Universal Serial Bus Type-C™ Authentication
Functional Test Specification for Authentication Responders

Date: September 19, 2017

Revision: 0.9

Revision History

Revision	**Issue Date**	**Comments**
0.9	September 19, 2017	Initial draft

Significant Contributors:	
Intel	Abdul Ismail
Intel	Stephanie Wallick
Intel	Enrique Fernandez
MicroChip	Richard Petrie
Renesas	Dan Aoki
Renesas	Kiichi Muto
Renesas	Philip Leung
Renesas	Bob Dunstan
Renesas	Toshifumi Yamaoka
Specwerkz	Diane Rose
Specwerkz	Søren Petersen

Contents

Introduction

These tests check that a USB Type-C™ Authentication Responder is compliant to the USB Type-C™ Authentication Specification assertions and functional requirements.

Terminology

The following table describes the terms used in this document.

ACB	Authentication Compliance Bridge – test equipment used to transfer authentication messages between an RVS and UUT. Authentication Compliance Bridge specification is available at TBD.
CVT	Certificate Verification Tool – test equipment capable of parsing certificates and verifying certificate fields.
UUT	Unit Under Test – the Authentication Responder that subject to the tests defined in this document.
RVS	Authentication Responder Verification System – test equipment capable of performing the tests defined in this document.

Assertions

Compliance criteria are provided as a list of assertions that describe specific characteristics or behaviors that must be met. Assertions are organized according to the section of the USB Type-C™ Authentication Specification from which they were derived. Each assertion provides a reference to the specific test description(s) where the assertion is tested.

Each assertion is formatted as follows:

Assertion #	Test #	Assertion Description

Assertion#: Unique identifier for each assertion. The identifier is in the form USBAUTH_SPEC_SECTION_NUMBER#X, where X is a unique integer for a requirement in that section.

Assertion Description: Specific requirement from the USB Type-C™ Authentication Specification

Test #: A label that identifies which test (if any) is used to test an assertion. Test # can have one of the following values:

NT	This item is not explicitly tested in a test description. Items can be labeled NT for several reasons – including items that are not testable, not important to test for interoperability, or are indirectly tested by other operations performed by the compliance test.
X.X	This item is covered by the test described in test description X.X in this specification.
IOP	This assertion is verified by the USB Type-C™ Authentication Interoperability Test Suite.
BC	This assertion is applied as a background check in all test descriptions.
TBD	This assertion will be tested in a later phase of compliance testing.

The following Table presents the USB Type-C™ Authentication Specification assertions.

Assertion #	Test #	Assertion Description
1.5.2.8 Reserved		
1.5.2.8#1	NT	The use and interpretation of reserved bits, bytes, words, and code values may be specified by future extensions to this specification and, unless otherwise stated, shall not be utilized or adapted by vendor implementation.
1.5.2.8#2	TD 1.2 General Test Procedure	A Reserved bit, byte, word, or field shall be set to zero by the sender and shall be ignored by the receiver.
1.5.2.8#3	General Test Procedure	Reserved field values shall not be sent by the sender and, if received, shall be ignored by the receiver.
3 Authentication Architecture		
3.1 Certificates		
3.1.1 Format		
3.1.1#1	TD 1.3	All Certificates shall use the X509v3 ASN.1 structure.

Assertion #	Test #	Assertion Description
3.1.1#2	TD 1.3	All Certificates shall use binary DER encoding for ASN.1.
3.1.1#3	TD 1.3	All Certificates shall use ECDSA Using the NIST P256, secp256r1 curve, uncompressed format for digital signing of certificates and Authentication Messages.
3.1.1#4	TD 1.1 TD 1.3	All Certificates shall use SHA256 for all cryptographic hashes.
3.1.1#5	TD 1.3	Leaf certificates shall not exceed *MaxLeafCertSize* in length.
3.1.1#6	TD 1.3	Intermediate Certificates shall not exceed *MaxIntermediateCertSize* in length.
3.1.2 Textual Format		
3.1.2#1	TD 1.3	All textual ASN.1 objects contained within Certificates, including DirectoryString, GeneralName, and DisplayText, shall be specified as either a UTF8String, PrintableString, or IA5String.
3.1.2#2	TD 1.3	The length of any textual object shall not exceed 64 bytes excluding the DER type and DER length encoding.
3.1.3 Attributes and Extensions		
3.1.3.1 Distinguished Name		
3.1.3.1#1	NT	A Certificate Authority shall not issue Certificates with the same distinguished name to different Entities.
3.1.3.1.1 Common Name (OID 2.5.4.3)		
3.1.3.1.1#1	TD 1.3	The common name attribute shall appear in every Certificate.
3.1.3.1.1#2	TD 1.3	The common name attribute shall contain a string matching one of the following three patterns: “USB::” “USB:<vid>:” “USB:<vid>:<pid>”
3.1.3.1.1#3	TD 1.3	When present, <vid> and <pid> shall be left zero padded and big endian.
3.1.3.1.1#4	TD 1.3	Uppercase letters shall not be used in the hex encoding of a VID or PID.
3.1.3.1.1#5	TD 1.3	The common name attribute in the Leaf Certificate of a Certificate Chain shall contain both a VID and a PID.
3.1.3.1.1#6	TD 1.3	If a VID value appears in a Certificate in the Chain, then the same VID value shall be used in all subsequent Certificates.
3.1.3.1.1#7	TD 1.3	If a PID value appears in a Certificate in the Chain, then the same PID value shall be used in all subsequent Certificates.
3.1.3.1.2 Organization Name (OID 2.5.4.10)		
3.1.3.1.2#1	NT	The organization name attribute shall be present in a Root Certificate.
3.1.3.1.2#2	TD 1.3	When present, the organization name attribute shall contain the human-readable name of the organization that owns the private key that corresponds to the Certificate.
3.1.3.1.3 Serial Number (OID 2.5.4.5)		
3.1.3.2 Basic Constraints (OID 2.5.29.19)		

Assertion #	Test #	Assertion Description
3.1.3.2#1	TD 1.3	The basic constraints extension shall be present and marked as critical.
3.1.3.2#2	TD 1.3	The cA component of the basic constraints exception shall be false in a Leaf Certificate.
3.1.3.2#3	TD 1.3	The cA component of the basic constraints exception shall be true for a non-Leaf Certificate.
3.1.3.2#4	TD 1.3	No other component of the basic constraints exception shall be included.
3.1.3.3 Key Usage (OID 2.5.29.15)		
3.1.3.3#1	TD 1.3	The key usage extension shall be present.
3.1.3.3#2	TD 1.3	Leaf Certificates shall have the digitalSignature bit of the key usage extension set, and all other bits cleared.
3.1.3.3#3	TD 1.3	Non-Leaf Certificates shall have the keyCertSign bit of the key usage extension set, may optionally have the cRLSign bit of the key usage extension set, and shall have all other bits cleared.
3.1.3.4 Extended Key Usage (OID 2.5.29.37)		
3.1.3.4#1	TD 1.3	The extended key usage extension shall be present and marked as critical.
3.1.3.4#2	TD 1.3	The extended key usage extension shall contain the USB-IF issued OID 2.23.145.1.1 for the “USB-Auth” extended key usage.
3.1.3.5 Validity		
3.1.3.5#1	NT/checklist	Certificate notBefore and notAfter validity times shall be ignored.
3.1.3.5#2	TD 1.3	Certificate notBefore and notAfter validity times shall be specified using either ASN.1 GeneralizedTime for any year, or ASN.1 UTCTime for years prior to 2050.
3.1.3.6 USB-IF ACD (OID 2.23.145.1.2)		
3.1.3.6#1	TD 1.3	Leaf certificates shall contain the USB-IF ACD extension.
3.1.3.6#2	TD 1.3	Non-Leaf certificates shall not contain the USB-IF ACD extension.
3.1.3.7 Additional Attributes and Extensions		
3.2 Certificate Chains		
3.2#1	TD 1.4	A Certificate Chain shall not exceed *MaxCertChainSize* bytes.
3.2#2	TD 1.4	Each slot shall either be empty or contain one complete certificate chain.
3.2#3	NT/checklist	A Product shall not contain more than 8 slots.
3.2#4	TD 1.4	Slots 0 through 3 shall only be used for Certificate Chains rooted with a USB-IF Root Certificate and shall not contain any other Certificate Chains.
3.2.1 Provisioning		
3.3 Private Keys		
3.3#1	TD 4.1	All private keys in a Product shall be different from one another.
3.3#2	NT/checklist	All private keys in a Product shall be generated, provisioned, and stored in a manner that adequately protects the confidentiality of the key.
3.3#3	NT/checklist	A private key used by one Product shall not be used by any other Products.

Assertion #	Test #	Assertion Description
4 Authentication Protocol		
4#1	TD 1.1 TD 1.4 TD 1.5	A Product shall not act as an Authentication Responder unless it contains a Certificate Chain in slot 0.
4.1 Digest Query		
4.1#1	TBD	If an error condition is encountered, the Authentication Responder shall respond with the appropriate ERROR Response.
4.1#2	TD 1.1 TD 1.3 TD 1.4 TD 1.9	If an error condition is not encountered, the Authentication Responder shall respond with a DIGESTS Response.
4.2 Certificate Chain Read		
4.2#1	TD 1.6	If an Authentication Responder receives a GET_CERTIFICATE request that targets an offset that is outside the Certificate Chain (i.e. offset > length) or attempts to read beyond the length of the target Certificate Chain (i.e. (offset + length) > Certificate Chain length), then the Authentication Responder shall return an ERROR Authentication Response with *Param1* set to INVALID_REQUEST and *Param2* set to 00h.
4.2#2	TD 1.4	If an error condition is encountered, the Authentication Responder shall respond with the appropriate ERROR Response.
4.2#3	TD 1.1 TD 1.3 TD 1.4 TD 1.9	If an error condition is not encountered, the Authentication Responder shall respond with a CERTIFICATE Response.
4.3 Authentication Challenge		
4.3#1	TD 1.4	If an error condition is encountered, the Authentication Responder shall respond with the appropriate ERROR Response.
4.3#2	TD 1.4 TD 1.5 TD 1.9	If an error condition is not encountered, the Authentication Responder shall respond with a CHALLENGE_AUTH Response
4.4 Errors and Alerts		
4.4.1 Invalid Requests		
4.4.1#1	TD 1.7	If an Authentication Responder receives an Authentication Request with one or more invalid fields, it shall respond to that Authentication Request with an ERROR Response that has *Param1* set to INVALID_REQUEST and *Param2* set to 00h.
4.4.2 Unsupported Protocol Version		
4.4.2#1	TD 1.8	If an Authentication Responder receives an Authentication Request that contains an unsupported Security Protocol Version in the *ProtocolVersion* field, it shall respond to that Authentication Request with an ERROR Response that has *ProtocolVersion* set to the minimum Security Protocol Version it supports, *Param1* set to UNSUPPORTED_PROTOCOL, and *Param2* set to the maximum Security Protocol Version it supports.
4.4.3 Busy		

