Platform Security: Infrastructure Protection with DMTF's Security Protocol & Data Model (SPDM)

Viswanath Ponnuru
Dell Technologies

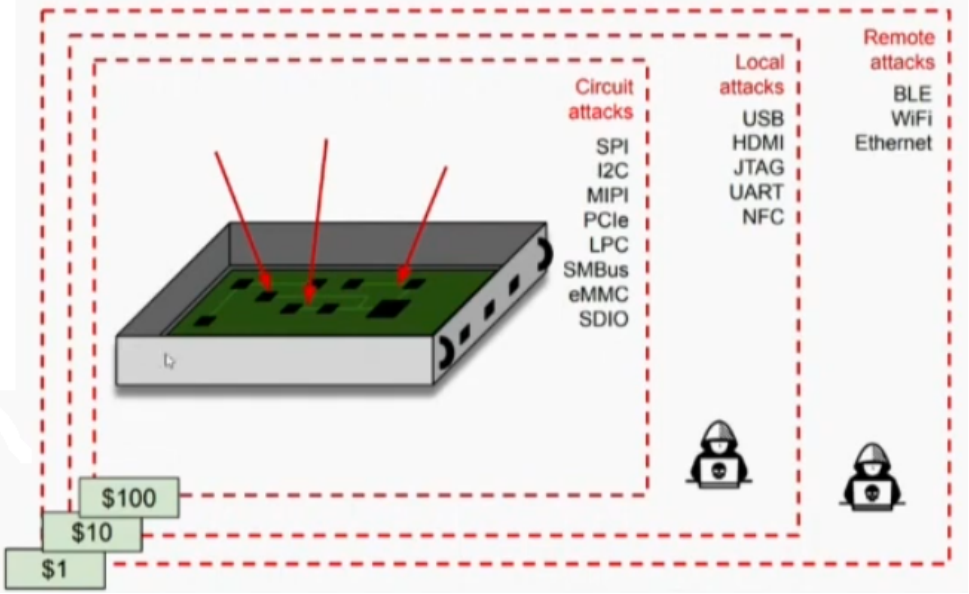
Agenda

- Platform Security – Attack Surface
- Platform Component Security Concerns
- Security Protocol Data Model (SPDM) Overview
- SPDM Authentication
- SPDM MCTP Binding
- Alliance Partners
- Additional Information

Platform Security – Attack Surface

Datacenter Service Provider Platform Security Concerns

- Detection of vulnerable hardware Components is not easy.
- Attackers are reportedly exploiting an unpatched vulnerability to take control of the platform device.
- Attackers abuse platform interface protocol analyzers to steal unencrypted information, spy on the network traffic and gather information to leverage in future attacks against the network.(I2C, SPI,..)



Ref: https://www.platformsecuritysummit.com/2019/speaker/wood/

Supply Chain Security

- Malicious code injection in the firmware
- Integrity of the firmware

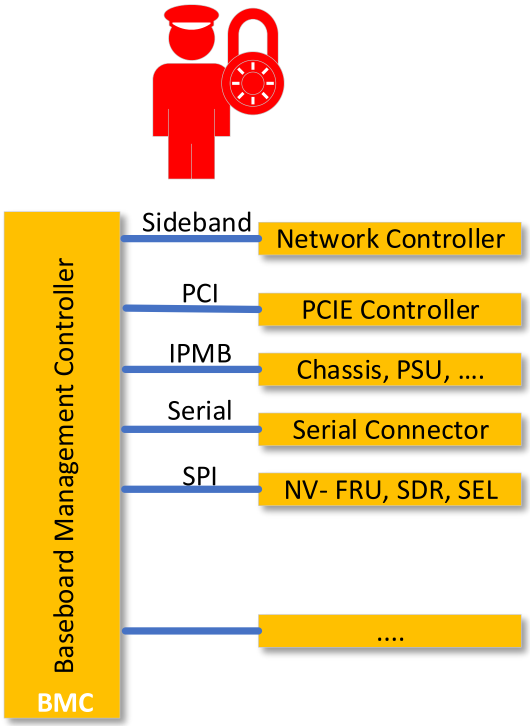
How to prevent and protect from platform component sensitive data disclosure?

Platform Component Security Concerns

These platform circuit attacks, are preying on data transfers that are unencrypted and vulnerable to eavesdropping, stealing, tampering and manipulations between the components of a platform subsystem.

Some of the security risks are:

- Sensitive (device credentials) information leakage
- Hostile component insertion, Compromised firmware(s) & Supply Chain issues
- Un-trusted device(s) snooping via probes.

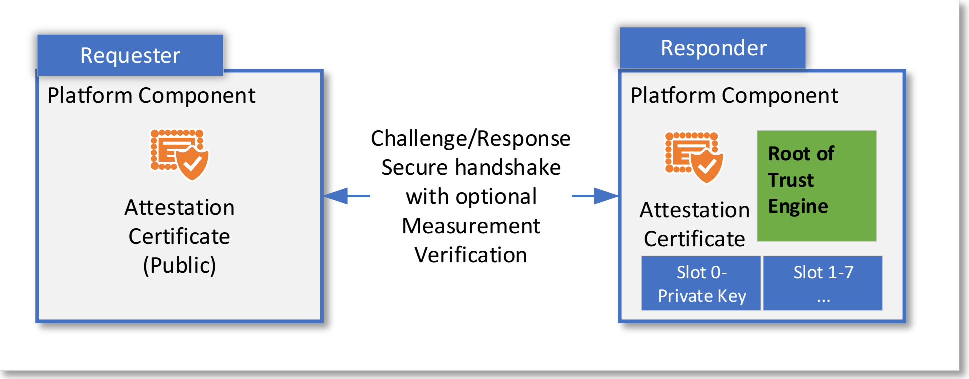


Security Protocol Data Model (SPDM) Overview

The primary goal of the Security Protocol Data Model specification is to cryptographically verify the identity and firmware integrity of each platform component is shown in diagram. And enable payload encryption and integrity protected management plane (MCTP) and other alliance partner interfaces.

Benefits:

- Certificate based authentication provides Platform Component Identity Assurance
- Facilitate privacy and data security communications over the platform interfaces.
- Root of Trust Measurement for firmware integrity checks.
- Leveraging the industry proven standards approach such TLS, USB Authentication, etc.

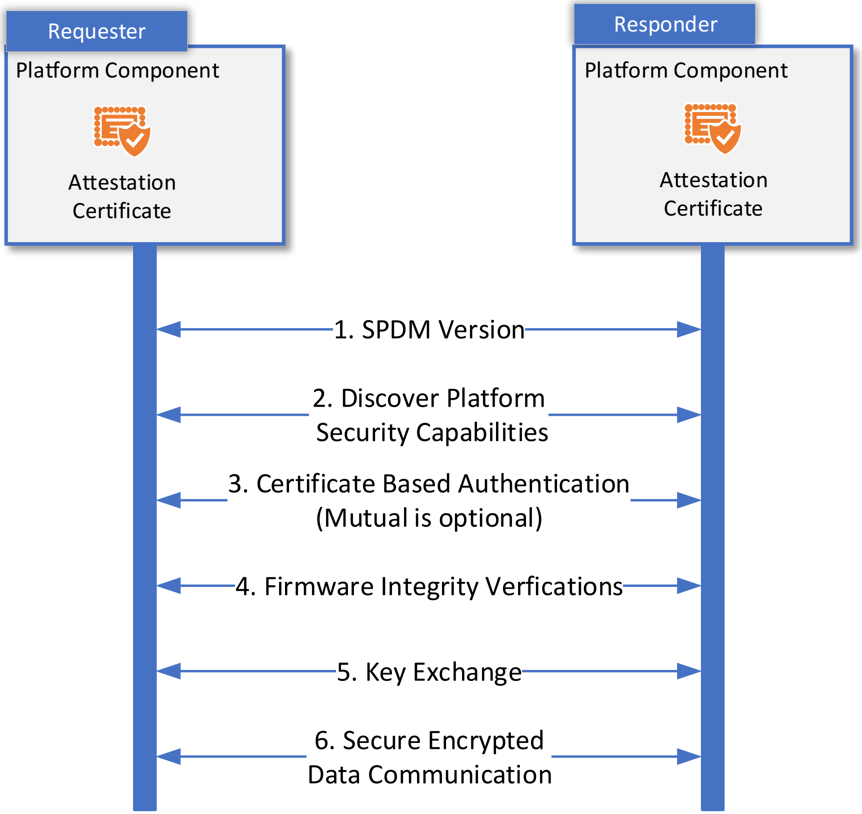


SPDM Authentication

The SPDM defines sets of messages that are exchanged between platform components for establishing the encrypted communications.

This process helps:

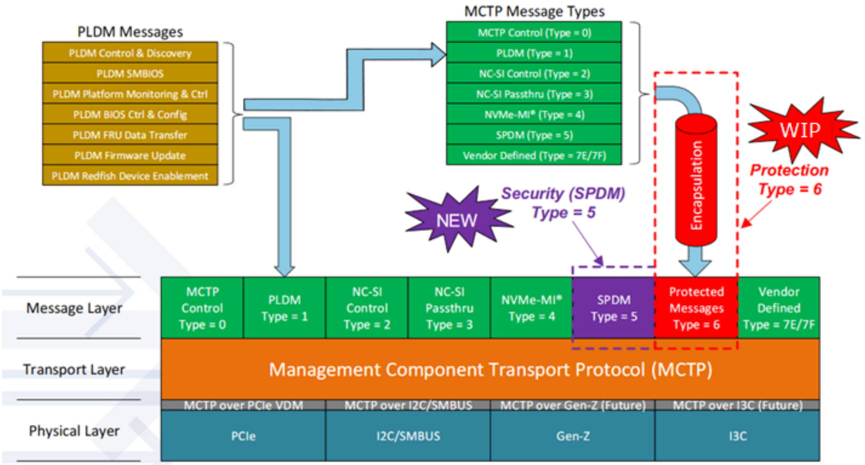
- Achieving both confidentiality, authenticity by verify each other identity
- Negotiate cipher suites and crypto algorithms required to establish a secure connection
- Determines what version of SPDM version will be used in the session
- Request/Response is bidirectional, any component can request for authentication.


