[4].Type := AudioOutput
  - Configuration[4].Token skipped
- If *profileList*[0] contains Configuration.AudioDecoder and *confTypeList* contains AudioDecoder:
  - Configuration[5].Type := AudioDecoder
  - Configuration[5].Token skipped
- If *profileList*[0] contains Configuration.Metadata and *confTypeList* contains Metadata:
  - Configuration[6].Type := Metadata
  - Configuration[6].Token skipped
- If *profileList*[0] contains Configuration.Analytics and *confTypeList* contains Analytics:
  - Configuration[7].Type := Analytics
  - Configuration[7].Token skipped
- If *profileList*[0] contains Configuration.PTZ and *confTypeList* contains PTZ:
  - Configuration[8].Type := PTZ
  - Configuration[8].Token skipped

3.2. The DUT responds with **RemoveConfigurationResponse** message.

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **RemoveConfigurationResponse** message.

## A.23 Removing Audio Encoder Configuration and Metadata Configuration from Media Profile

**Name:** HelperRemoveAudioEncoderConfigAndMetadataConfigFromMediaProfile

**Procedure Purpose:** Helper Procedure to guarantee that Media Profile does not contain Audio Encoder Configuration and Metadata Configuration.

**Pre-requisite:** Media2 Service is received from the DUT.

**Input:** Media Profile (*profile*)

**Returns:** None.

**Procedure:**

1. ONVIF Client invokes **GetProfiles** request with parameters
   - Token := *profile*.@token
   - Type[0] := AudioEncoder
   - Type[1] := Metadata
2. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
3. If *profileList*[0] contains Configuration.AudioEncoder or Configuration.Metadata:

   3.1. ONVIF Client invokes **RemoveConfiguration** request with parameters

   - ProfileToken := *profile1*.@token

        - If *profileList*[0] contains Configuration.AudioEncoder:
            - Configuration[0].Type := AudioEncoder
            - Configuration[0].Token skipped
        - If *profileList*[0] contains Configuration.Metadata:
            - Configuration[1].Type := Metadata
            - Configuration[1].Token skipped

    3.2. The DUT responds with **RemoveConfigurationResponse** message.

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **RemoveConfigurationResponse** message.

## A.24 Get Stream Uri

**Name:** HelperGetStreamUri

**Procedure Purpose:** Helper procedure to get stream URI from the DUT.

**Pre-requisite:** Media2 Service is received from the DUT.

**Input:** Protocol (*protocol*). Media Profile token (*token*). IP type (*ipType*) (optional parameter, IPv4 by default).

**Returns:** Stream Uri (*streamUri*).

**Procedure:**

1. ONVIF Client invokes **GetStreamUri** request with parameters
    - Protocol := *protocol*
    - ProfileToken := *token*
2. The DUT responds with **GetStreamUriResponse** message with parameters

- Uri =: *streamUri*

3. If *streamUri* is longer than 128 octets, FAIL the test and skip other steps.

4. If *ipType* skipped, set *ipType* := IPv4.

5. If *streamUri* ip type is not equal to *ipType*, FAIL the test and skip other steps.

6. If *protocol* = RtspOverHttp:

   6.1. If *streamUri* doesn't have the same port with the web service, FAIL the test and skip other steps.

   6.2. If *streamUri* doesn't have the same scheme with the web service ('http' or 'https'), FAIL the test and skip other steps.

7. If *protocol* != RtspOverHttp:

   7.1. If *streamUri* doesn't have scheme equal to 'rtsp', FAIL the test and skip other steps.

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **GetStreamUriResponse** message.

## A.25 Name and Token Parameters

There are the following limitations on maximum length of the Name and Token parameters that shall be used during tests by ONVIF Device Test Tool to prevent faults from DUT:

- Name shall be less than or equal to 64 characters (only readable characters accepted).
- Token shall be less than or equal to 64 characters (only readable characters accepted).
- UTF-8 character set shall be used for Name and Token.

**Note:** these limitations will not be used, if ONVIF Device Test Tool reuses values that were received from the DUT.

## A.26 Set Up Uplink Connection

**Name:** HelperSetUpUplinkConnection

**Procedure Purpose:** Helper procedure to set up Uplink connection between the DUT and uplink service endpoint of the ONVIF Client.