Assertion #	Test #	Assertion Description
4.4.3#1	General Test Procedure	If an Authentication Responder receives an Authentication Request but is unable to meet either the timing requirements listed in Section 6.4 (for PD Products) or Section 7.4 (for USB Products), it shall respond to that Authentication Request with an ERROR Response that has *Param1* set to BUSY and *Param2* set to 00h
4.4.4 Unspecified		
4.4.4#1	General Test Procedure	If an Authentication Responder, upon receiving an Authentication Request, encounters an error that is not covered by any condition that is not otherwise covered in section 4.4 of the USB Type-C™ Authentication specification, it shall respond to that Authentication Request with an ERROR Response that has *Param1* set to UNSPECIFIED and *Param2* set to 00h.
5 Authentication Messages		
5.1 Header		
5.1#1	IOP	All Authentication Messages shall start with the 4-byte header defined in Table 5-1 of the USB Type-C™ Authentication specification.
5.1.1 USB Type-C Authentication Protocol Version		
5.1.1#1	General Test Procedure	A Product shall not use a USB Type-C™ Authentication Protocol Version value corresponding to a specification revision that it does not support.
5.1.2 Message Type		
5.1.2#1	General Test Procedure	The Message Type field shall contain one of the Authentication Message Types listed in Tables 5-3 or 5-9 of the USB Type-C™ Authentication specification.
5.1.3 Param1		
5.1.4 Param2		
5.2 Authentication Requests		
5.2#1	NT	Authentication Message types 00h - 7Fh shall only be used for Authentication Responses.
5.2.1 GET_DIGESTS		
5.2.2 GET_CERTIFICATE		
5.2.2#1	NT	The value in the *Param1* field shall be between 0 and 7 inclusive.
5.2.3 CHALLENGE		
5.3 Authentication Responses		
5.3#1	General Test Procedure	Authentication Message types 80h - FFh shall only be used for Authentication Requests.
5.3.1 DIGESTS		
5.3.1#1	TD 1.1	The *Capabilities* Field shall be set to 01h.
5.3.1#2	TD 1.4	The bit in position K of the *Param2* field shall be set if and only if slot number K contains a Certificate Chain for the protocol version in the *ProtocolVersion* field. (Bit 0 is the least significant bit of the byte.)
5.3.1#3	TD 1.1	The number of digests returned shall be equal to the number of bits set in *Param2*.
5.3.1#4	TD 1.1	The digests shall be returned in order of increasing slot number.
5.3.2 CERTIFICATE		

Assertion #	**Test #**	**Assertion Description**
5.3.3 CHALLENGE_AUTH		
5.3.3#1	TD 1.5	The value in the *Param1* field shall contain the Slot number in the *Param1* field of the corresponding CHALLENGE Request.
5.3.3#2	TD 1.4	The bit in position K of the Param2 field shall be set if and only if slot number K contains a Certificate Chain for the protocol version in the *ProtocolVersion* field. (Bit 0 is the least significant bit of the byte.)
5.3.3.1 Signature		
5.3.4 ERROR		
6 Authentication of PD Products		
6.1 Transfers less than or equal to *MaxExtendedMsgLen*		
6.2 Transfers greater than *MaxExtendedMsgLen*		
6.2#1	NT	An Authentication Initiator shall break up a security transfer into Authentication Messages that don't exceed *MaxExtendedMsgLen*.
6.3 Examples		
6.4 Timing Requirements for PD Security Extended Messages		
6.4.1 Authentication Initiator		
6.4.2 Authentication Responder		
6.4.2#1	PD Timing Test	An Authentication Responder shall send an Authentication Response within *tDigestSent* of receiving a GET_DIGESTS Authentication Request.
6.4.2#2	PD Timing Test	An Authentication Responder shall send an Authentication Response within *tCertSent* of receiving a GET_CERTIFICATE Authentication Request.
6.4.2#3	PD Timing Test	An Authentication Responder shall send an Authentication Response within *tChallengeAuthSent* of receiving a CHALLENGE Authentication Request.
6.5 Context Hash		
6.5#1	TD 1.5	The Context Hash field in a CHALLENGE_AUTH Authentication Response shall be zero for PD Sources, Sinks and Cable Plugs.
6.5#2	TD 1.5	The Context Hash field in a CHALLENGE_AUTH Authentication Response shall be zero for PD Alternate Mode devices.
7 Authentication of USB Products		
7#1	NT/checklist	A USB Device shall not act as an Authentication Initiator.
7.1 Descriptors		
7.1.1 Authentication Capability Descriptor		
7.1.1#1	TD 3.1	The Authentication Capability Descriptor shall be returned as part of the BOS Descriptor set for a USB Device that supports Authentication.
7.1.1#2	TD 3.1	Bit 0 in the *bmAttributes* field of the Authentication Capability Descriptor shall be set to 1 if firmware can be updated. Otherwise, set to zero.
7.1.1#3	TD 3.1	Bit 1 in the *bmAttributes* field of the Authentication Capability Descriptor shall be set to 1 to indicate that Device changes interfaces when updated. Otherwise, set to zero.

Assertion #	Test #	Assertion Description
7.1.1#4	TD 3.1	The *bcdProtocolVersion* in an Authentication Capability Descriptor shall be set to USB Type-C™ Authentication Protocol Version (01h).
7.1.1#5	TD 3.1	The *bcdCapability* field in an Authentication Capability Descriptor shall be set to 01h.
7.2 Mapping Authentication Messages to USB		
7.2.1 Authentication IN		
7.2.1#1	IOP	A Device shall respond to an Authentication IN Request when in the Address state.
7.2.1#2	TD 3.2	A Device shall respond to an Authentication IN Request with a Request Error when in the Configured state.
7.2.1#3	TD 3.2	A Device shall respond to an Authentication IN Request with a Request Error when in the Configured state.
7.2.1#4	TD 3.4	A USB Device shall respond with a Request Error if *wLength* for a particular Response type does not match the values set forth in this section.
7.2.2 Authentication OUT		
7.2.2#1	IOP	A Device shall respond to an Authentication OUT Request when in the Address state.
7.2.2#2	TD 3.2	A Device shall respond to an Authentication OUT Request with a Request Error when in the Configured state.
7.2.2#3	TD 3.2	A Device shall respond to an Authentication IN Request with a Request Error when in the Configured state.
7.2.2#4	TD 3.4	A USB Device shall respond with a Request Error if *wLength* for a particular Response type does not match the values set forth in this section.
7.3 Authentication Protocol		
7.3.1 Digest Query		
7.3.2 Certificate Read		
7.3.3 Authentication Challenge		
7.3.4 Errors		
7.3.4#1	IOP	If a USB Device encounters an Authentication-related error condition during an AUTH_IN control transfer, it shall respond with an ERROR Response.
7.3.4#2	IOP	If a USB Device encounters an Authentication-related error condition during an AUTH_OUT control transfer, it shall respond to the next AUTH_IN control transfer with an ERROR Response.
7.4 Timing Requirements for USB		
7.4#1	IOP	All Authentication Message exchanges over USB shall follow the timing for control transfers set forth in USB2.0 and USB3.1.
7.4.1 USB Host Timing Requirements		
7.4.1#1	NT	If a USB Host does not receive an Authentication Response within *tDigestIN* (100 ms) of sending a GET_DIGESTS Authentication Request, it is considered an error.

Assertion #	Test #	Assertion Description
7.4.1#2	NT	If a USB Host does not receive an ACK within *tCertOUT* (100 ms) of sending a GET_CERTIFICATE Authentication Request to an Authentication Responder, it is considered an error.
7.4.1#3	NT	If a USB Host does not receive an Authentication Response within *tCertIN* (500 ms) of sending a CERTIFICATE Authentication Request to an Authentication Responder, it is considered an error.
7.4.1#4	NT	If a USB Host does not receive an ACK within *tChallengeOUT* (100 ms) of sending a CHALLENGE Authentication Request to an Authentication Responder, it is considered an error.
7.4.1#5	NT	If a USB Host does not receive an Authentication Response within *tChallengeIN* (600 ms) of sending a CHALLENGE_AUTH Authentication Request to an Authentication Responder, it is considered an error.
7.4.2 USB Device Timing Requirements		
7.4.2#1	USB Timing Test	A USB Device shall respond to an Authentication Initiator within *tDigestSent* (95 ms) of receiving an AUTH_IN control transfer carrying a GET_DIGESTS Authentication Request.
7.4.2#2	USB Timing Test	A USB Device shall ACK an AUTH_OUT control transfer carrying a GET_CERTIFICATE Request within *tCertACK* (95 ms) of receiving it.
7.4.2#3	USB Timing Test	A USB Device shall respond to an Authentication Initiator within *tCertSent* (495 ms) of receiving an AUTH_IN control transfer carrying a CERTIFICATE Authentication Request.
7.4.2#4	USB Timing Test	A USB Device shall ACK an AUTH_OUT control transfer carrying a CHALLENGE Request within *tChallengeACK* (95 ms) of receiving it.
7.4.2#5	USB Timing Test	A USB Device shall respond to an Authentication Initiator within *tChallengeAuthSent* (595 ms) of receiving an AUTH_IN control transfer carrying a CHALLENGE_AUTH Authentication Request.
7.5 Context Hash		
7.5#1	TD 3.3	The Context Hash field in a CHALLENGE AUTH Authentication Response shall contain a 32-byte SHA256 hash of the following USB Descriptor data (as defined in USB2.0 and USB3.1) for current operating speed, concatenated together in following order: 1) Device Descriptor 2) Complete BOS Descriptor (if present) 3) Complete Configuration 1 Descriptor 4) Complete Configuration 2 Descriptor (if present) 5) ... 6) Complete Configuration N Descriptor (if present).
7.5#2	TD 3.3	The contents of each descriptor used to create the hash in the Context Hash shall match that which the device presents during enumeration at the USB Device’s current connection.
8 Protocol Constants		
A ACD		
A.1 ACD Formatting		
A.1#1	TD 1.3	No TLV type shall occur more than once.
A.1#2	TD 1.3	Each TLV shall appear in increasing order by TLV value.
A.1.1 Version TLV		