Requester
Platform Component
Attestation Certificate
Responder
Platform Component
Attestation Certificate
1. SPDM Version
2. Discover Platform Security Capabilities
3. Certificate Based Authentication (Mutual is optional)
4. Firmware Integrity Verfications
5. Key Exchange
6. Secure Encrypted Data Communication

SPDM over MCTP Binding

- SPDM over MCTP binding defines the format of SPDM messages transported over MCTP

- MCTP Message Types for SPDM is shown in the figure and details:
  - Type 5: Device security capability discovery, initial handshake and session key exchange
  - Type 6: Encrypting the payload once type 5 is established between Requester and Responder



PLDM Messages

PLDM Control & Discovery
PLDM SMBIOS
PLDM Platform Monitoring & Ctrl
PLDM BIOS Ctrl & Config
PLDM FRU Data Transfer
PLDM Firmware Update
PLDM Redfish Device Enablement

MCTP Message Types

MCTP Control (Type = 0)
PLDM (Type = 1)
NC-SI Control (Type = 2)
NC-SI Passthru (Type = 3)
NVMe-MI® (Type = 4)
SPDM (Type = 5)
Vendor Defined (Type = 7E/7F)

Encapsulation

WIP

Protection Type = 6

NEW

Security (SPDM) Type = 5

Message Layer: MCTP Control Type = 0 | PLDM Type = 1 | NC-SI Control Type = 2 | NC-SI Passthru Type = 3 | NVMe-MI® Type = 4 | SPDM Type = 5 | Protected Messages Type = 6 | Vendor Defined Type = 7E/7F

Transport Layer: Management Component Transport Protocol (MCTP)

MCTP over PCIe VDM | MCTP over I2C/SMBUS | MCTP over Gen-Z (Future) | MCTP over I3C (Future)

Physical Layer: PCIe | I2C/SMBUS | Gen-Z | I3C

Alliance Partners

The SPDM message exchanges are defined in generic fashion that allows the messages to be communicated across different physical mediums over different transport protocols. For the complete list of DMTF alliance partners are available in the location.

OPEN
Compute Project®
PCI
SIG
nvm
EXPRESS®
CXL
Compute
Express
Link™

Some of the SPDM message exchange capabilities are based on security model that the USB Authentication Specification Rev 1.0 with ECN and Errata through January 7, 2019.

Additional Information

DMTF SPDM
Version 0.9 - https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_0.9.0a.pdf
Version 1.0 - https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.0.0.pdf
Version 1.1 - https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.1.0.pdf

SPDM over MCTP Binding
Version 1.0 - https://www.dmtf.org/sites/default/files/standards/documents/DSP0275_1.0.0.pdf

Security Protocol and Data Model (SPDM) Architecture

Version 1.0.0 Release

PMCI Security Task Force

Last Updated: 12/11/2019

Disclaimer

- The information in this presentation represents a snapshot of work in progress within the DMTF.
- This information is subject to change without notice. The standard specifications remain the normative reference for all information.
- For additional information, see the DMTF website.

- This information is a summary of the information that will appear in the specifications. See the specifications for further details.

Security Protocol and Data Model 1.0

- How?
  - Two Major Features
    - Authentication
    - Attestation (authenticated measurements)
  - Capable of being referenced by other standards.
    - DMTF is initially mapping to MCTP.
    - Alliance Partners are considering mapping SPDM to their standards.

HDBaseT™

PCI SIG®

mipi® alliance

GEN Z™

OPEN Compute Project®

SPDM 1.0 – Authentication

- Allows a platform to verify the identity of the attached component.
- Redfish
  - Identity is also exposed in Redfish.
- Enables a platform to determine what to do if the identity of a component did not verify correctly.

- Cryptography
  - Leverage X.509v3 certificates

SPDM 1.0 – Attestation

- Allows a platform to verify the state of the component.
- Multiple measurements allow platforms to verify various configurations of the component.

- Measurements:
  - Hashes and raw bit streams of various configurations of a component

- Examples of Measurement Coverage (Implementation Choices):
  - Immutable Code
  - Mutable Code
  - Boot Stages
  - Configuration Data
  - State Variables

Background and Use Cases

- Security Requirements for PMCI Standards and Protocol, September 2018 (https://www.dmtf.org/sites/default/files/PMCI_Security-Release_1.0.pdf)
- SPDM 1.0 – Keynote, July 2019 (https://www.dmtf.org/sites/default/files/SPDM_1.0_Keynote_APTS.pdf)
- PCIe® Component Authentication (https://pcisig.com/pcie%C2%AE-component-authentication)

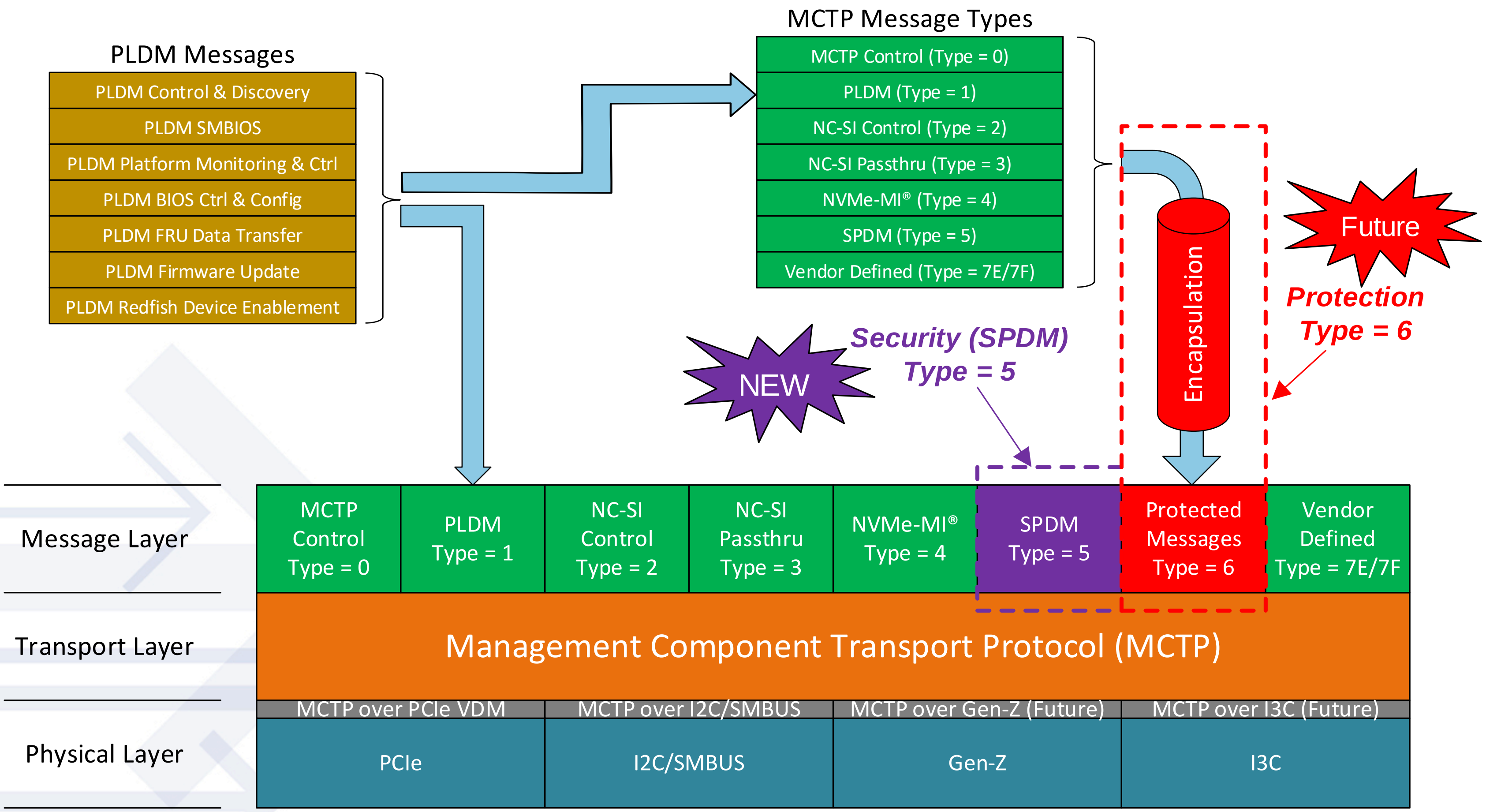
Guiding Principles

- Use MCTP message type 5 for all authentication commands including the future ones used for setting up secure sessions
- Use MCTP message type 6 for secured transport of encapsulated MCTP messages as appropriate (Future Version)
- Derived from USB Authentication –
  - Some of the content is derived from USB Authentication Specification Rev 1.0 with ECN and Errata through January 7, 2019
  - https://www.usb.org/sites/default/files/USB%20Authentication%20Specification%20Rev%201.0%20with%20ECN%20and%20Errata%20through%20January%207%2C%202019.zip
- Fields are defined to be little endian unless otherwise noted