**Pre-requisite:** Uplink Service is received from the DUT.

**Input:** None

**Returns:** None.

**Procedure:**

1. ONVIF Client removes any existing uplink configurations by following the procedure mentioned in Annex A.1.
2. ONVIF Client configures certificate on a DUT that will be used for client authentication by following the procedure mentioned in Annex A.2
   - out *certificateDUT* - Certificate that uploaded to the DUT with private key.
   - out *certId* - Certificate Id for the certificatethat uploaded to the DUT with private key.
3. ONVIF Client starts service endpoint.
4. ONVIF Client invokes **SetUplink** request with parameters
   - RemoteAddress := IPv4 address and port of service endpoint
   - CertificateID := *certId*
   - UserLevel := "Administrator"
   - Status skipped
   - CertPathValidationPolicyID skipped
5. The DUT responds with **SetUplinkResponse** message.
6. ONVIF Client awaits device connecting to service endpoint by following the procedure mentioned in Annex A.19
   - in *certificate* - Client Certificate for TLS authentification.

**Procedure Result:**

**PASS –**

- DUT passed all assertions.

**FAIL –**

- DUT did not send **SetUplinkResponse** message.

## A.27 Media Streaming over WebSocket

**Name:** HelperStreamingOverWebSocket

**Procedure Purpose:** Helper procedure to verify media streaming over WebSocket.

**Pre-requisite:** WebSocket is supported by the DUT.

**Input:** Uri for media streaming (*streamUri*). Media type (*mediaType*). Expected media stream encoding (*encoding*).

**Returns:** None

**Procedure:**

1. ONVIF Client gets Web Socket Uri by following the procedure mentioned in Annex A.28 with the following output parameters
   - out *uri* - Web Socket Uri
2. ONVIF Client establishes a WebSocket Connection by following the procedure mentioned in Annex A.30 with the following input and output parameters
   - in *uri* - Web Socket Uri
3. ONVIF Client invokes **RTSP DESCRIBE** request to *streamUri* address over WebSocket.
4. The DUT responds with **200 OK** message over WebSocket with parameters
   - Response header =: *responseHeader*
   - SDP information =: *sdp*
5. If *sdp* does not contain Media Type = *mediaType* with rtpmap value corresponding to *encoding* and without session attribute "sendonly" (a=sendonly), FAIL the test and skip other steps.
6. ONVIF Client checks types of IP addresses returned in response to DESCRIBE by following the procedure mentioned in Annex A.32 with the following input parameters
   - in *responseHeader* - header of response to DESCRIBE
   - in *sdp* - SDP information
   - in *streamUri* - Uri for media streaming

7. ONVIF Client invokes **RTSP SETUP** request over WebSocket to uri address, which corresponds to *mediaType* media type (see [RFC2326] for details), with parameters
   - Transport := RTP/AVP/TCP;unicast;interleaved=0-1
8. The DUT responds with **200 OK** message over WebSocket with parameters
   - Transport
   - Session =: *session*
9. ONVIF Client invokes **RTSP PLAY** request over WebSocket to uri address, which corresponds to aggregate control (see [RFC2326] for details), with parameters
   - Session := *session*
10. The DUT responds with **200 OK** message over WebSocket with parameters
    - Session
    - RTP-Info
11. If DUT does not send *encoding* RTP media stream to ONVIF Client over RTSP control connection, FAIL the test and skip other steps.
12. If DUT does not send valid RTCP packets, FAIL the test and skip other steps.
13. ONVIF Client invokes **RTSP TEARDOWN** request over WebSocket to uri address, which corresponds to aggregate control (see [RFC2326] for details), with parameters
    - Session := *session*
14. The DUT responds with **200 OK** message over WebSocket with parameters
    - Session

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **RTSP 200 OK** response over WebSocket for **RTSP DESCRIBE**, **RTSP SETUP**, **RTSP PLAY** and **RTSP TEARDOWN** requests.
- RTSP Session is terminated by DUT during media streaming.