Assertion #	Test #	Assertion Description
A.1.2 XID TLV		
A.1.3 Power Source Capabilities TLV		
A.1.4 Power Source Certifications TLV		
A.1.5 Cable Capabilities TLV		
A.1.6 Security Description TLV		
A.1.6#1	TD 1.3	The certifications claimed in the Security Description TLV shall be relevant and appropriate to the security functions used in the product.
A.1.6#2	TD 1.3	The IC Vendor Identified field in a Security Description TLV shall contain either the USB-IF assigned VID that identifies that IC vendor or zero if not used.
A.1.7 Playpen TLV		
A.1.7#1	TD 1.3	The Playpen TLV shall not be used or interpreted by any Products.
A.1.8 Vendor Extension TLV		
A.1.8#1	TD 1.3	The first two bytes of the Data field in a Vendor Extension TLV shall contain the Vendor ID of the vendor defining the field.
A.1.9 Extension TLV		
A.1.9#1	TD 1.3	The Extension TLV shall not be used.
A.2 ACD for a PD Product		
A.2#1	TD 1.3	VERSION, XID, POWER_SOURCE_CAPABILITIES, and SECURITY_DESCRIPTION TLVs shall be present in the ACD of a PD Source or PD Sink.
A.2#2	TD 1.3	The CABLE_CAPABILITIES TLV shall not be used in the ACD of a PD Source or PD Sink.
A.2#3	TD 1.3	VERSION, XID, CABLE_CAPABILITIES, and SECURITY_DESCRIPTION TLVs shall be present in the ACD of a USB Type-C™ Cable.
A.2#4	TD 1.3	POWER_SOURCE_CAPABILITIES and POWER_SOURCE_CERTIFICATIONS TLVs shall not be used in the ACD of a USB Type-C™ Cable.
A.3 ACD for a USB Product		
A.3#1	TD 1.3	VERSION and SECURITY_DESCRIPTION TLVs shall be present in the ACD of a USB Product.
A.3#2	TD 1.3	CABLE_CAPABILITIES TLV shall not be used in the ACD of a USB Product.
B Cryptographic Examples		
B.1 Example Authentication Sequence		
B.2 Example Certificate Chain Topology		
B.2.1 Certificate Chain		
B.2.1.1 Intermediate Certificate		
B.2.1.2 Leaf Certificate		

Assertion #	**Test #**	**Assertion Description**
B.2.2 Root Certificate		
B.2.3 Key Pairs		
B.2.3.1 Root Key Pair		
B.2.3.2 Intermediate Key Pair		
B.2.3.3 Leaf Key Pair		
B.3 Example Authentication Signature Verification		
B.3.1 CHALLENGE Request		
B.3.2 CHALLENGE_AUTH Response		
C Potential Attack Vectors		

Test Requirements

Software

The RVS is a Windows-based software solution capable of testing the assertions defined in this document. The RVS appears to a UUT as an Authentication Initiator. The RVS is capable of testing a UUT over both the PD bus and USB data bus depending on the underlying hardware.

Hardware

PD

The RVS uses an Authentication Compliance Bridge (ACB) device to communicate with a UUT over the PD bus.

USB

TBD

Timing

Timing is measured in Milliseconds (ms)

Certificates

At a minimum, a UUT must contain a certificate chain in slot 0 that is rooted with a USB-IF Root certificate. A UUT may also contain certificate chains in slots 1 through 7.

Unless specified otherwise in the test description, all certificate chain slots are subject to compliance testing.

Test Setup

PD

PD Products are tested first with chunking (i.e. ACB indicates to UUT that it does not support Unchunked PD Extended Messages). If UUT supports unchunked PD messages, UUT is also tested using unchunked PD messages (i.e. both ACB and UUT indicate support for Unchunked PD Extended Messages).

The figure below show test setups for a UUT as SOP, SOP’, or SOP”. Note that the test setups for SOP’ and SOP” require either a PD3.0 Sink or USB Type-C-to-Type-A Adapter downstream in order to establish PD communications.

Figure 1 UUT as SOP

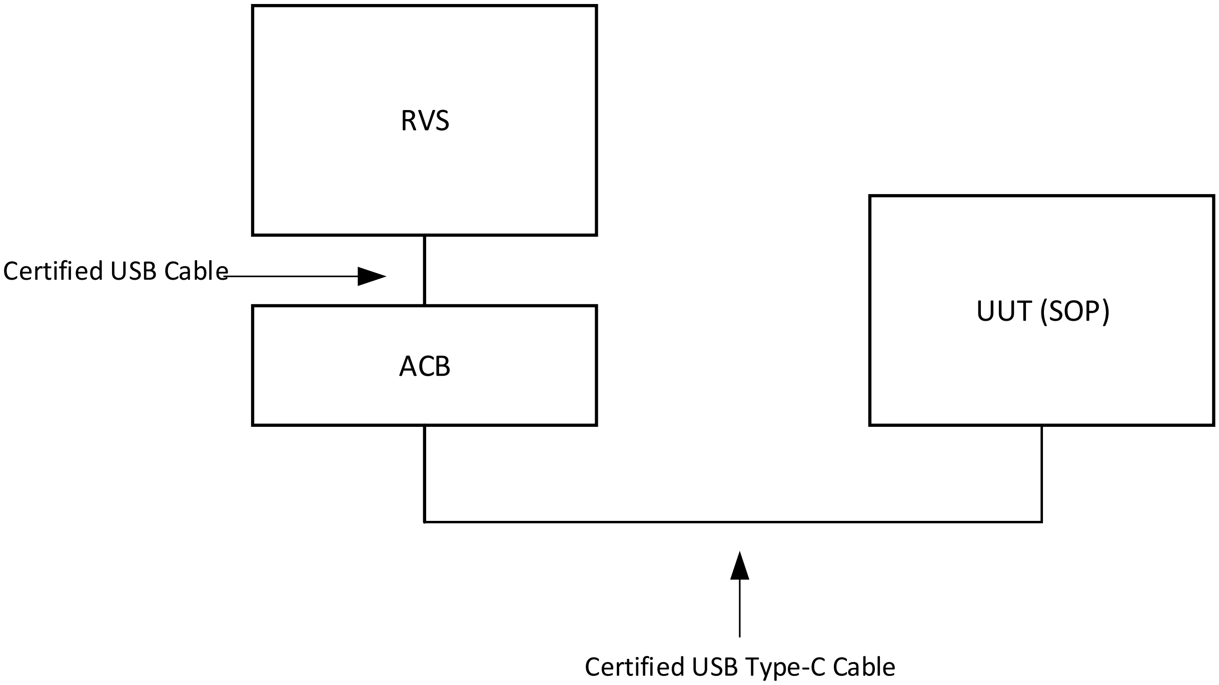


Figure 2 UUT as SOP’

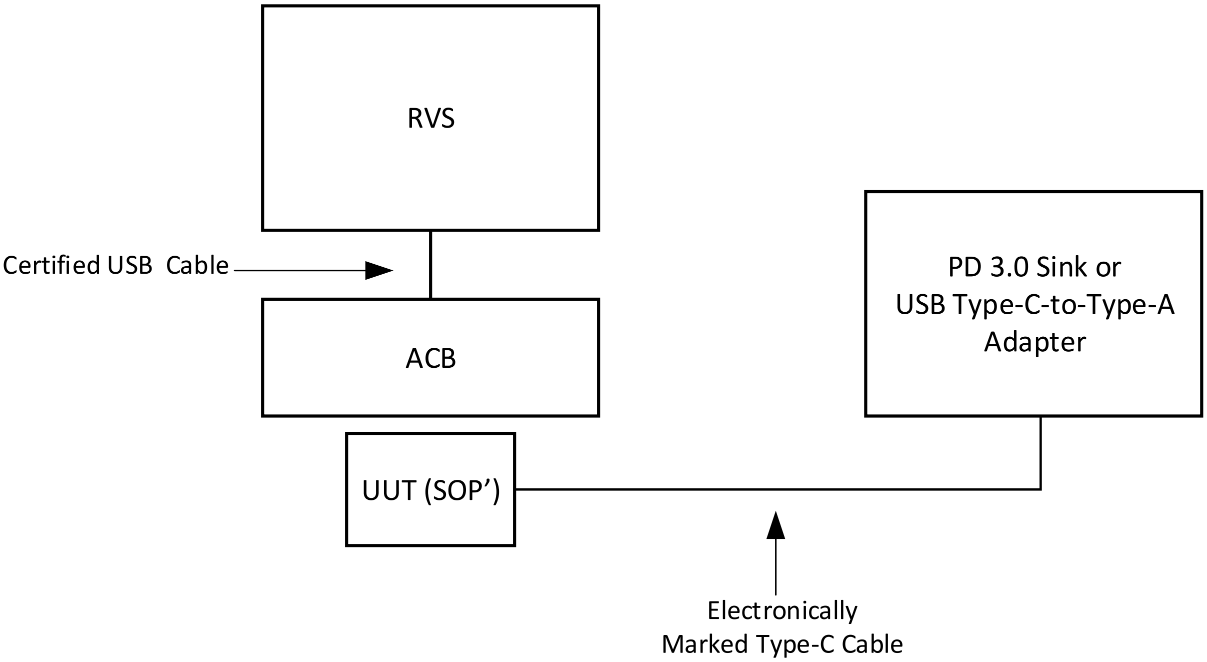
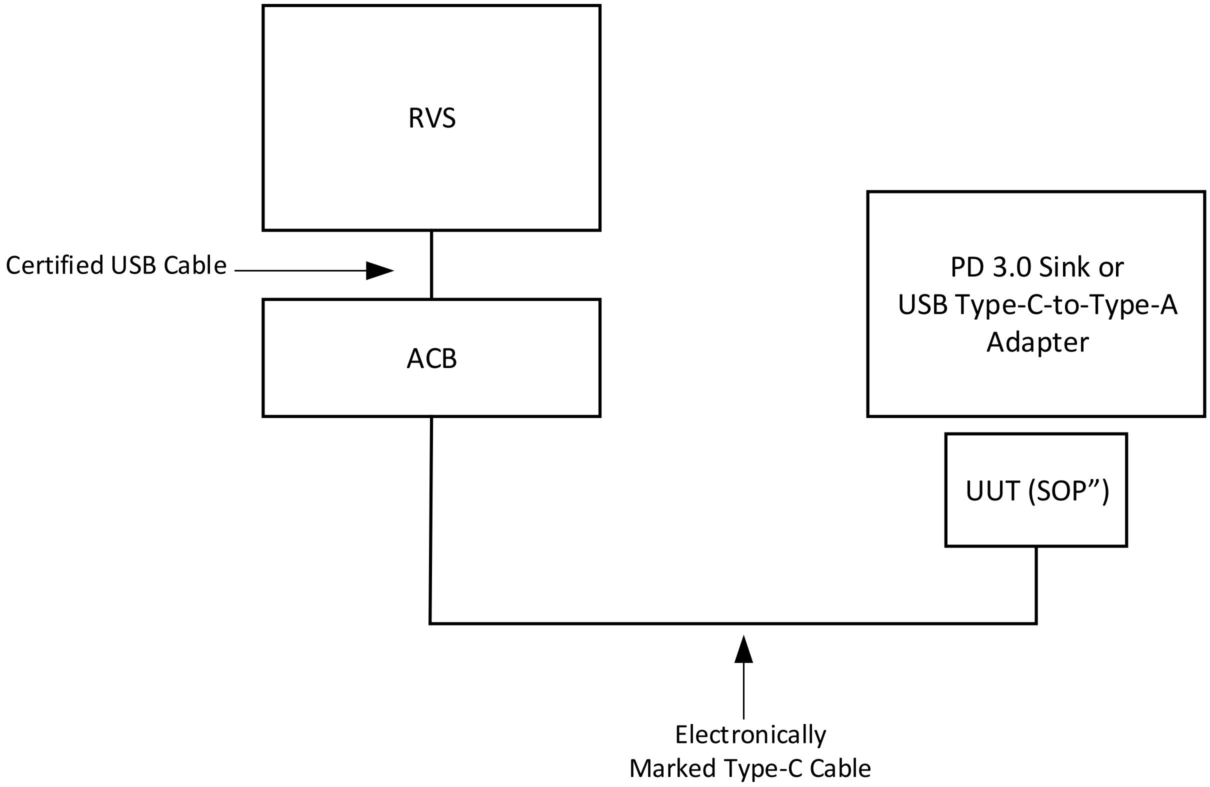