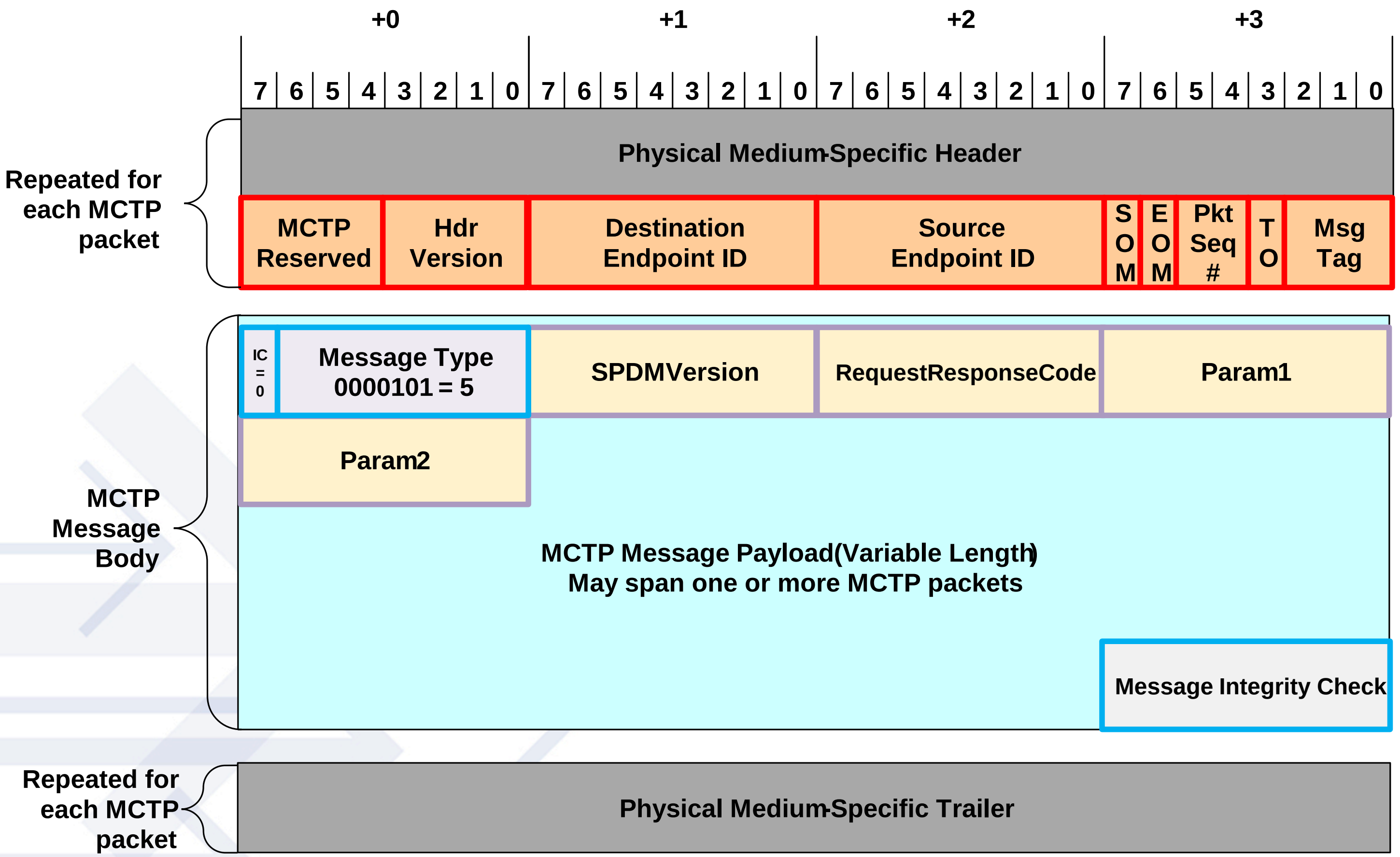
Specifications

- DSP0274
  - Security Protocol and Data Model (SPDM) Specification
    - This specification contains message exchange, sequence diagrams, message formats, and other relevant semantics for authentication, firmware measurement, and certificate management
- DSP0275
  - SPDM over MCTP Binding Specification
    - This specification contains the mapping of SPDM to MCTP message type 5

PMCI MCTP Security Proposal – Diagram View



MCTP Message Type 5 (Security Commands) Format



+0 +1 +2 +3

7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0

Repeated for each MCTP packet

Physical Medium-Specific Header

MCTP Reserved | Hdr Version | Destination Endpoint ID | Source Endpoint ID | SOM | EOM | Pkt Seq # | TO | Msg Tag

MCTP Message Body

IC = 0 | Message Type 0000101 = 5 | SPDMVersion | RequestResponseCode | Param1

Param2

MCTP Message Payload(Variable Length)
May span one or more MCTP packets

Message Integrity Check

Repeated for each MCTP packet

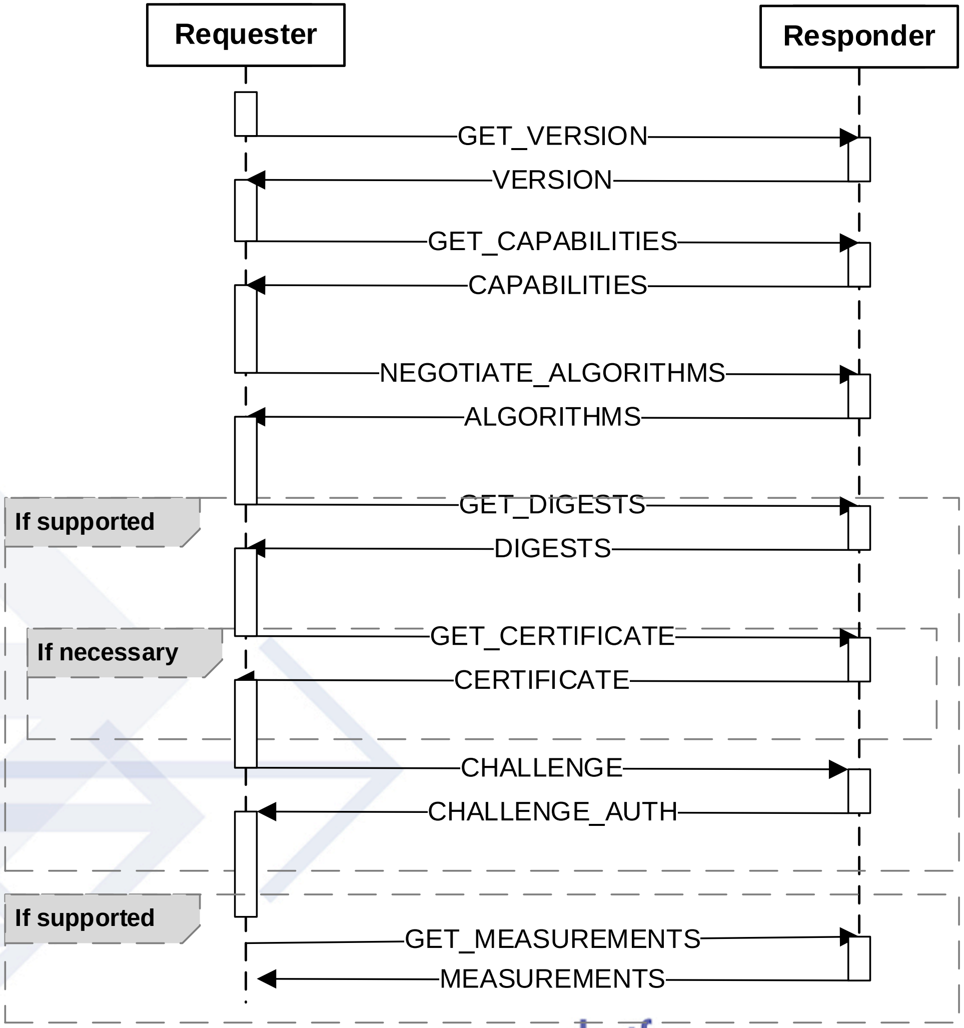
Physical Medium-Specific Trailer

SPDM Specification Details

SPDM Common Format

Offset Byte[bit]	Field Name	Size bits	Definition
0[7:4]	*SPDMMajorVersion*	4	The major version of the SPDM Specification.
0[3:0]	*SPDMMinorVersion*	4	The minor version of the SPDM Specification.
1	*RequestResponseCode*	8	Identifies type of request or type of response.
2	*Param1*	8	The first one-byte parameter. The contents of the parameter is specific to the Request Response Code.
3	*Param2*	8	The second one-byte parameter. The contents of the parameter is specific to the Request Response Code.