**Note:** See Annex A.33 for invalid RTP header definition.

**Note:** If *encoding* = MP4A-LATM, then rtpmap value may be equal either MP4A-LATM or MPEG4-GENERIC at step 5.

## A.28 Get WebSocket URI

**Name:** HelperGetWebSocketURI

**Procedure Purpose:** Helper procedure to get WebSocket URI.

**Pre-requisite:** WebSocket is supported by the DUT.

**Input:** None.

**Returns:** WebSocket URI *uri*.

**Procedure:**

1. ONVIF Client retrieves Media2 Service capabilities by following the procedure mentioned in Annex A.29 with the following input and output parameters
   - out *cap* - Media2 Service capabilities
2. Set *uri* := *cap*.StreamingCapabilities.RTSPWebSocketUri
3. If hierarchical component (hier_part in [rfc2396]) of *uri* is absolute path construction (abs_path in [rfc2396]):

   3.1. ONVIF Client confugures WebSocket URI (*uri*) with host and port based on *uri*, URI of the DUT, and HTTP/HTTPS port of the DUT.

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- None.

## A.29 Get Media2 Service Capabilities

**Name:** HelperGetServiceCapabilities

**Procedure Purpose:** Helper procedure to get Media2 Service Capabilities from the DUT.

**Pre-requisite:** Media2 Service is received from the DUT.

**Input:** None

**Returns:** The service capabilities (*cap*).

**Procedure:**

1. ONVIF Client invokes **GetServiceCapabilities** request.
2. The DUT responds with **GetServiceCapabilitiesResponse** message with parameters
   - Capabilities =: *cap*

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **GetServiceCapabilitiesResponse** message.

## A.30 Web Socket Handshake

**Name:** HelperWebSocketHandshake

**Procedure Purpose:** Helper procedure to establish a WebSocket Connection.

**Pre-requisite:** WebSocket is supported by the DUT.

**Input:** Web Socket Uri (*uri*)

**Returns:** None.

**Procedure:**

1. ONVIF Client generates a Sec-WebSocket-Key value by following the procedure mentioned in Annex A.31 with the following input and output parameters
   - out *webSocketKey* - Sec-WebSocket-Key value.
2. ONVIF Client invokes **HTTPS GET** request to *uri* with parameters
   - Upgrade =: "websocket"

  - Connection =: "Upgrade"
  - Sec-WebSocket-Key =: *webSocketKey*
  - Sec-WebSocket-Protocol =: "rtsp.onvif.org"
  - Sec-WebSocket-Version =: "13"

3. The DUT responds with **HTTPS 101 Switching Protocols** message with parameters
    - Upgrade =: *upgrade*
    - Connection =: *connection*
    - Sec-WebSocket-Accept =: *accept*
    - Sec-WebSocket-Protocol =: *protocol*
4. If *upgrade* is not equal to "websocket", FAIL the test and skip other steps.
5. If *connection* is not equal to "Upgrade", FAIL the test and skip other steps.
6. If *accept* other than the base64-encoded SHA-1 of the concatenation of the *webSocketKey* (see RFC[6455] 4.1. Client Requirements), FAIL the test and skip other steps.
7. If *protocol* is not equal to "rtsp.onvif.org", FAIL the test and skip other steps.

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **HTTP 101**.

## A.31 Sec-WebSocket-Key value generation

**Name:** HelperGenerateSecWebSocketKey

**Procedure Purpose:** Helper procedure to generate a Sec-WebSocket-Key value that is compliant to [RFC6455] and [RFC4648].

**Pre-requisite:** None.

**Input:** None.

**Returns:** Sec-WebSocket-Key value (*webSocketKey*)

**Procedure:**

1. ONVIF Client generates a nonce consisting of a randomly selected 16-byte Sec-WebSocket-Key value that has been base64-encoded (see Section 4 of [RFC4648] and section 4.1 of [RFC6455]).

## A.32 Check of IP address type in response to RTSP DESCRIBE

**Name:** HelperIPAddressTypeInRTSP

**Procedure Purpose:** Helper procedure to check IP addresses types returned by DUT in response to RTSP DESCRIBE.

**Pre-requisite:** None.

**Input:** Header of response to DESCRIBE (*responseHeader*). SDP (*sdp*). Stream Uri (*streamUri*).

**Returns:** None.

**Procedure:**

1. Set *ipType* := *streamUri* IP type.
2. For each **Content-Base** field in *responseHeader* (*contentBase*) that has absolute IP value:
   - 2.1. If *contentBase* IP value does not correspond to *ipType*, FAIL the test and skip other steps (see [RFC2326] for details).
3. For each **Content-Location** field in *responseHeader* (*contentLocation*) that has absolute IP value:
   - 3.1. If *contentLocation* IP value does not correspond to *ipType*, FAIL the test and skip other steps (see [RFC2326] for details).
4. For each **"a=control"** attribute in *sdp* (*aControl*) that has absolute IP value:
   - 4.1. If *aControl* IP value does not correspond to *ipType*, FAIL the test and skip other steps (see [RFC2326] for details).
5. If *ipType* = IPv4:
   - 5.1. If *sdp* contains at least one origin field ("o=") with **addrtype** != "IP4", FAIL the test and skip other steps (see [RFC4566] for details).