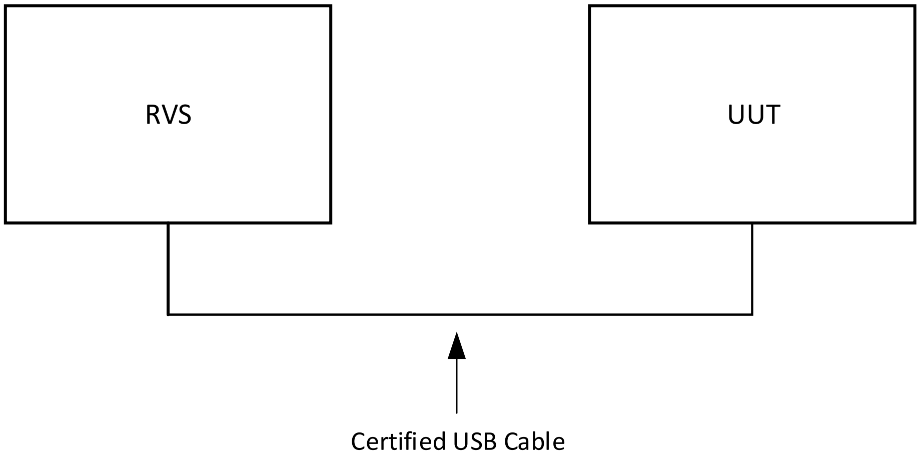
Figure 3 UUT as SOP”



USB

Unless otherwise specified, compliance tests are run with a UUT in the Addressed state. The figure below shows the test setup for a UUT that is a USB device.

Figure 4 UUT as USB Device



Vendor Checklist

Product Information

Field	**--All fields must be filled in--**
Date	
Vendor Name	
Vendor ID (VID) Assigned by USB-IF	
Vendor Street Address	
Vendor City, State, Postal Code	
Vendor Country	
Vendor Phone Number	
Vendor Contact Name	
Vendor Contact Title	
Vendor Contact Email Address	
Product Name	
Product Model Number	
Product Revision Level	
Certifications Claimed in Security Description TLV	
Name of Preparer	
Signature of Preparer	

Vendor Attestations

ID	Question	Response	Reference Assertion
A1	Product ignores *notBefore* and *notAfter* validity times in all certificates	☐ yes ☐ no	3.1.3.5#1
A2	Product does not have more than 8 certificate chain slots	☐ yes ☐ no	3.2#3
A3	Product keys were generated, provisioned, and stored in a manner that adequately protects them.	☐ yes ☐ no	3.3#2
A4	Product keys are not used by any other Products	☐ yes ☐ no	3.3#3
A5	If Product is a USB device, it does not act as an Authentication Initiator	☐ yes ☐ no ☐ N/A	7#1

Tests

Authentication Responder Tests

Application: All Authentication Responders participate in the tests defined in this section.

Pass/Fail Criteria: The UUT passes a test if all steps are successfully executed and RVS verifies all listed criteria. Unless otherwise specified, each test assumes a non-error response from the UUT. If the UUT sends an unexpected ERROR Response, the UUT fails that particular test.

General Test Procedure

The following test steps are run in conjunction with the tests defined in this chapter anytime an Authentication Response is received from the UUT.

1. For any Authentication Message from the UUT, RVS verifies that:
   a. The *ProtocolVersion* field in set to 01h (5.1.1#1)
   b. The *ProtocolVersion* field contains neither 00h nor values 02h through FFh (1.5.2.8#3)
   c. The *MessageType* set to either 01h, 02h, 03h, or 7Fh and no other value (5.1.2#1, 5.3#1)
   d. The *MessageType* field contains neither 00h nor values 04h through 7Eh (1.5.2.8#3)
2. For each CERTIFICATE Response from the UUT, the RVS verifies that the *Param2* field is 00h (reserved) (1.5.2.8#2)
3. For any ERROR Responses from the UUT that have *Param1* set to INVALID_REQUEST, BUSY, or UNSPECIFIED, RVS verifies that the *Param2* field is 00h (reserved) (1.5.2.8#2)
4. For any ERROR Responses from the UUT, RVS verifies that *Param1* (Error Code) contains neither 00h nor values 05h through EFh (1.5.2.8#3)
5. If UUT sends an ERROR Response with *Param1* set to UNSPECIFIED, RVS verifies that the ERROR Response has *Param2* set to 00h (4.4.4#1)
6. If UUT sends an ERROR Response with *Param1* set to BUSY, RVS verifies that the ERROR Response has *Param2* set to 00h (4.4.3#1)

TD 1.1 Digest Query and Certificate Chain Read Test

A. Purpose:
   1. Verify that the UUT responds to a GET_DIGEST Request
   2. Verify that a DIGEST Response has the correct fields and values
   3. Verify that the contents of a DIGEST Response do not vary
   4. Verify that the UUT responds to a GET_CERTIFICATE Request
   5. Verify that Certificate Chains do not vary

B. Asserts:
   1. 3.1.1#4
   2. 4#1
   3. 4.1#2
   4. 4.2#3
   5. 5.3.1#1
   6. 5.3.1#3
   7. 5.3.1#4

C. Procedure:
   1. RVS sends a GET_DIGESTS Request to the UUT

2. RVS verifies that the UUT responds with a DIGESTS Response (4.1#2)
3. RVS saves the Slot Mask from the DIGESTS response for further use.
4. RVS verifies that:
    a. The DIGESTS Response has bit 0 of the slot mask set to 1 (to indicate a valid certificate chain in slot 0) (4#1)
    b. The *Capabilities* field in the DIGESTS Response is set to 01h (5.3.1#1)
    c. The number of digests returned in the DIGESTS Response is equal to the number of bits set in *Param2* (slot mask) (5.3.1#3)
    d. The digests returned in the DIGESTS Response are in order of increasing slot number. (5.3.1#4)
5. RVS sends a series of GET_CERTIFICATE Requests to read the full Certificate Chains in each populated slot of the UUT as follows:
    a. RVS sends a GET_CERTIFICATE Request to the UUT with Offset set to 0 and Length set to 4.
    b. RVS verifies that the UUT responds with a CERTIFICATE Response containing a response header and the first 4 bytes of the Certificate Chain. (4.2#3)
        i. Save the *Length* field of the Certificate Chain for further use.
    c. RVS sends sufficient valid GET_CERTIFICATE requests to read the remaining contents of the Certificate Chain (as determined by the Certificate Chain *Length* field) in 256 byte chunks, other than possibly the last request which shall request the smaller of 256 and the number of bytes remaining in the Certificate Chain.
        i. *Note: a valid GET_CERTIFICATE request is one in which* Offset + Length *is not greater than the Length field of the Certificate Chain.*
        ii. RVS verifies that UUT responds to each request with a CERTIFICATE Response containing a response header followed by the request *Length* field number of bytes of the Certificate Chain. (4.2#3)
6. RVS creates a SHA256 hash for each Certificate Chain.
7. The RVS verifies that each digest returned in the DIGESTS Response matches the 32-byte SHA256 hash of the Certificate Chain in its corresponding slot (3.1.1#4)
8. The RVS sends a Second GET_DIGESTS Request to the UUT.
9. The RVS verifies that the second DIGESTs Response returned by the UUT is identical to the first DIGESTS response sent by the UUT.
10. Unplug/disconnect the UUT, then replug/reconnect UUT to RVS
11. RVS sends a third GET_DIGESTS Request to the UUT
12. RVS sends a second series of GET_CERTIFICATE Requests to read the full Certificate Chains in each populated slot of the UUT.
13. RVS creates a SHA256 hash for each Certificate Chain.
14. The RVS verifies that the SHA256 hash of each Certificate Chain read during the second series of GET_CERTIFICATE Requests matches the SHA256 hash of the Certificate Chains read during the first series of GET_CERTIFICATE Requests (i.e. first and second SHA256 hash of certificate chain in Slot 0 are identical, first and second SHA256 hash of certificate chain in Slot 1 are identical, etc.).
15. The RVS verifies that the third DIGESTs Response returned by the UUT is identical to the first DIGESTS response sent by the UUT.

TD 1.2 Reserved Fields Ignored Test

A. Purpose:
    1. Verify that reserved fields are ignored
B. Asserts:
    1. 1.5.2.8#2
C. Procedure:
    1. RVS sends a GET_DIGESTS Request to the UUT.
    2. RVS saves the Slot Mask from the DIGESTS response for further use.