High-level Authentication Sequence Diagram



RequestResponseCode Field: Part 1 Requests

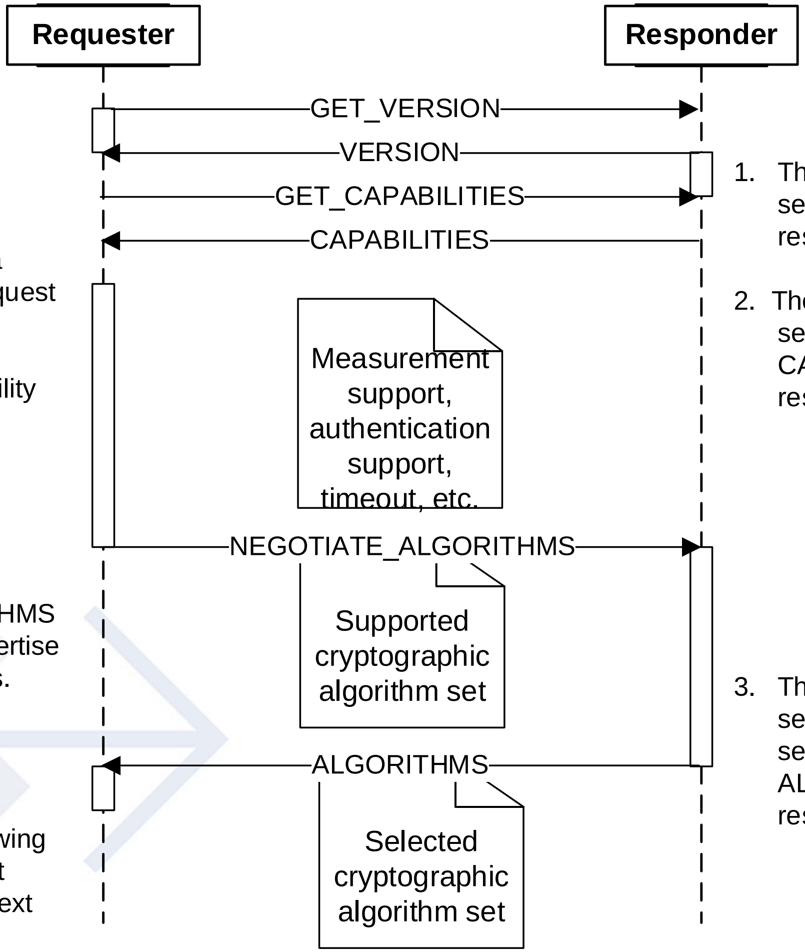
Request	Code Value	Implementation Requirement
GET_DIGESTS	0x81	Optional
GET_CERTIFICATE	0x82	Optional
CHALLENGE	0x83	Optional
GET_VERSION	0x84	Required
GET_MEASUREMENTS	0xE0	Optional
GET_CAPABILITIES	0xE1	Required
NEGOTIATE_ALGORITHMS	0xE3	Required
VENDOR_DEFINED_REQUEST	0xFE	Optional
RESPOND_IF_READY	0xFF	Required
Reserved	0x80, 0x85-0xDF, 0xE2, 0xE4-0xFD	SPDM implementations compatible with this version shall not use the reserved request codes.

RequestResponseCode Field: Part 2 Responses

Response	Value	Implementation Requirement
DIGESTS	0x01	Optional
CERTIFICATE	0x02	Optional
CHALLENGE_AUTH	0x03	Optional
VERSION	0x04	Required
MEASUREMENTS	0x60	Optional
CAPABILITIES	0x61	Required
ALGORITHMS	0x63	Required
VENDOR_DEFINED_RESPONSE	0x7E	Optional
ERROR	0x7F	See later slide
Reserved	0x00, 0x05-0x5F, 0x62, 0x64-0x7D	SPDM implementations compatible with this version shall not use the reserved response codes.

Requester/Responder State Negotiation Sequence Diagram

1. The Requester sends a GET_VERSION request message.
2. The Requester sends a GET_CAPABILITIES request message.
3. Determine device capability and feature support.
4. The Requester sends a NEGOTIATE_ALGORITHMS request message to advertise the supported algorithms.
5. The Requester uses the selected cryptographic algorithm set for all following exchanges, until the next GET_VERSION or the next reset.



Requester

Responder

GET_VERSION

VERSION

GET_CAPABILITIES

CAPABILITIES

Measurement support, authentication support, timeout, etc.

NEGOTIATE_ALGORITHMS

Supported cryptographic algorithm set

ALGORITHMS

Selected cryptographic algorithm set

1. The Responder sends a VERSION response message.
2. The Responder sends a CAPABILITIES response message.
3. The Responder selects the algorithm set and sends a ALGORITHMS response message.

GET_VERSION Request

This request message shall retrieve an endpoint's SPDM version.

Offset	Field	Size (Bytes)	Value
0	*SPDMVersion*	1	V1.0 = 10h
1	*Request/ResponseCode*	1	84h = GET_VERSION
2	*Reserved1*	1	Reserved
3	*Reserved2*	1	Reserved

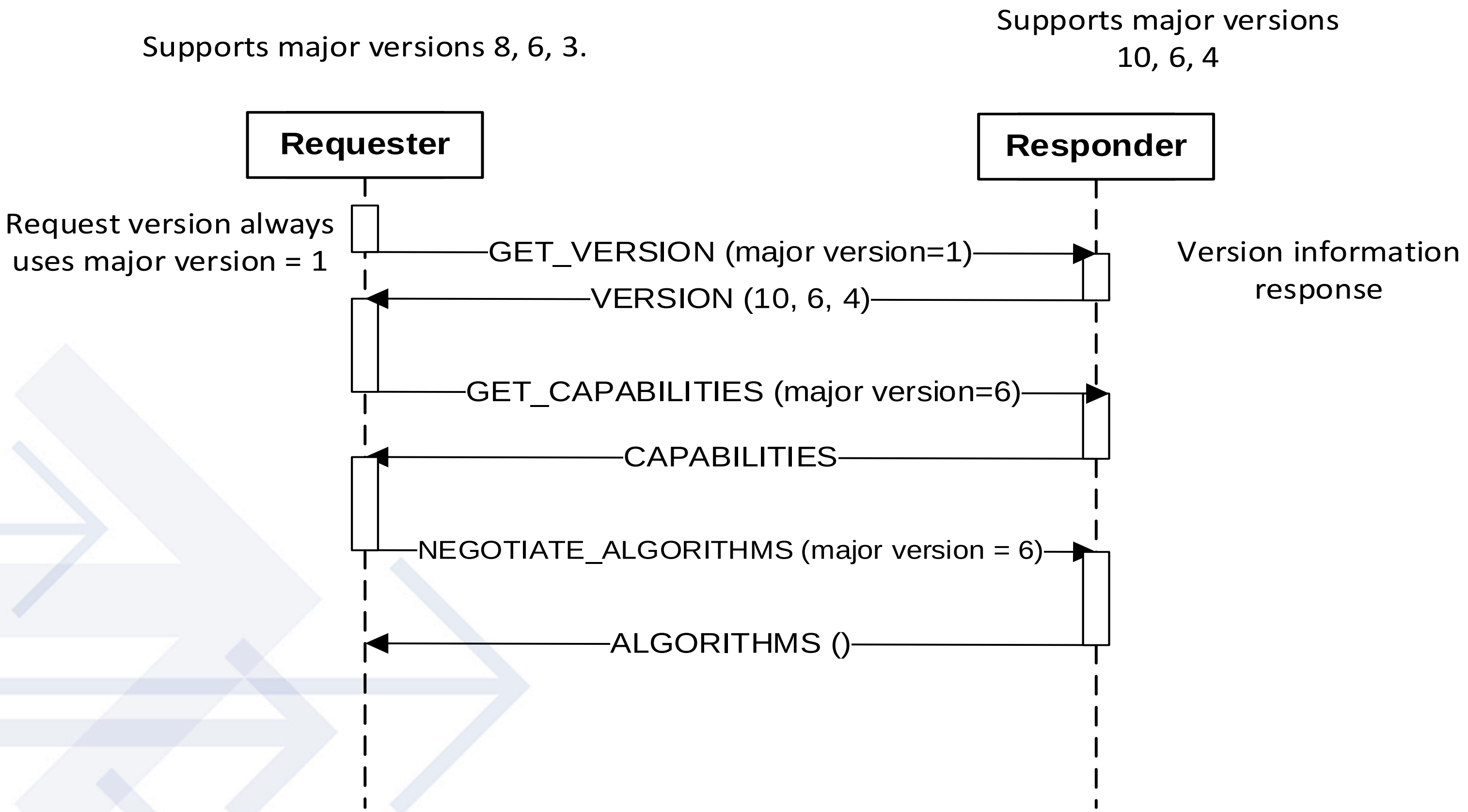
Successful VERSION Response

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x04 = VERSION
2	Param1	1	Reserved
3	Param2	1	Reserved
4	Reserved	1	Reserved
5	VersionNumberEntryCount	1	Number of version entries present in this table (=n).
6	VersionNumberEntry1:n	2 x n	16-bit version entry.

VERSION Number Entry Definition

Bit	Field	Value
[15:12]	MajorVersion	Version of the specification with changes that are incompatible with one or more functions in earlier major versions of the specification. [15:12]
[11:8]	MinorVersion	Version of the specification with changes that are compatible with functions in earlier minor versions of this major version specification. [11:8]
[7:4]	UpdateVersionNumber	Version of the specification with editorial updates but no functionality additions or changes. Informational; possible errata fixes. Ignore when checking versions for interoperability. [7:4]
[3:0]	Alpha	Pre-release work-in-progress version of the specification. Backward compatible with earlier minor versions of this major version specification. However, because the Alpha value represents an in-development version of the specification, versions that share the same major and minor version numbers but have different Alpha versions may not be fully interoperable. Released versions must have an Alpha value of zero. [3:0]

Discovering Common Major Version


Supports major versions 8, 6, 3.
Supports major versions 10, 6, 4
Requester
Responder
Request version always uses major version = 1
GET_VERSION (major version=1)
Version information response
VERSION (10, 6, 4)
GET_CAPABILITIES (major version=6)
CAPABILITIES
NEGOTIATE_ALGORITHMS (major version = 6)
ALGORITHMS ()

GET_CAPABILITIES Request

This request is used to discover endpoint protocol capabilities.