5.2. If *sdp* contains at least one origin field ("o=") with IP type of **unicast-address** sub-field != IPv4 type, FAIL the test and skip other steps (see [RFC4566] for details).

5.3. If *sdp* contains at least one connection data field ("c=") with **addrtype** != "IP4", FAIL the test and skip other steps (see [RFC4566] for details).

5.4. If *sdp* contains at least one connection data field ("c=") with IP type of **connection address** sub-field != IPv4 type, FAIL the test and skip other steps (see [RFC4566] for details).

6. If *ipType* = IPv6:

6.1. If *sdp* contains at least one origin field ("o=") with **addrtype** != "IP6", FAIL the test and skip other steps (see [RFC4566] for details).

6.2. If *sdp* contains at least one origin field ("o=") with IP type of **unicast-address** sub-field != IPv6 type, FAIL the test and skip other steps (see [RFC4566] for details).

6.3. If *sdp* contains at least one connection data field ("c=") with **addrtype** != "IP6", FAIL the test and skip other steps (see [RFC4566] for details).

6.4. If *sdp* contains at least one connection data field ("c=") with IP type of **connection address** sub-field != IPv6 type, FAIL the test and skip other steps (see [RFC4566] for details).

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- None.

## A.33 Invalid RTP Header

A RTP header, which is not formed according to the header field format defined in the RFC 3550 Section 5.1, is considered an invalid RTP header.

## A.34 Get service capabilities for Advanced Security service

**Name:** HelperGetServiceCapabilities_AdvancedSecurity

**Notes:** Annex is used at:

- ONVIF Real Time Streaming using Media2 Device Test Specification
- ONVIF Security Configuration Device Test Specification
- ONVIF Base Device Test Specification
- ONVIF Media2 Configuration Device Test Specification

**Procedure Purpose:** Helper procedure to get service capabilities.

**Pre-requisite:** Security Configuration Service is received from the DUT.

**Input:** None

**Returns:** The service capabilities (*cap*).

**Procedure:**

1. ONVIF Client invokes **GetServiceCapabilities**
2. The DUT responds with **GetServiceCapabilitiesResponse** message with parameters
    - Capabilities =: *cap*

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **GetServiceCapabilitiesResponse** message.

## A.35 Create a certification path validation policy for authentication server configuration

**Name:** HelperCreateCertPathValidationPolicyForAuthServer

**Notes:** Annex is used at:

- ONVIF Security Configuration Device Test Specification
- ONVIF Uplink Test Specification
- ONVIF Real Time Streaming using Media2 Device Test Specification

- ONVIF Media2 Configuration Device Test Specification

**Procedure Purpose:** Helper procedure to create a certification path validation policy for authentication server configuration.

**Pre-requisite:** Security Configuration Service is received from the DUT. Certification path validation policy supported by the DUT as indicated by the MaximumNumberOfCertificationPathValidationPolicies capability. UploadCertificate is supported by the DUT as indicated by the PKCS10ExternalCertificationWithRSA or PKCS10 capability. The DUT shall have enough free storage capacity for one additional certification path validation policy. The DUT shall have enough free storage capacity for one additional certification path. The DUT shall have enough free storage capacity for one additional certificate. The DUT shall have enough free storage capacity for one additional key pair.

**Input:** The service capabilities (*cap*). The key pair algorithm (*keyAlgorithm*). The certification path validation policy alias (*certPathValidationPolicyAlias*). The subject (*subject*) of CA certificate (optional input parameter, could be skipped).

**Returns:** The certification path validation policy identifier (*certPathValidationPolicyID*), related certificate (*certID*), key pair (*keyID*), CA certificate (out *CAcert*) CA certificate private key (out *privateKey*).