3. RVS sends a GET_DIGESTS Request to the UUT
    a. GET_DIGESTS Request has the *Param1* field (reserved) set to F0h.
    b. GET_DIGESTS Request has the *Param2* field (reserved) set to 0Fh.
4. RVS verifies that the UUT responds with a DIGESTS Response (1.5.2.8#2).
5. RVS repeats the following steps for each populated slot:
    a. RVS sends a GET_CERTIFICATE Request to the UUT targeting the first 36 bytes of the certificate chain:
        i. GET_CERTIFICATE Request has the *Param1* field set to the target slot number.
        ii. GET_CERTIFICATE Request has the *Param2* field (reserved) set to FFh.
        iii. GET_CERTIFICATE Request has the *Offset* field set to 0 to target the start of the chain.
        iv. GET_CERTIFICATE Request has the *Length* field set to 36 to read 36 bytes of the chain.
    b. RVS verifies that the UUT responds with a CERTIFICATE Response (1.5.2.8#2).
    c. RVS sends a CHALLENGE Request to the UUT:
        i. CHALLENGE Request has the *Param1* field set to the target slot number.
        ii. CHALLENGE Request has the *Param2* field (reserved) set to FFh
    d. RVS verifies that the UUT responds with a CHALLENGE_AUTH Response (1.5.2.8#2).

TD 1.3 Certificate Format Test

A. Purpose:
    1. Verify that Certificates are in X509v3 format
    2. Verify that Certificates use ASN.1 with binary DER encoding
    3. Verify that Certificates use SHA256 for all cryptographic hashes
    4. Verify that Certificates do not exceed the maximum length
    5. Verify that Certificates contain the required fields and values
    6. Verify that Certificates do not contain prohibited fields and values
    7. Verify that the ACD in a Certificate is properly formatted

B. Asserts:
    1. 3.1.1#1
    2. 3.1.1#2
    3. 3.1.1#3
    4. 3.1.1#4
    5. 3.1.1#5
    6. 3.1.1#6
    7. 3.1.2#1
    8. 3.1.2#2
    9. 3.1.3.1.1#1
    10. 3.1.3.1.1#2
    11. 3.1.3.1.1#3
    12. 3.1.3.1.1#4
    13. 3.1.3.1.1#5
    14. 3.1.3.1.1#6
    15. 3.1.3.1.1#7
    16. 3.1.3.1.2#2
    17. 3.1.3.2#1
    18. 3.1.3.2#2
    19. 3.1.3.2#3
    20. 3.1.3.2#4

21. 3.1.3.3#1
22. 3.1.3.3#2
23. 3.1.3.3#3
24. 3.1.3.3#4
25. 3.1.3.4#1
26. 3.1.3.4#2
27. 3.1.3.5#2
28. 3.1.3.6#1
29. 3.1.3.6#2
30. 4.1#2
31. 4.2#3
32. A.1#1
33. A.1#2
34. A.1.6#1
35. A.1.6#2
36. A.1.7#1
37. A.1.8#1
38. A.1.9#1
39. A.2#1
40. A.2#2
41. A.2#3
42. A.2#4
43. A.3#1
44. A.3#2

C. Procedure:

1. RVS sends a GET_DIGESTS Request to the UUT.
2. RVS verifies that the UUT responds with a DIGESTS response (4.1#2).
3. RVS records which Certificate Chains slots are populated (i.e. bit corresponding to slot is set in the slot mask returned from the UUT in the DIGESTS Response) and performs the following steps for each populated slot:
4. RVS reads the target Certificate Chain from the UUT
    a. RVS sends a series of GET_CERTIFICATE Requests that read the full length of the target Certificate Chain
    b. RVS verifies that the UUT responds to each GET_CERTIFICATE Request with a CERTIFICATE Response (4.2#3)
5. RVS passes the certificate chain to the CVT.
6. CVT parses the Certificate Chain.
7. For each non-Root Certificate in the Certificate Chain CVT verifies that:
    a. Certificate uses X509v3 ASN.1 structure (3.1.1#1)
    b. Certificate uses binary DER encoding for ASN.1 (3.1.1#2)
    c. Certificate was signed using ECDSA using NIST P256 secp256r1 curve with uncompressed format (3.1.1#3)
    d. If certificate is in Slot 0 through 3, verify that the certificate is signed using the key in the preceding certificate in the chain.
    e. If the certificate is in Slot 4 through 7 and is a second intermediate or leaf certificate, verify that the certificate is signed using the key in the preceding certificate in the chain.
    f. The length of any textual object is not more than 64 bytes (excluding DER type and DER length encoding) (3.1.2#2)
    g. Certificate contains a common name attribute (3.1.3.1.1#1)
    h. The common name attribute:
        i. Contains a string with one of the three patterns (excluding quotation marks): “USB::” , “USB:<vid>” , or “USB:<vid>:<pid>” (3.1.3.1.1#2)
        ii. If the string above contains a <vid> and/or <pid>, each is left zero padded and big endian (3.1.3.1.1#3)

      iii. If the string above contains a <vid> and/or <pid>, neither contains uppercase letters (3.1.3.1.1#4)
   i. In the common name attribute, the <vid> represents a 4-character lowercase hexadecimal string encoding the 16-bit VID of the UUT vendor (3.1.3.1.1#6)
   j. In the common name attribute:
      i. If the <pid> is not present in the preceding certificate, the <pid> is either not present or is represented by a 4-character lowercase hexadecimal string encoding the 16-bit PID of the UUT
      ii. If the <pid> is present in the preceding certificate, the current <pid> is identical (3.1.3.1.1#7)
   k. Certificate has the basic constraints extension (3.1.3.2#1)
   l. The basic constraints extension is marked as critical (3.1.3.2#1)
   m. Only the cA component is included in the basic constraints extension (3.1.3.2#4)
   n. Certificate has the key usage extension (3.1.3.3#1)
   o. Certificate has the extended key usage extension (3.1.3.4#1)
   p. The extended key usage extension is marked as critical (3.1.3.4#1)
   q. The extended key usage extension contains the value 2.23.145.1.1 (USB-ID issued OID for extended key usage) (3.1.3.4#2)
   r. notBefore and notAfter times in the Certificate are specified using either ASN.1 GeneralizedTime for any year, or ASN.1 UTCTime for years prior to 2050 (3.1.3.5#2)

8. For slots 0 through 3, CVT verifies that:
   a. The hash of the Root Certificate in the Certificate Chain matches the SHA256 hash of the USB-IF Root Certificate. (3.1.1#4)
9. For each Intermediate Certificate, CVT verifies that:
   a. Certificate does not exceed *MaxIntermediateCertSize* in length. (3.1.1#6)
   b. The cA component of the basic constraints extension is true (3.1.3.2#3)
   c. The keyCertSign bit is set in the key usage extension and all other bits except the cRLSign bit are cleared (3.1.3.3#3)
   d. Certificate does not contain the USB-IF ACD extension (3.1.3.6#2)
10. For the Leaf Certificate, CVT verifies that:
   a. Certificate does not exceed *MaxLeafCertSize* in length. (3.1.1#5)
   b. The common name attribute contains both a VID and a PID value (3.1.3.1.1#5)
   c. The cA component of the basic constraints extension is false (3.1.3.2#2)
   d. The digitalSignature bit in the key usage extension is set and all other bits are cleared (3.1.3.3#2)
   e. Certificate contains the USB-IF ACD extension (3.1.3.6#1)
   f. The ACD extension has:
      i. No more than one TLV of any given type (A.1#1)
      ii. Each TLV appearing in increasing order by TLV value (A.1#2)
   g. If UUT is a PD Source or PD Sink:
      i. CVT verifies that the ACD contains the following type TLVs: (A.2#1)

01	VERSION
02	XID
03	POWER_SOURCE_CAPABILITIES
04	SECURITY_DESCRIPTION

      ii. CVT verifies that the ACD does not contain a CABLE_CAPABILITIES TLV (A.2#2)
   h. If UUT is a USB Type-C™ Cable:
      i. CVT verifies that the ACD contains the following type TLVs: (A.2#3)

01	VERSION
02	XID
03	CABLE_CAPABILITIES
04	SECURITY_DESCRIPTION

        ii. CVT verifies that the ACD contains neither a POWER_SOURCE_CAPABILITIES TLV nor a POWER_SOURCE_CERTIFICATIONS TLV (A.2#4)
    i. If UUT is a USB Product:
        i. CVT verifies that the ACD contains both a VERSION and a SECURITY_DESCRIPTION TLV (A.3#1)
        ii. RVS verifies that the ACD does not contain a CABLE_CAPABILITIES TLV (A.3#2)
    j. CVT verifies that the certifications claimed in the Security Description TLV match what is claimed in the
    k. or zero (A.1.6#2)
    l. CVT verifies that the ACD does not contain a Playpen TLV (A.1.7#1)
    m. If a Vendor Extension TLV is present, CVT verifies that the first two bytes contain a VID that matches either the VID of the UUT vendor or the VID declared in the Test Setup

PD

PD Products are tested first with chunking (i.e. ACB indicates to UUT that it does not support Unchunked PD Extended Messages). If UUT supports unchunked PD messages, UUT is also tested using unchunked PD messages (i.e. both ACB and UUT indicate support for Unchunked PD Extended Messages).

The figure below show test setups for a UUT as SOP, SOP', or SOP". Note that the test setups for SOP' and SOP" require either a PD3.0 Sink or USB Type-C-to-Type-A Adapter downstream in order to establish PD communications.