Offset	Field	Size	Value
0	*SPDMVersion*	1	V1.0 = 10h
1	*Request/Response Code*	1	E1h = GET_CAPABILITIES
2	*Reserved1*	1	Reserved
3	*Reserved2*	1	Reserved

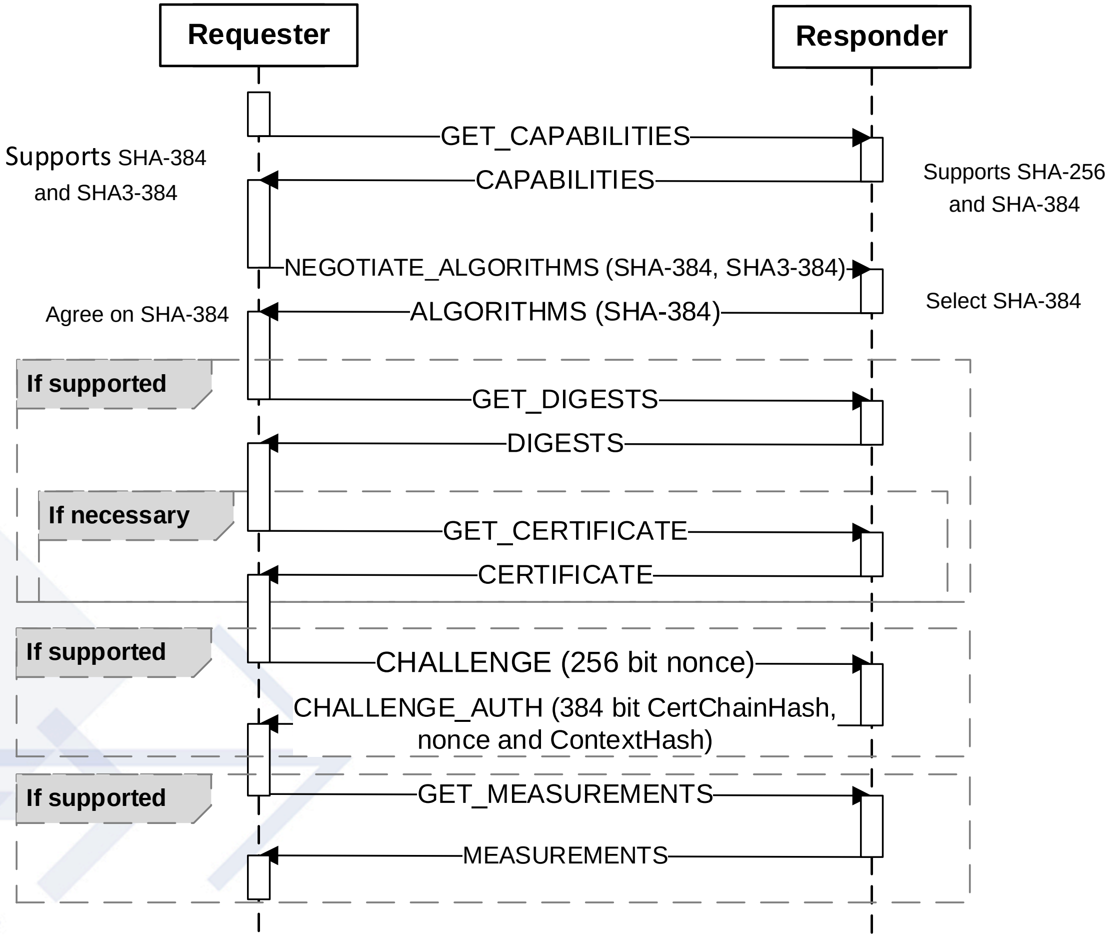
Successful CAPABILITIES

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x61 = CAPABILITIES
2	Param1	1	Reserved
3	Param2	1	Reserved
4	Reserved	1	Reserved
5	CTExponent	1	The value of this shall be the exponent of base 2. Used to calculate CT. The equation for CT shall be 2^{CT} microseconds (us). For example, if CTExponent is 10, CT is 2^{10} = 1024 us.
6	Reserved	2	Reserved
8	Flags	4	See next slide.

CAPABILITIES Flags Field Definition

Byte	Bit	Field	Value
0	0	CACHE_CAP	If set, the Responder supports the ability to cache the Negotiated State across a reset. This allows the Requester to skip reissuing the GET_VERSION , GET_CAPABILITIES and NEGOTIATE_ALGORITHMS requests after a reset. The Responder shall cache the selected cryptographic algorithms as one of the parameters of the Negotiated State. If the Requester chooses to skip issuing these requests after the reset, the Requester shall also cache the same selected cryptographic algorithms.
0	1	CERT_CAP	If set, Responder supports GET_DIGESTS and GET_CERTIFICATE messages.
0	2	CHAL_CAP	If set, Responder supports CHALLENGE request message.
0	4:3	MEAS_CAP	•The Responder's MEASUREMENT capabilities. •00b. The Responder does not support MEASUREMENTS capabilities. •01b. The Responder supports MEASUREMENTS but cannot perform signature generation. •10b. The Responder supports MEASUREMENTS and can generate signatures. •11b. Reserved
0	5	MEAS_FRESH_CAP	•0. As part of MEASUREMENTS response message, the Responder may return MEASUREMENTS that were computed during the last Responder's reset. •1. The Responder can recompute all MEASUREMENTS in a manner that is transparent to the rest of the system and shall always return fresh MEASUREMENTS as part of MEASUREMENTS response message.
0	7:6	Reserved	Reserved
1	7:0	Reserved	Reserved
2	7:0	Reserved	Reserved
3	7:0	Reserved	Reserved

Hashing Algorithm Selection Sequence Diagram



NEGOTIATE_ALGORITHMS Request Part 1

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0xE3 = NEGOTIATE_ALGORITHMS
2	Param1	1	Reserved
3	Param2	1	Reserved
4	Length	2	Length of the entire request message, in bytes. Length shall be less than 64 bytes.
6	MeasurementSpecification	1	This field is a bitmask. The values for this field shall be those defined in the MeasurementSpecification field of GET_MEASUREMENTS request message and MEASUREMENTS response message. The Requester may set more than one bit to indicate multiple measurement specification support.
7	Reserved	1	Reserved
8	*BaseAsymAlgo*	4	•Bit mask listing Requester-supported SPDM-enumerated asymmetric key signature algorithms for the purposes of signature verification. •Byte 0 Bit 0. TPM_ALG_RSASSA_2048 •Byte 0 Bit 1. TPM_ALG_RSAPSS_2048 •Byte 0 Bit 2. TPM_ALG_RSASSA_3072 •Byte 0 Bit 3. TPM_ALG_RSAPSS_3072 •Byte 0 Bit 4. TPM_ALG_ECDSA_ECC_NIST_P256 •Byte 0 Bit 5. TPM_ALG_RSASSA_4096 •Byte 0 Bit 6. TPM_ALG_RSAPSS_4096 •Byte 0 Bit 7. TPM_ALG_ECDSA_ECC_NIST_P384 •Byte 1 Bit 0. TPM_ALG_ECDSA_ECC_NIST_P521 All other values reserved.

NEGOTIATE_ALGORITHMS Request Part 2

Offset	Field	Size (bytes)	Value
12	*BaseHashAlgo*	4	•Bit mask listing Requester-supported SPDM-enumerated cryptographic hashing algorithms .Byte 0 Bit 0. TPM_ALG_SHA_256 •Byte 0 Bit 1. TPM_ALG_SHA_384 •Byte 0 Bit 2. TPM_ALG_SHA_512 •Byte 0 Bit 3. TPM_ALG_SHA3_256 •Byte 0 Bit 4. TPM_ALG_SHA3_384 •Byte 0 Bit 5. TPM_ALG_SHA3_512 All other values reserved.
16	Reserved	12	Reserved
28	*ExtAsymCount*	1	Number of Requester-supported extended asymmetric key signature algorithms (=A). A + E shall be less than or equal to 8.
29	*ExtHashCount*	1	Number of Requester-supported extended hashing algorithms (=E). A + E shall be less than or equal to 8.
30	Reserved	2	Reserved for future use
32	*ExtAsym*	4*A	List of Requester-supported extended asymmetric key signature algorithms. The format of this field is described in Extended Algorithm Field Format Table.
32+4*A	*ExtHash*	4*E	List of the extended hashing algorithms supported by Requester. The format of this field is described in Extended Algorithm Field Format Table.

Successful ALGORITHMS Part 1

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x63 = ALGORITHMS
2	Param1	1	Reserved
3	Param2	1	Reserved
4	Length	2	Length of the response message, in bytes.
6	MeasurementSpecificationSel	1	Bit mask. The Responder shall select one of the measurement specifications supported by the Requester. Thus, no more than one bit shall be set. The values in this field shall be those defined in the MeasurementSpecification field.
7	Reserved	1	Reserved
8	MeasurementHashAlgo	4	•Bit mask listing SPDM-enumerated hashing algorithm for measurements. M represents the length of the measurement hash field in measurement block structure. The Responder shall ensure the length of measurement hash field during all subsequent MEASUREMENT response messages to the Requester until the next ALGORITHMS response message is M. •Bit 0. Raw Bit Stream Only, M=0 •Bit 1. TPM_ALG_SHA_256, M=32 •Bit 2. TPM_ALG_SHA_384, M=48 •Bit 3. TPM_ALG_SHA_512, M=64 •Bit 4. TPM_ALG_SHA3_256, M=32 •Bit 5. TPM_ALG_SHA3_384, M=48 •Bit 6. TPM_ALG_SHA3_512, M=64 If the Responder supports GET_MEASUREMENTS, exactly one bit in this bit field shall be set. Otherwise, the Responder shall set this field to 0. A Responder shall only select Bit 0 if the Responder supports Raw Bit Streams as the only form of measurement; otherwise, it shall select one of the other bits.
12	BaseAsymSel	4	Bit mask listing the SPDM-enumerated asymmetric key signature algorithm selected. A Responder that returns CHAL_CAP=0 and MEAS_CAP != 2 shall set this field 0. Other Responders shall set no more than one bit.
16	BaseHashSel	4	Bit mask listing the SPDM-enumerated hashing algorithm selected. A Responder that returns CHAL_CAP=0 and MEAS_CAP != 2 shall set this field 0. Other Responders shall set no more than one bit.