**Procedure:**

1. ONVIF Client creates a CA certificate and a corresponding private key by following the procedure described in Annex A.7 with the following input and output parameters:
   - in *cap* - DUT capabilities
   - out *CAcert* - CA certificate
   - out *privateKey* - private key
2. ONVIF Client invokes **UploadCertificate** with parameters
   - Certificate := *CAcert*
   - Alias := "ONVIF Test"
   - PrivateKeyRequired : = false
3. The DUT responds with a **UploadCertificateResponse** message with parameters
   - CertificateID =: *certID*
   - KeyID =: *keyID*

4. ONVIF Client creates certification path validation policy identifier with specified alias and the certificate identifier for trust anchor by following the procedure mentioned in Annex A.36 with the following input and output parameters:.

    - in *certID* - certificate identifier for trust anchor
    - in *certPathValidationPolicyAlias* - certification path validation policy alias
    - out *certPathValidationPolicyID* - certification path validation policy identifier

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **UploadCertificateResponse** message.

## A.36 Create a certification path validation policy with provided certificate identifier

**Name:** HelperCreateCertPathValidationPolicyWithCertID

**Notes:** Annex is used at:

- ONVIF Security Configuration Device Test Specification
- ONVIF Uplink Test Specification
- ONVIF Real Time Streaming using Media2 Device Test Specification

**Procedure Purpose:** Helper procedure to create a certification path validation policy with provided certificate identifier.

**Pre-requisite:** Security Configuration Service is received from the DUT. Certification path validation policy supported by the DUT as indicated by the MaximumNumberOfCertificationPathValidationPolicies capability. The DUT shall have enough free storage capacity for one additional certification path validation policy.

**Input:** The certification path validation policy alias (*alias*) and the certificate identifier (*certID*) for trust anchor.

**Returns:** The certification path validation policy identifier (*certPathValidationPolicyID*).

**Procedure:**

1. ONVIF Client invokes **CreateCertPathValidationPolicy** with parameters
    - Alias := *alias*
    - Parameters.RequireTLSWWWClientAuthExtendedKeyUsage skipped
    - Parameters.UseDeltaCRLs = true
    - Parameters.anyParameters skipped
    - TrustAnchor[0].CertificateID := *certID*
    - anyParameters skipped
2. The DUT responds with **CreateCertPathValidationPolicyResponse** message with parameters
    - CertPathValidationPolicyID := *certPathValidationPolicyID*

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **CreateCertPathValidationPolicyResponse** message.

## A.37 Provide certificate signed by private key of other certificate

**Name:** HelperCreateSignedCertificate

**Notes:** Annex is used at:

- ONVIF Security Configuration Device Test Specification
- ONVIF Uplink Test Specification
- ONVIF Real Time Streaming using Media2 Device Test Specification

**Procedure Purpose:** Helper procedure to create an X.509 certificate signed by private key of other certificate.

**Pre-requisite:** None.

**Input:** The subject (*subject*) of certificate and private key (*inputPrivateKey*) of the CA-certificate (*cert*). The service capabilities (*cap*). The key pair algorithm (*keyAlgorithm*). Certificate SAN (*certSAN*, optional).

**Returns:** An X.509 certificate (*cert*) signed by input private key that is compliant to [RFC 5280] and a corresponding key pair (*keyPair*) with the corresponding private key and public key.

**Procedure:**

1. ONVIF Client generates a key pair by following the procedure mentioned in Annex A.9 with the following input and output parameters:
   - in *cap* - DUT capabilities
   - in *keyAlgorithm* - key pair algorithm
   - out *keyPair* - key pair
2. ONVIF Client selects signature algorithm by following the procedure mentioned in Annex A.8 with the following input and output parameters:
   - in *cap* - DUT capabilities
   - in *inputPrivateKey*.algorithm - key pair algorithm
   - out *signatureAlgorithm* - signature algorithm
3. ONVIF Client creates an X.509 certificate signed by *inputPrivateKey* that is compliant to [RFC 5280] and has the following properties:
   - version:= v3
   - signature := *signatureAlgorithm*
   - publicKey := *keyPair*.publicKey
   - validity := validity from *cert*
   - subject := *subject*
   - SAN := *certSAN*
   - issuerDN := subjectDN from *cert*

**Note:** ONVIF Client may return the same certificate in subsequent invocations of this procedure for the same subject and private key.