Figure 1 UUT as SOP

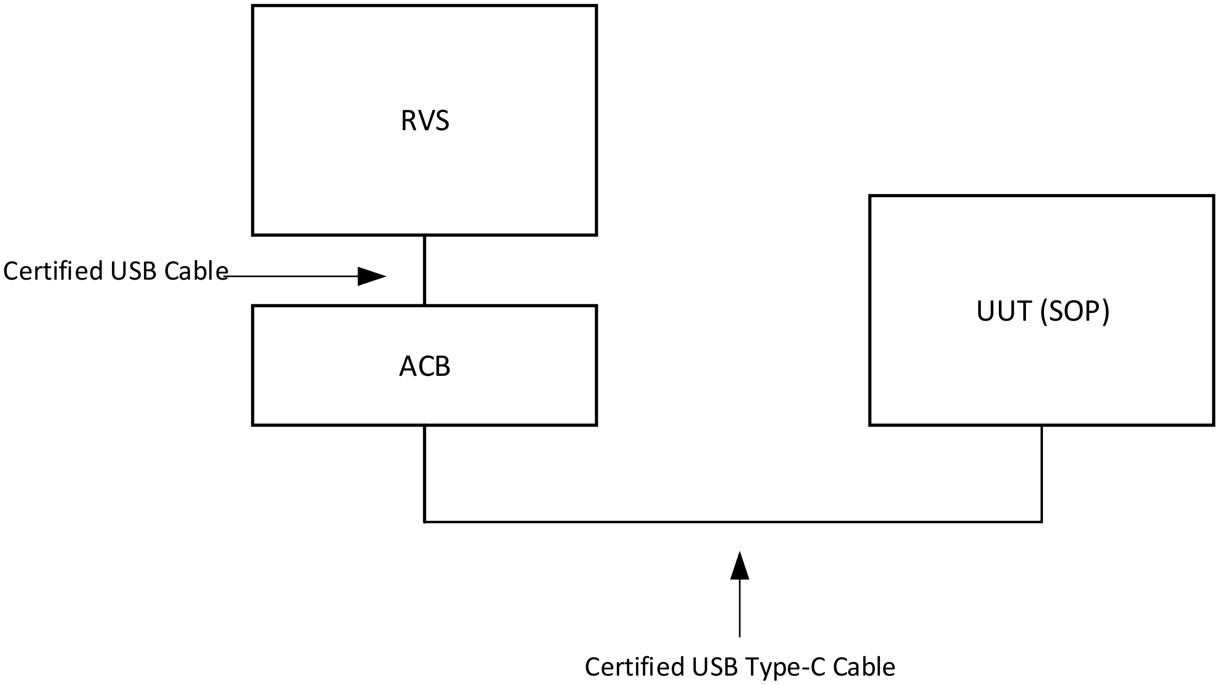


Figure 2 UUT as SOP'

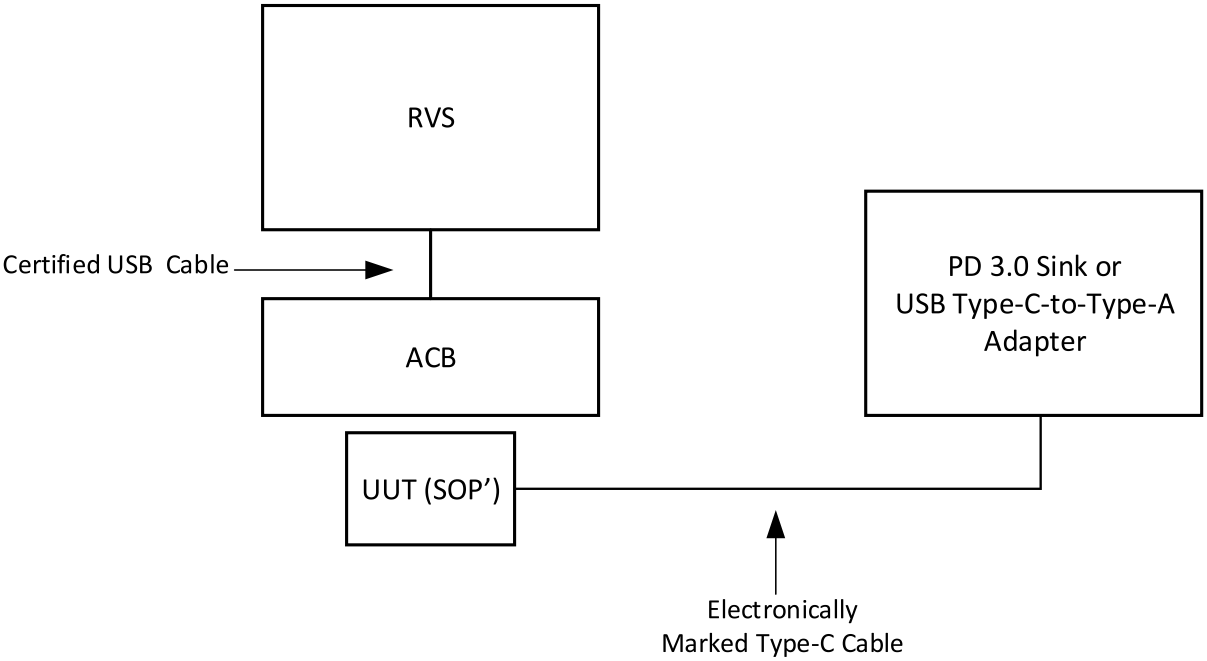
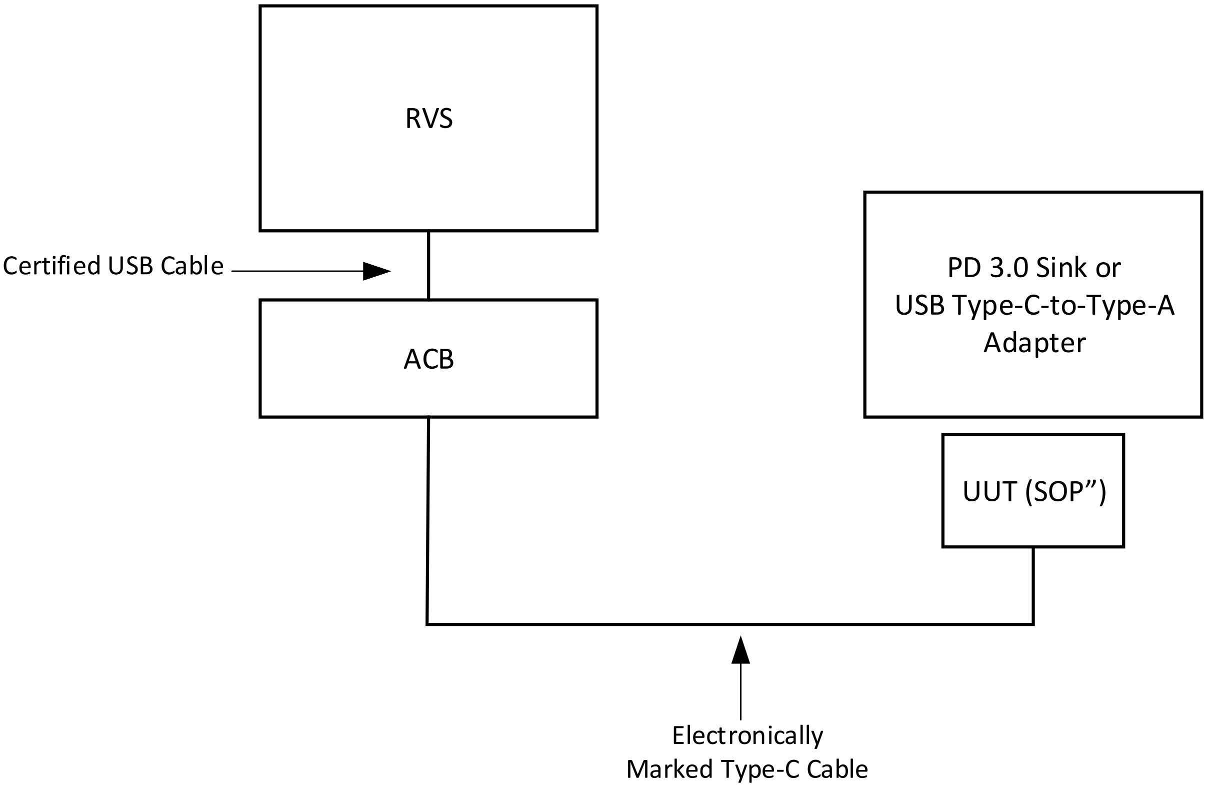


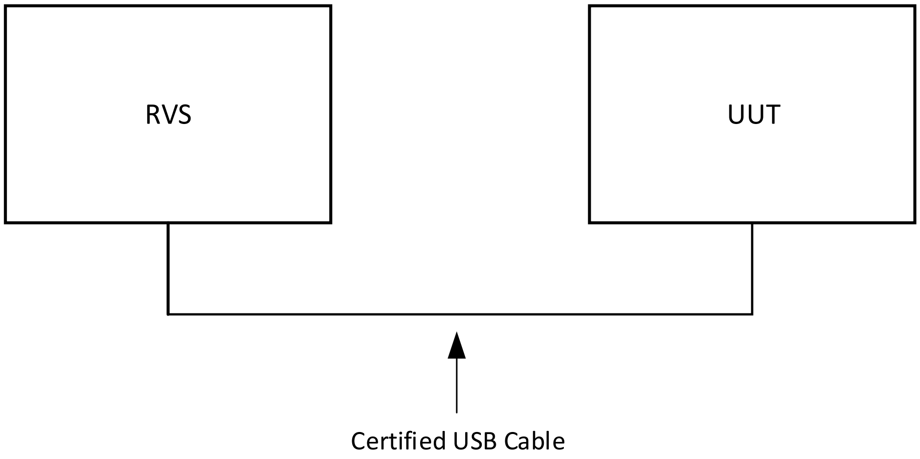
Figure 3 UUT as SOP"



USB

Unless otherwise specified, compliance tests are run with a UUT in the Addressed state. The figure below shows the test setup for a UUT that is a USB device.

Figure 4 UUT as USB Device



        n. Vendor Checklist (A.1.8#1)
        o. CVT verifies that the ACD does not contain an Extension TLV (A.1.9#1)
    11. For each non-Root Certificate in the Certificate Chain, the CVT prints the Certificate to a human-readable output and verifies:
        a. Certificate was printed without any printer errors. (3.1.2#1)
        b. Organization Name matches the Vendor Name in the Product Information section (3.1.3.1.2#2)

Note: the UUT fails this test if the CVT is unable to verify any of the Certificate Chains in slots 0 through 7. For example, if the Certificate chain in slot 0 can be verified, but the certificate chain in slot 1 cannot be verified, the UUT fails the entire test.