The responder shall respond showing no more than one chosen algorithm per method.

Successful ALGORITHMS Part 2

Offset	Field	Size (bytes)	Value
20	Reserved	12	Reserved.
32	ExtAsymSelCount	1	The number of extended asymmetric key signature algorithms selected. Shall be either 0 or 1 (=A'). A Requester that returns CHAL_CAP=0 and MEAS_CAP != 2 shall set this field 0.
33	ExtHashSelCount	1	The number of extended hashing algorithms selected. Shall be either 0 or 1 (=E'). A Requester that returns CHAL_CAP=0 and MEAS_CAP != 2 shall set this field 0.
34	Reserved	2	Reserved
36	ExtAsymSel	4*A'	The extended asymmetric key signature algorithm selected. Responder must be able to sign a response message using this algorithm and Requester must have listed this algorithm in the request message indicating it can verify a response message using this algorithm. The Responder shall use this asymmetric signature algorithm for all subsequent applicable response messages to the Requester. The format of this field is described in Extended Algorithm Field Format Table.
36+4*A	ExtHashSel	4*E'	The extended Hashing algorithm selected. The Responder shall use this hashing algorithm during all subsequent response messages to the Requester. The Requester shall use this hashing algorithm during all subsequent applicable request messages to the Responder. The format of this field is described in Extended Algorithm Field Format Table.

The responder shall respond showing no more than one chosen algorithm per method.

Extended Algorithm Format Field Table

Offset	Field	Description
0	Registry ID	This field shall represent the registry or standards body. This field's value shall be one listed in the **ID** column of Table 29.
1	Reserved	Reserved
[2:3]	Algorithm ID	This field shall indicate the desired algorithm. The value of this field is owned by the registry or standards body.

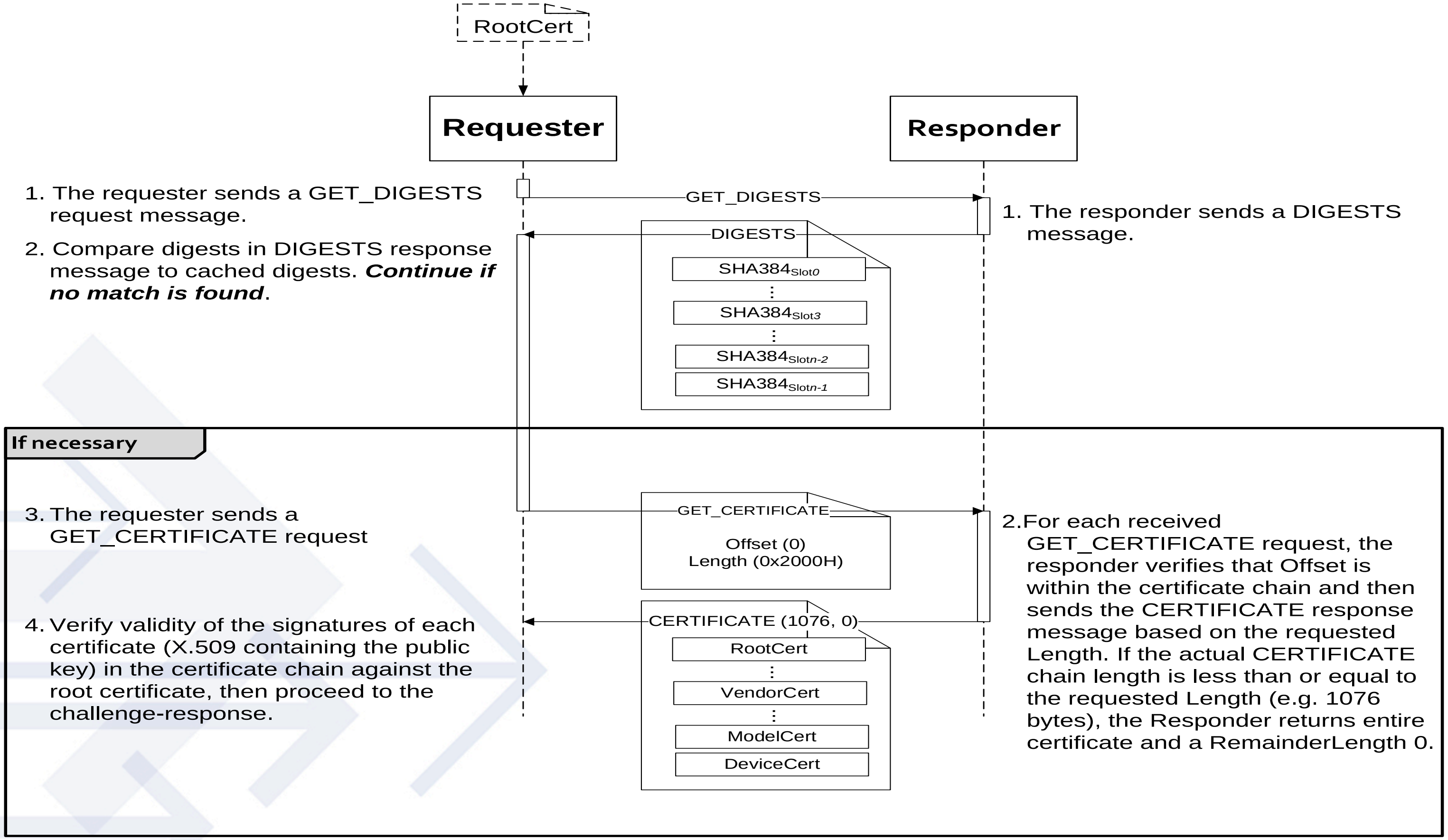
The responder shall respond showing no more than one chosen algorithm per method.

Registry or Standards Body ID

ID	Vendor ID Len (bytes)	Registry or standards body name	Description
0x0	0	DMTF	DMTF does not have a Vendor ID registry. At present, DMTF does not have any algorithms defined for use in extended algorithms fields.
0x1	2	TCG	Vendor is identified using TCG Vendor ID Registry. For extended algorithms, see TCG Algorithm Registry.
0x2	2	USB	Vendor is identified using USB's vendor ID.
0x3	2	PCI-SIG	Vendor is identified using PCI-SIG Vendor ID.
0x4	4	IANA	Vendor is identified using the Internet Assigned Numbers Authority's Private Enterprise Number (PEN).
0x5	4	HDBaseT	Vendor is identified using HDBaseT HDCD Entity.
0x6	2	MIPI	Vendor is identified using MIPI’s Manufacturer ID

The responder shall respond showing no more than one chosen algorithm per method.

GET_DIGESTS / GET_CERTIFICATE Sequence Diagram (Single Certificate Chain)



1. The requester sends a GET_DIGESTS request message.
2. Compare digests in DIGESTS response message to cached digests. ***Continue if no match is found***.

1. The responder sends a DIGESTS message.

3. The requester sends a GET_CERTIFICATE request

4. Verify validity of the signatures of each certificate (X.509 containing the public key) in the certificate chain against the root certificate, then proceed to the challenge-response.

2. For each received GET_CERTIFICATE request, the responder verifies that Offset is within the certificate chain and then sends the CERTIFICATE response message based on the requested Length. If the actual CERTIFICATE chain length is less than or equal to the requested Length (e.g. 1076 bytes), the Responder returns entire certificate and a RemainderLength 0.

GET_DIGESTS Request

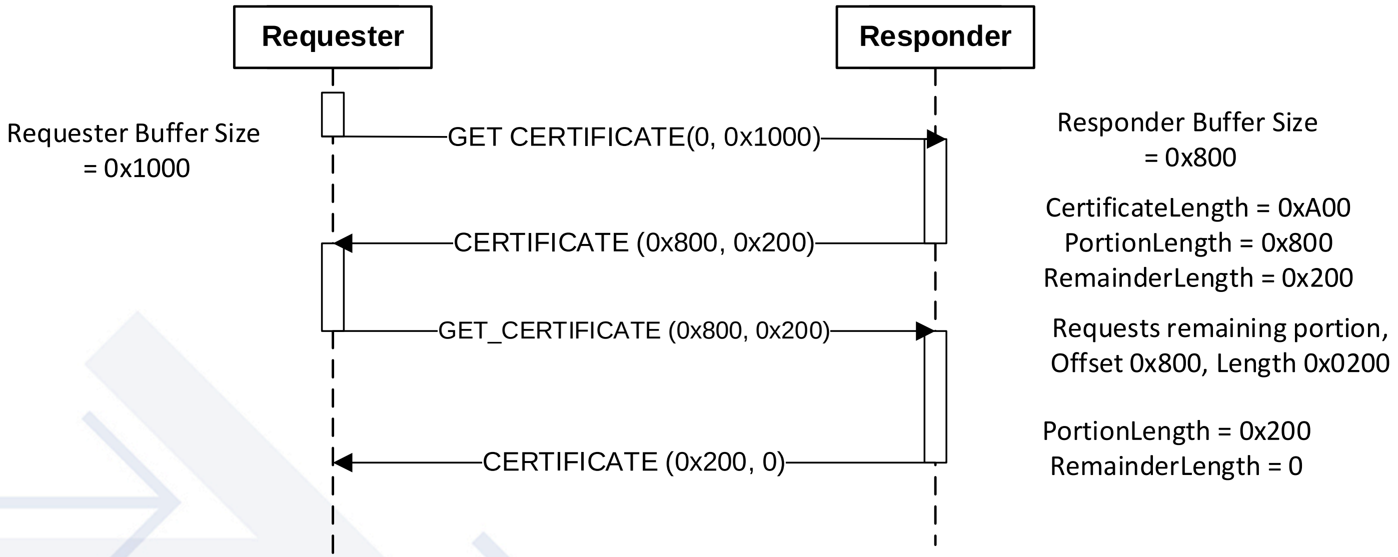
This Request is used to retrieve Certificate Chain digests.