## A.38 Make Sure That At Least One Authorization Server Configuration Could Be Created

**Name:** HelperEmptySpaceForOneAuthorizationServerConfiguration

**Notes:** Annex is used at:

- ONVIF Security Configuration Device Test Specification
- ONVIF Uplink Test Specification
- ONVIF Real Time Streaming using Media2 Device Test Specification

**Procedure Purpose:** Helper procedure to remove one authorization server configuration if maximum is reached.

**Pre-requisite:** Security Configuration Service is received from the DUT. Authorization Server Configuration is supported by the DUT as indicated by the AuthorizationServer.MaxConfigurations capability.

**Input:** The service capabilities (*cap*).

**Returns:** Removed authorization server configuration (*removedAuthServerConfiguration*), could be empty.

**Procedure:**

1. ONVIF Client gets current authorization server configurations by following the procedure mentioned in Annex A.39 with the following input and output parameters:
   - out *authServerConfigurations1* list - authorization server configurations
2. If *authServerConfigurations1* items number is equal to *cap*.AuthorizationServer.MaxConfigurations:
   2.1. Set the following:
      - *removedAuthServerConfiguration* := *authServerConfigurations1*[0]

   2.2. ONVIF Client invokes **DeleteAuthorizationServerConfiguration** with parameter
      - Token := *removedAuthServerConfiguration*.token

   2.3. The DUT responds with **DeleteAuthorizationServerConfigurationResponse** message.

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **DeleteAuthorizationServerConfigurationResponse** message.

## A.39 Get Authorization Server Configurations List

**Name:** HelperGetAuthorizationServerConfigurations

**Notes:** Annex is used at:

- ONVIF Security Configuration Device Test Specification
- ONVIF Uplink Test Specification
- ONVIF Real Time Streaming using Media2 Device Test Specification

**Procedure Purpose:** Helper procedure to get authorization server configurations list.

**Pre-requisite:** Security Configuration Service is received from the DUT. Authorization Server Configuration is supported by the DUT as indicated by the AuthorizationServer.MaxConfigurations capability.

**Input:** None

**Returns:** Authorization server configurations list (*authServerConfigurations*).

**Procedure:**

1. ONVIF Client invokes **GetAuthorizationServerConfigurations** request with parameters
    - Token is skipped
2. The DUT responds with **GetAuthorizationServerConfigurationsResponse** message with parameters
    - Configuration list =: *authServerConfigurations*

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **GetAuthorizationServerConfigurationsResponse** message.

## A.40 Create Key Pair and Receive Public Key

**Name:** HelperCreateKeyPairAndReceivePublicKey

**Notes:** Annex is used at:

- ONVIF Real Time Streaming using Media2 Device Test Specification
- ONVIF Uplink Test Specification

**Procedure Purpose:** Helper procedure to create a key pair and get public key from the device.

**Pre-requisite:** Security Configuration Service is received from the DUT. Create PCKS#10 supported by the DUT as indicated by the PKCS10 or PKCS10ExternalCertificationWithRSA capability. On-board ECC or RSA key pair generation is supported by the DUT as indicated by the ECCKeyPairGeneration or RSAKeyPairGeneration capability. The DUT shall have enough free storage capacity for one additional key pair. Current time of the DUT shall be at least Jan 01, 1970.

**Input:** The service capabilities (*cap*). The CSR key pair algorithm (*csrKeyAlgorithm*).

**Returns:** The identifier of the new key pair (*keyID*), a public key (*publicKey*).

**Procedure:**

1. ONVIF Client creates a key pair by following the procedure mentioned in Annex A.13 with the following input and output parameters:
   - in *cap* - DUT capabilities
   - in *csrKeyAlgorithm* - key pair algorithm
   - out *csrKeyID* - key pair ID
2. ONVIF Client selects signature algorithm by following the procedure mentioned in Annex A.8 with the following input and output parameters:
   - in *cap* - DUT capabilities
   - in csrKeyAlgorithm - key pair algorithm
   - out *caSignatureAlgorithm* - signature algorithm
3. ONVIF Client invokes **CreatePKCS10CSR** with parameters
   - Subject := *subject* (see Annex A.15)

- KeyID := *csrKeyID*
- CSRAttribute skipped
- SignatureAlgorithm.algorithm := *signatureAlgorithm*

4. The DUT responds with **CreatePKCS10CSRResponse** message with parameters
   - PKCS10CSR =: *PKCS10request*
5. ONVIF Client extracts public key *publicKey* from *PKCS10request*.

**Procedure Result:**

**PASS –**

- DUT passes all assertions.

**FAIL –**

- DUT did not send **CreatePKCS10CSRResponse** message.