TD 1.4 Certificate Chain Test

A. Purpose:
    1. Verify that Certificate Chains do not exceed the maximum length
    2. Verify that Certificate Chain slots are properly used
    3. Verify that Certificate Chain slot 0 is populated

B. Asserts:
    1. 3.2#1
    2. 3.2#2
    3. 3.2#4
    4. 4#1
    5. 4.1#2
    6. 4.2#2
    7. 4.2#3
    8. 4.3#2
    9. 5.3.1#2
    10. 5.3.3#2

C. Procedure:
    1. RVS sends a GET_DIGESTS request to the UUT
    2. RVS verifies that UUT Responds with a DIGESTS Response (4.1#2)
    3. RVS saves the Slot Mask from the UUT's response for future use
    4. RVS verifies that bit 0 in the Slot mask is set to 1 (4#1).
    5. For slots 0 through 7, RVS attempts to read a Certificate Chain from the UUT:
        a. RVS sends a series of GET_CERTIFICATE Requests that read the full length of the target Certificate Chain
    6. For slots that correspond to a bit in the Slot Mask that is set to 1b:
        a. RVS verifies that the UUT responds to each GET_CERTIFICATE Request with a CERTIFICATE Response (4.2#3) (5.3.1#2) (3.2#2)
            i. Note: If at any point in the target Certificate Chain read, the RVS receives an ERROR Response, the target Certificate Chain is considered incomplete for that slot and the test fails
        b. RVS verifies that the Certificate chain does not exceed *MaxCertChainSize* in length (3.2#1)
        c. For slots 0 through 3, RVS verifies that the *RootCert* field in the Certificate Chain read from the UUT matches a SHA256 hash of the USB-IF root certificate (3.2#4)
    7. For slots that correspond to a bit in the Slot Mask that is set to 0b:
        a. RVS verifies that the UUT responds to a GET_CERTIFICATE Request with an ERROR Response (4.2#2) (5.3.1#2) (3.2#2)
    8. For slots 0 through 7, RVS sends a CHALLENGE Request to the UUT:

        a. RVS verifies that for each slot that is populated, the UUT returns a CHALLENGE_AUTH response (4.3#2)
            i. RVS verifies that for each bit set to 1 in the slot mask of the CHALLENGE_AUTH Response, the UUT contains a valid certificate chain in that slot (5.3.3#2)
            ii. RVS verifies that for each bit set to 0 in the slot mask of the CHALLENGE_AUTH Response, the UUT does not contain a valid certificate chain in that slot (5.3.3#2)
        b. RVS verifies that for each slot that is not populated, the UUT returns an ERROR response (4.3#1)

TD 1.5 Authentication Challenge Test

A. Purpose:
    1. Verify that the UUT responds to a CHALLENGE Request

B. Asserts:
    1. 4#1
    2. 4.3#2
    3. 5.3.3#1
    4. 6.5#1
    5. 6.5#2

C. Procedure:
    1. RVS sends a GET_DIGESTS request to the UUT.
    2. RVS saves the Slot Mask from the UUT's response for future use.
    3. Repeat the following steps for all populated slots between slot 0 and slot 3:
        a. RVS sends a CHALLENGE Request to the UUT
        b. RVS verifies that the UUT responds with a CHALLENGE_AUTH Response (4.3#2)
        c. RVS verifies that:
            i. The CHALLENGE_AUTH Response has bit 0 of the slot mask set (to indicate a valid certificate chain in slot 0) (4#1)
            ii. The slot mask in the CHALLENGE_AUTH Response is identical to the slot mask in the DIGESTS Response from the UUT.
            iii. The *Param1* field in the CHALLENGE_AUTH Response is equal to the *Param1* field in the CHALLENGE Request sent by the RVS (5.3.3#1)
            iv. The *Capabilities* field is set to 01h.
            v. The *CertChainHash* field contains the hash of the certificate chain used for authentication.
            vi. If UUT is a PD Product, verify that the *ContextHash* field in the CHALLENGE_AUTH Response is set to zero. (6.5#1, 6.5#2)
            vii. If UUT is a USB Product, run TD 3.3
            viii. The *Signature* field contains a 64-byte ECDSA digital signature on the message contents, where the message contents consist of the full contents of the CHALLENGE Request followed by the contents of the CHALLENGE_AUTH Response excluding the *Signature* field.
            ix. The digital signature in the *Signature* field is generated using the key that corresponds to the leaf certificate of the UUT.

TD 1.6 Certificate Chain Read Error Handling Test

A. Purpose:
    1. Verify that Certificate Chain read errors are properly handled

B. Asserts:
    1. 4.2#1
C. Procedure:
    1. RVS sends a GET_DIGESTS request to the UUT.
    2. RVS saves the Slot Mask from the UUT's response for future use.
    3. Repeat the following steps for all slots which contain a Certificate Chain as determined by the Slot Mask:
        a. RVS sends a GET_CERTIFICATE Request to the UUT to read the first 4 bytes of the certificate chain.
            i. Save the Certificate Chain Length for later use. In the following steps, this value will be referred to as *ChainLength*.
        b. RVS sends a second GET_CERTIFICATE Request to the UUT that attempts to read 101 bytes, starting at offset *ChainLength* - 100.
        c. RVS verifies that the UUT responds with an ERROR Response that has *Param1* set to (INVALID_REQUEST) and *Param2* set to 00h. (4.2#1)
        d. RVS sends a third GET_CERTIFICATE Request to the UUT that attempts to read 100 bytes starting at offset *ChainLength* + 1.
        e. RVS verifies that the UUT responds with an ERROR Response that has *Param1* set to (INVALID_REQUEST) and *Param2* set to 00h. (4.2#1)
        f. RVS sends a Fourth GET_CERTIFICATE Request to the UUT that attempts to read 101 bytes, starting at offset *ChainLength* - 1.
        g. RVS verifies that the UUT responds with an ERROR Response that has *Param1* set to (INVALID_REQUEST) and *Param2* set to 00h. (4.2#1)
        h. RVS sends a fifth GET_CERTIFICATE Request to the UUT that attempts to read 2 bytes starting at offset *ChainLength* + 100.
        i. RVS verifies that the UUT responds with an ERROR Response that has *Param1* set to (INVALID_REQUEST) and *Param2* set to 00h. (4.2#1)
        j. RVS sends a sixth GET_CERTIFICATE Request to the UUT that attempts to read the first 257 bytes of the certificate chain.
        k. RVS verifies that the UUT responds with an ERROR Response that has *Param1* set to (INVALID_REQUEST) and *Param2* set to 00h. (4.2#1)
        l. RVS sends a GET_CERTIFICATE Request to the UUT to read the first 4 bytes of the certificate chain.
        m. RVS verifies that the UUT responds with a CERTIFICATE Response.

TD 1.7 Invalid Request Error Handling Test

A. Purpose:
    1. Verify that invalid Requests are properly handled
B. Asserts:
    1. 4.4.1#1
C. Procedure:
    1. RVS sends a GET_DIGESTS Request to the UUT.
    2. RVS records what certificate chains are populated (as indicated by the slot mask in the DIGESTS Response returned by the UUT).
    3. RVS sends an Authentication Request to the UUT with the *MessageType* field set to 84h (reserved)
    4. RVS verifies that the UUT responds with an ERROR Response that has *Param1* set to INVALID_REQUEST and *Param2* set to 00h (4.4.1#1)
    5. Repeat the following steps for all unpopulated slots:
        a. RVS sends a GET_CERTIFICATE Request to the UUT with the *Param1* field set to the first unpopulated slot number.
        b. RVS verifies that the UUT responds with an ERROR Response that has *Param1* set to INVALID_REQUEST and *Param2* set to 00h (4.4.1#1)

    c. RVS sends a CHALLENGE Request to the UUT with the *Param1* field set to the first unpopulated slot number.
    d. RVS verifies that the UUT responds with an ERROR Response that has *Param1* set to INVALID_REQUEST and *Param2* set to 00h (4.4.1#1)

TD 1.8 Unsupported Protocol Error Handling Test

A. Purpose:
  1. Verify that the UUT handles an unsupported Security Protocol Version correctly
B. Asserts:
  1. 4.4.2#1
C. Procedure:
  1. RVS sends a GET_DIGESTS Request to the UUT.
  2. RVS saves the Slot Mask from the DIGESTS response for future use.
  3. RVS sends a GET_DIGESTS Request to the UUT with the *ProtocolVersion* field set to 00h.
  4. RVS verifies that the UUT responds with an ERROR Response with the *ProtocolVersion* field set to 01h, *Param1* set to UNSUPPORTED_PROTOCOL, and *Param2* set to 01h. (4.4.2#1)
  5. RVS sends a GET_DIGESTS Request to the UUT with the *ProtocolVersion* field set to FFh.
  6. RVS verifies that the UUT responds with an ERROR Response with the *ProtocolVersion* field set to 01h, *Param1* set to UNSUPPORTED_PROTOCOL, and *Param2* set to 01h. (4.4.2#1)
  7. Repeat the following steps for each populated slot:
    a. RVS sends a GET_CERTIFICATE Request to the UUT with the *ProtocolVersion* field set to 00h.
    b. RVS verifies that the UUT responds with an ERROR Response with the *ProtocolVersion* field set to 01h, *Param1* set to UNSUPPORTED_PROTOCOL, and *Param2* set to 01h. (4.4.2#1)
    c. RVS sends a GET_CERTIFICATE Request to the UUT with the *ProtocolVersion* field set to FFh.
    d. RVS verifies that the UUT responds with an ERROR Response with the *ProtocolVersion* field set to 01h, *Param1* set to UNSUPPORTED_PROTOCOL, and *Param2* set to 01h. (4.4.2#1)
  8. Repeat the following steps for each populated slot:
    a. RVS sends a CHALLENGE Request to the UUT with the *ProtocolVersion* field set to 00h.
    b. RVS verifies that the UUT responds with an ERROR Response with the *ProtocolVersion* field set to 01h, *Param1* set to UNSUPPORTED_PROTOCOL, and *Param2* set to 01h. (4.4.2#1)
    c. RVS sends a CHALLENGE Request to the UUT with the *ProtocolVersion* field set to FFh.
    d. RVS verifies that the UUT responds with an ERROR Response with the *ProtocolVersion* field set to 01h, *Param1* set to UNSUPPORTED_PROTOCOL, and *Param2* set to 01h. (4.4.2#1)

TD 1.9 Stateless Test

A. Purpose:
  1. Verify that the UUT can respond to Authentication Requests in any order
B. Asserts:
  1. 4.1#2
  2. 4.2#3
  3. 4.3#2