Offset	Field	Size	Value
0	*SPDMVersion*	1	V1.0 = 10h
1	*Request/Response Code*	1	81h = GET_DIGESTS
2	*Reserved1*	1	Reserved
3	*Reserved2*	1	Reserved

Successful DIGESTS

ffset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x01 = DIGESTS
2	Param1	1	Reserved
3	Param2	1	Slot mask. The bit in position K of this byte shall be set to 1b if and only if slot number K contains a certificate chain for the protocol version in the SPDMVersion field. (Bit 0 is the least significant bit of the byte.) The number of digests returned shall be equal to the number of bits set in this byte. The digests shall be returned in order of increasing slot number.
4	Digest[0]	H	Digest of the first certificate chain.
...	...	...	...
4 + (H * (n -1))	Digest[n-1]	H	Digest of the last (nth) certificate chain.

GET_CERTIFICATE Sequence Diagram


Requester
Responder
Requester Buffer Size = 0x1000
GET CERTIFICATE(0, 0x1000)
Responder Buffer Size = 0x800
CERTIFICATE (0x800, 0x200)
CertificateLength = 0xA00
PortionLength = 0x800
RemainderLength = 0x200
GET_CERTIFICATE (0x800, 0x200)
Requests remaining portion, Offset 0x800, Length 0x0200
CERTIFICATE (0x200, 0)
PortionLength = 0x200
RemainderLength = 0

GET_CERTIFICATE Request

This Request is used to retrieve Certificate Chains.

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x82 = GET_CERTIFICATE
2	Param1	1	Slot number of the target certificate chain to read from. The value in this field shall be between 0 and 7 inclusive.
3	Param2	1	Reserved
4	Offset	2	Offset in bytes from the start of the certificate chain to where the read request message begins. The Responder should send its certificate chain starting from this offset. For the first GET_CERTIFICATE request, the Requester must set this field to 0. For non-first requests, Offset is the sum of PortionLength values in all previous GET_CERTIFICATE responses.
6	Length	2	Length of certificate chain data, in bytes, to be returned in the corresponding response. Length is an unsigned 16-bit integer. This is the smaller of the following two values: capacity of Requester's internal buffer for receiving Responder's certificate chain, and, RemainderLength of the preceding GET_CERTIFICATE response. For the first GET_CERTIFICATE request, the Requester should use the capacity of the Requester's receiving buffer. If offset=0 and length=0xFFFF, the Requester is requesting the entire chain

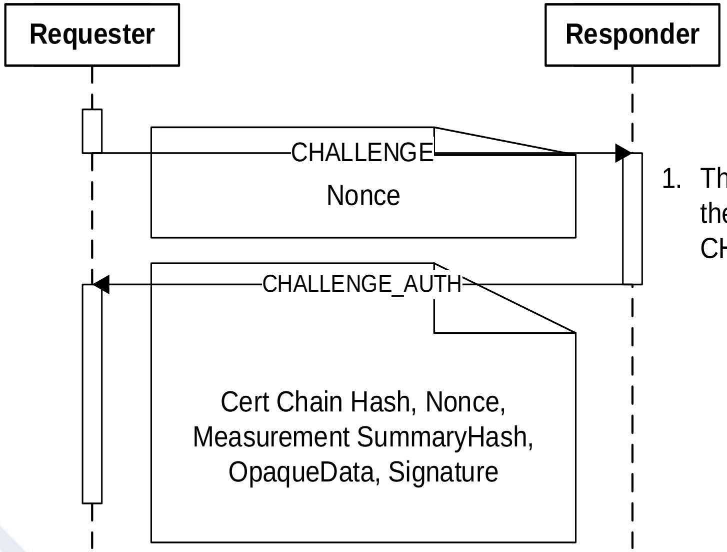
Successful CERTIFICATE

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x02 = CERTIFICATE
2	Param1	1	Slot number of the certificate chain returned.
3	Param2	1	Reserved.
4	PortionLength	2	Number of bytes of this portion of certificate chain. This should be less than or equal to Length received as part of the request. For example, the Responder might set this field to a value less than Length received as part of the request due limitations on the Responder's internal buffer.
6	RemainderLength	2	Number of bytes of the certificate chain that have not been sent yet after the current response. For the last response, this field shall be 0 as an indication to the Requester that the entire certificate chain has been sent.
8	CertChain	Portion Length	Requested contents of target certificate chain, formatted in DER. This field is big endian.

CHALLENGE Sequence Diagram

1. The Requester sends a CHALLENGE request message.

2. The Requester verifies signature against expected values.


Requester
Responder
CHALLENGE
Nonce
CHALLENGE_AUTH
Cert Chain Hash, Nonce, Measurement SummaryHash, OpaqueData, Signature

1. The Responder computes signature using the Nonce and generates a CHALLENGE_AUTH response message

CHALLENGE Request

This Request is used to authenticate an endpoint.

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x83 = CHALLENGE
2	Param1	1	Slot number of the Responder's certificate chain that shall be used for authentication.
3	Param2	1	Requested Measurement Summary Hash Type: 0 = No Measurement Summary Hash, 1 = TCB Component Measurement Hash, 0xFF = All measurements Hash. All other values reserved. When Responder does not support any measurements, Requester shall set this value to 0.
4	Nonce	32	The Requester should choose a random value.

Successful CHALLENGE_AUTH

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x03 = CHALLENGE_AUTH
2	Param1	1	Shall contain the Slot number in the Param1 field of the corresponding CHALLENGE request. This value can be used, by the Requester, to check that the certificate matched what was requested.
3	Param2	1	Slot mask. The bit in position K of this byte shall be set to 1b if and only if slot number K contains a certificate chain for the protocol version in the SPDMVersion field. (Bit 0 is the least significant bit of the byte.) .
4	CertChainHash	H	Hash of the certificate chain used for authentication. This field is big endian. This value can be used, by the Requester, to check that the certificate matched what was requested.
4 + H	Nonce	32	Responder-selected random value.
36 + H	MeasurementSummaryHash	H	When the Responder does not support measurement or requested param2 = 0, the field shall be absent.When the requested param2 = 1, this field shall be the combined hash of all measurements of all measurable components considered to be in the TCB required to generate this response. When the requested param2 = 1 and there are no measurable components in the TCB required to generate this response, this field shall be 0. When requested param2 = 0xFF; the hash is computed using Concatenation(Measurement 1, Measurement 2, ..., Measurement N) of all supported measurements.
36 + 2H	OpaqueLength	2	Size of the OpaqueData field. The value shall not be greater than 1024 bytes.
38 + 2H	OpaqueData	OpaqueLength	Free-form field, if present. The Responder may include Responder-specific information and/or information defined by its transport.
38 + 2H + OpaqueLength	Signature	S	S is the size of the asymmetric signing algorithm output the Responder selected via the last ALGORITHMS response message to the Requester. Signature generation and verification processes are defined in the CHALLENGE_AUTH Signature generation and CHALLENGE_AUTH Signature verification clauses, respectively..

Possible Request Orderings

The possible request orderings after Power on Reset are listed below explicitly:

- GET_VERSION, GET_CAPABILITIES, NEGOTIATE_ALGORITHMS, GET_DIGESTS, GET_CERTIFICATE, CHALLENGE
- GET_VERSION, GET_CAPABILITIES, NEGOTIATE_ALGORITHMS, GET_DIGESTS, CHALLENGE
- GET_VERSION, GET_CAPABILITIES, NEGOTIATE_ALGORITHMS, CHALLENGE
- GET_DIGESTS, GET_CERTIFICATE, CHALLENGE
- GET_DIGESTS, CHALLENGE
- GET_DIGESTS
- CHALLENGE

Request ordering and message transcript computation rules for M1/M2 Part 1

Requests	Implementation Requirements	M1/M2 = Concatenate (A, B, C)
Power on Reset	NA	M1/M2 = null
GET_VERSION issue	The Requester may choose to issue this request any time, to allow Requester / Responder to determine an agreed upon Negotiated state. A Requester may detect out of synch condition typically when signature verification fails or when the Responder provides an unexpected error response.	M1/M2 = null
GET_VERSION, GET_CAPABILITIES, NEGOTIATE_ALGORITHMS	Requester shall always issue these requests in the order shown.	A = Concatenate (GET_VERSION, VERSION, GET_CAPABILITIES, CAPABILITIES, NEGOTIATE_ALGORITHMs, ALGORITHMS)
GET_VERSION, GET_CAPABILITIES, NEGOTIATE_ALGORITHMS	Requester may skip issuing these requests after a new Power on Reset, if the Responder has previously indicated CACHE_CAP = 1. In this case the Requester and Responder shall proceed with the previously Negotiated State	A = null