C. Procedure:
  1. RVS sends a GET_DIGESTS Request to the UUT.
     a. RVS saves the Slot Mask for future use.
  2. Repeat the following iterations for all populated slots:
  3. Iteration 1:
     a. RVS sends a GET_DIGESTS Request to the UUT.
     b. RVS verifies that the UUT responds with a DIGESTS response (4.1#2)
     c. RVS sends a GET_CERTIFICATE Request to the first populated slot of the UUT.
     d. RVS verifies that the UUT responds with a CERTIFICATE Response (4.2#3)
     e. RVS sends a CHALLENGE Request to the first populated slot of the UUT.
     f. RVS verified that UUT responds with a CHALLENGE_AUTH Response (4.3#2)
  4. Iteration 2:
     a. RVS sends a GET_DIGESTS Request to the UUT.
     b. RVS verifies that the UUT responds with a DIGESTS response (4.1#2)
     c. RVS sends a CHALLENGE Request to the first populated slot of the UUT.
     d. RVS verifies that the UUT responds with a CHALLENGE_AUTH Response (4.3#2)
     e. RVS sends a GET_CERTIFICATE Request to the first populated slot of the UUT.
     f. RVS verified that UUT responds with a CERTIFICATE Response (4.2#3)
  5. Iteration 3
     a. RVS sends a GET_CERTIFICATE Request to the first populated slot of the UUT.
     b. RVS verifies that the UUT responds with a CERTIFICATE response (4.2#3)
     c. RVS sends a GET_DIGEST Request to the UUT.
     d. RVS verifies that the UUT responds with a DIGESTS Response (4.1#2)
     e. RVS sends a CHALLENGE Request to the first populated slot of the UUT.
     f. RVS verified that UUT responds with a CHALLENGE_AUTH Response (4.3#2)
  6. Iteration 4
     a. RVS sends a GET_CERTIFICATE Request first populated slot of the UUT
     b. RVS verifies that the UUT responds with a CERTIFICATE response (4.2#3)
     c. RVS sends a CHALLENGE Request to the first populated slot of the UUT.
     d. RVS verifies that the UUT responds with a CHALLENGE_AUTH Response (4.3#2)
     e. RVS sends a DIGESTS Request to the UUT.
     f. RVS verified that UUT responds with a GET_DIGESTS Response (4.1#2)
  7. Iteration 5
     a. RVS sends a CHALLENGE Request to the first populated slot of the UUT
     b. RVS verifies that the UUT responds with a CHALENGE_AUTH response (4.3#2)
     c. RVS sends a GET_DIGEST Request to the UUT.
     d. RVS verifies that the UUT responds with a DIGESTS Response (4.1#2)
     e. RVS sends a GET_CERTIFICATE Request to the first populated slot of the UUT.
     f. RVS verified that UUT responds with a CERTIFICATE Response (4.2#3)
  8. Iteration 6
     a. RVS sends a CHALLENGE Request to the first populated slot of the UUT.
     b. RVS verifies that the UUT responds with a CHALLENGE_AUTH response (4.3#2)
     c. RVS sends a GET_CERTIFICATE Request to the first populated slot of the UUT.
     d. RVS verifies that the UUT responds with a CERTIFICATE Response (4.2#3)
     e. RVS sends a GET_DIGESTS Request to the UUT.

        f. RVS verified that UUT responds with a DIGESTS Response (4.1#2)

PD Product Authentication Responder Tests

The following test steps are run in conjunction with the tests defined in Chapter 1 when the UUT is PD Responder.

PD Timing Test Procedure

1. RVS verifies that the UUT responds to a GET_DIGEST Request within *tDigestSent* (unchunked = 30ms, chunked = 135ms) of receiving the GET_DIGESTS Request (6.4.2#1)
2. RVS verifies that the UUT responds to a GET_CERTIFICATE Request within *tCertSent* (unchunked = 30ms, chunked = 135ms) of receiving the GET_CERTIFICATE Request (6.4.2#2)
3. RVS verifies that the UUT responds to a CHALLENGE Request within *tChallengeAuthSent* (unchunked = 230ms, chunked = 635ms) of receiving the CHALLENGE Request (6.4.2#3)

USB Product Tests

The following test steps are run in conjunction with the tests defined in Chapter 1 when the UUT is USB device.

USB Timing Test Procedure

1. RVS verifies that the UUT responds to a GET_DIGESTS Request within *tDigestSent* (95ms) of receiving the AUTH_IN control transfer carrying the GET_DIGESTS Request (7.4.2#1)
2. RVS verifies that the UUT sends an ACK for an AUTH_OUT control transfer carrying a GET_CERTIFICATE Request within *tCertACK* (95 ms) of receiving it (7.4.2#2)
3. RVS verifies that the UUT responds to a GET_CERTIFICATE Request within *tCertSent* (495ms) of receiving the AUTH_IN control transfer carrying the GET_CERTIFICATE Request (7.4.2#3)
4. RVS verifies that the UUT sends an ACK for an AUTH_OUT control transfer carrying a CHALLENGE_AUTH Request within *tCertACK* (95 ms) of receiving it (7.4.2#4)
5. RVS verifies that the UUT responds to a CHALLENGE_AUTH Request within *tCertSent* (495ms) of receiving the AUTH_IN control transfer carrying the CHALLENGE_AUTH Request (7.4.2#5)

TD 3.1 Authentication Capability Descriptor Test

A. Purpose:
    1. Verify that UUT contains Authentication Capability Descriptor with proper values

B. Asserts:
    1. 7.1.1#1
    2. 7.1.1#2
    3. 7.1.1#3
    4. 7.1.1#4
    5. 7.1.1#5

C. Procedure:
    1. Connect UUT to RVS.
    2. RVS verifies that the UUT returns an Authentication Capability Descriptor as part of its BOS Descriptor set (7.1.1#1)
    3. RVS verifies that the Authentication Capability Descriptor from the UUT has:
        a. Bit 0 in the *bmAttributes* field set to 1 if firmware can be updated (insert reference to checklist). Otherwise, set to zero. (7.1.1#2)
        b. Bit 1 in the *bmAttributes* field set to 1 to indicate that Device changes interfaces when updated (insert reference to checklist). Otherwise, set to zero. (7.1.1#3)

        c. The *bcdProtocolVersion* set to USB 01h (Type-C™ Authentication Protocol Version) (7.1.1#4)
        d. The *bcdCapability* field set to 01h (7.1.1#5)

D. Device States for Test
    1. This test is run with the UUT in Default, Addressed and Configured state.

TD 3.2 USB Device State Error Test

A. Purpose:
    1. Verify that UUT responds to Authentication Requests with a Request Error when in the Default and Configured states

B. Asserts:
    1. 7.2.1#2, 7.2.1#3
    2. 7.2.2#2, 7.2.2#3

C. Procedure:
    1. Connect UUT to RVS.
    2. While UUT is in the Default state:
        a. RVS sends UUT a GET_DIGESTS Request
        b. RVS verifies that UUT returns a STALL (7.2.1#3)
        c. RVS clears STALL
        d. Repeat the following steps for slots 0 through 7:
        e. RVS sends UUT a GET_CERTIFICATE Request
        f. RVS verifies that UUT returns a STALL (7.2.2#3)
        g. RVS clears STALL
        h. RVS sends UUT a CHALLENGE Request
        i. RVS verifies that UUT returns a STALL (7.2.2#3)
    3. RVS enumerates and configures the UUT.
    4. While UUT is in the Configured state:
        a. RVS sends UUT a GET_DIGESTS Request
        b. RVS verifies that UUT returns a STALL (7.2.1#2)
        c. RVS clears STALL
        d. Repeat the following steps for slots 0 through 7:
        e. RVS sends UUT a GET_CERTIFICATE Request
        f. RVS verifies that UUT returns a STALL (7.2.2#2)
        g. RVS clears STALL
        h. RVS sends UUT a CHALLENGE Request
        i. RVS verifies that UUT returns a STALL (7.2.2#2)

TD 3.3 USB Context Hash Test

A. Purpose:
    1. Verify that Context Hash has the required contents

B. Asserts:
    1. 7.5#1
    2. 7.5#2

C. Procedure:
    1. Connect UUT to RVS.
    2. During UUT enumeration, RVS records all USB Descriptors required for following step.
    3. RVS calculates a 32-byte SHA256 hash of the following descriptors in the following order:
        a. Device Descriptor
        b. Complete BOS Descriptor

      c. all full Configuration Descriptors in order 1 through n (where n is the last Configuration Descriptor)
4. RVS sends a CHALLENGE Request to the UUT followed by a CHALLENGE_AUTH Request.
5. RVS verifies that the *ContextHash* field in the CHALLENGE_AUTH Response from the UUT matches the context hash calculated in step 3 (7.5#1, 7.5#2)
6. Repeat for all populated slots 0-3

TD 3.4 USB *wLength* Error Test

A. Purpose:
   1. Verify that a USB Request with invalid *wLength* field returns an error.

B. Asserts:
   1. 7.2.1#4, 7.2.2#4

C. Procedure:
   1. Connect UUT to RVS.
   2. RVS sends a GET_DIGESTS request to the UUT.
   3. RVS sends the UUT a GET_DIGESTS Request with *wLength* in the AUTH_IN Request set to zero.
   4. RVS verifies that the UUT returns a STALL (7.2.1#4)
   5. RVS clears STALL
   6. RVS sends the UUT a GET_DIGESTS Request with *wLength* in the AUTH_IN Request set to 261 (max DIGESTS Response length + 1). (7.2.1#4)
   7. RVS verifies that the UUT returns a STALL
   8. RVS clears STALL
   9. Repeat the following steps for slots 0 through 7:
   10. RVS sends the UUT a GET_CERTIFICATE Request
   11. RVS sends the UUT an AUTH_OUT Request with *wLength* set to a value that is less than the requested certificate chain read.
   12. RVS verifies that the UUT returns a STALL (7.2.2#4)
   13. RVS clears STALL
   14. RVS sends the UUT a GET_CERTIFICATE Request
   15. RVS sends the UUT an AUTH_OUT Request with *wLength* set to a value that is more than the requested certificate chain read
   16. RVS verifies that the UUT returns a STALL (7.2.2#4)
   17. RVS clears STALL
   18. RVS sends the UUT a CHALLENGE Request
   19. RVS sends the UUT an AUTH_OUT Request with *wLength* set to a value that is less than the length of a CHALLENGE_AUTH Response (168B)
   20. RVS verifies that the UUT returns a STALL (7.2.2#4)
   21. RVS clears STALL
   22. RVS sends the UUT a CHALLENGE Request
   23. RVS sends the UUT an AUTH_OUT Request with *wLength* set to a value that is greater than the length of a CHALLENGE_AUTH Response (168B).
   24. RVS verifies that the UUT returns a STALL (7.2.2#4)
   25. RVS clears STALL

Product Key Tests

TD 4.1 Private Key Test

A. Purpose:
   1. Verify that private keys are unique

B. Asserts:
   1. 3.3#1

C. Procedure:
   1. Vendor provides 3 different products to test (UUT#1, UUT#2, and UUT#3) where UUT#1 is the UUT that was used for compliance testing and UUT#2 and UUT#3 are different instances of the same end product as UUT#1.
   2. RVS reads the Leaf Certificate from each slot of each UUT
   3. RVS passes Leaf certificate to the CVT
   4. For all Leaf certificates:
      a. CVT verifies that the public key in the Leaf Certificate is different from all other public keys in the UUT from which it was read (3.3#1)
      b. CVT verifies that the public key in the Leaf certificate is different from the public keys in all other UUT used in this test (i.e. the UUT that do not contain the Leaf certificate with the public key that is being verified) (3.3#1)