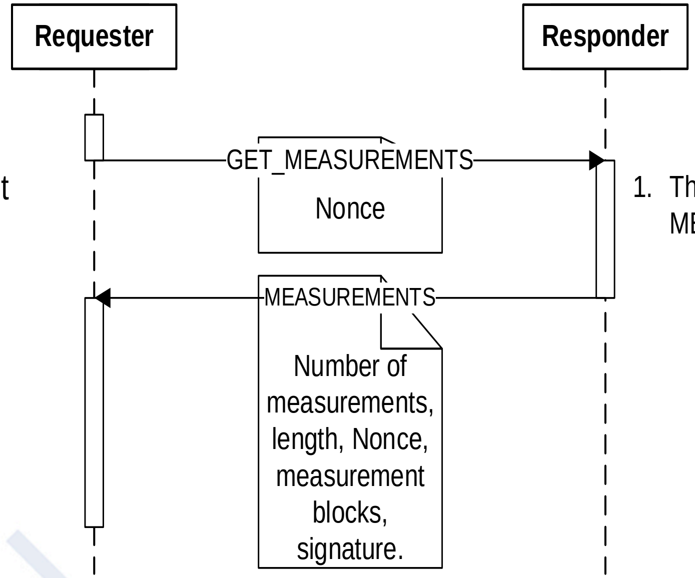
Request ordering and message transcript computation rules for M1/M2 Part 2

Requests	Implementation Requirements	M1/M2 = Concatenate (A, B, C)
GET_DIGEST, GET_CERTIFICATE	Requester shall always issue these requests in the order shown after NEGOTIATE_ALGORITHMS request completion or immediately after Power on Reset if it chose to skip the previous three requests.	B = Concatenate (GET_DIGEST, DIGEST, GET_CERTFICATE, CERTIFICATE)
GET_DIGEST, GET_CERTFICATE	Requester may choose to skip both requests after a new Power on Reset if it is capable of using previously cached response to these requests.	B = Null
GET_DIGEST, GET_CERTIFICATE	Requester may choose to skip GET_CERTIFICATE request after a new Power on Reset if it is capable of using previously cached CERTIFICATE response.	B = (GET DIGEST, DIGEST)
CHALLENGE	Requester shall issue this request to complete security verification of current requests and responses.	C = (CHALLENGE, CHALLENGE_AUTH).
CHALLENGE completion	Completion of CHALLENGE resets M1 and M2	M1/M2 = null
CHALLENGE	Requester may choose to skip this request and forgo security verification of previous requests and responses. Requester may typically skip CHALLENGE when it issues GET_DIGEST directly after Power on Reset.	NA
GET_MEASUREMENTS	If the Requester chooses to issue GET_MEASUREMENTS and skips CHALLENGE completion, M1 and M2 are reset to null	M1/M2 = null

GET_MEASUREMENTS Sequence Diagram

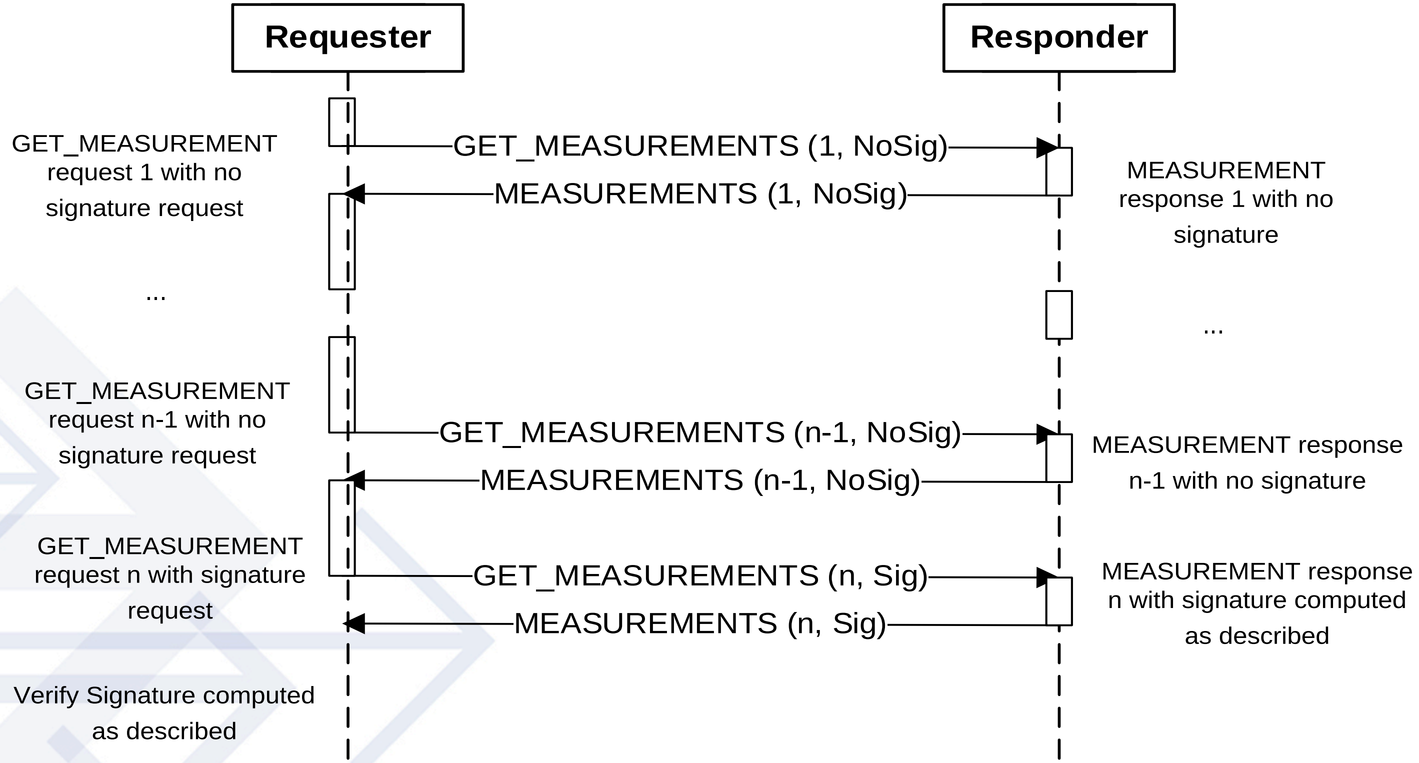
1. The Requester sends a GET_MEASUREMENTS request message.

2. Verify signature and verify measurements match expected values.


Requester
Responder
GET_MEASUREMENTS
Nonce
MEASUREMENTS
Number of measurements, length, Nonce, measurement blocks, signature.

1. The Responder sends a MEASUREMENTS response message.

GET_MEASUREMENTS with and without Signature Sequence Diagram



Requester
Responder
GET_MEASUREMENT request 1 with no signature request
GET_MEASUREMENTS (1, NoSig)
MEASUREMENTS (1, NoSig)
MEASUREMENT response 1 with no signature
...
...
GET_MEASUREMENT request n-1 with no signature request
GET_MEASUREMENTS (n-1, NoSig)
MEASUREMENTS (n-1, NoSig)
MEASUREMENT response n-1 with no signature
GET_MEASUREMENT request n with signature request
GET_MEASUREMENTS (n, Sig)
MEASUREMENTS (n, Sig)
MEASUREMENT response n with signature computed as described
Verify Signature computed as described

GET_MEASUREMENTS Request

This Request is used to retrieve measurements of mutable firmware component(s) that the recipient endpoint is executing.

Measurements on their own are one of several methods to provide identity.

Signing shall use the device private key.

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0xE0 = GET_MEASUREMENTS
2	Param1	1	Request attributes.
3	Param2	1	Measurement operation. A value of 0x0 shall query the Responder for the total number of measurements available. A value of 0xFF shall request all measurements. A value between 0x1 and 0xFE inclusively shall request the measurement at the index corresponding to that value.
4	Nonce	32	The Requester should choose a random value. This field is only present if a signature is required on the response.

GET_MEASUREMENT Request Attributes

it(s)	Value	Description
0	1	If the Responder can generate a signature as indicated in CAPABILITIES message, this bit's value shall indicate to the Responder to generate a signature. The Responder shall generate a signature in the corresponding response. The Nonce field shall be present in the request.
0	0	This bit's value shall be used for Responders incapable of generating a signature as indicated in CAPABILITIES message. For Responders capable of signature generation, this bit's value shall indicate the Requester does not want a signature. The Responder shall not generate a signature in the response. The Nonce field shall be absent in the request.
[7:1]	Reserved	Reserved

Successful MEASUREMENTS

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x60 = MEASUREMENTS
2	Param1	1	When Param2 in the requested measurement operation is 0, this parameter shall return the total number of measurement indices on the device. Otherwise, this field is reserved.
3	Param2	1	Reserved
4	NumberOfBlocks	1	Number of measurement blocks (N) in **MeasurementRecord**. This field shall reflect the number of measurement blocks in **MeasurementRecord**. If Param2 in the requested measurement operation is 0, this field shall be 0.
5	MeasurementRecordLength	3	Size of the MeasurementRecord field in bytes. If Param2 in the requested measurement operation is 0, this field shall be 0.
8	MeasurementRecord	L=Measurement RecordLength	Concatenation of all Measurement Blocks that correspond to the requested Measurement operation. The Measurement Block structure is defined in Measurement block.
8 + L	Nonce	32	The Responder should choose a random value.
40 + L	OpaqueLength	2	Size of the OpaqueData field in bytes. The value shall not be greater than 1024 bytes.
42 + L	OpaqueData	OpaqueLength	Free-form field, if present. The Responder may include Responder-specific information and/or information defined by its transport.
42 + L + OpaqueLength	Signature	S	Signature of the GET_MEASUREMENTS Request and MEASUREMENTS Response messages, excluding the Signature field and signed using the device private key (slot 0 leaf certificate private key). The Responder shall use the asymmetric signing algorithm it selected during the last ALGORITHMS response message to the Requester and S is the size of that asymmetric signing algorithm output.

Measurement Block

- Each Measurement block contains a 1-DWORD descriptor, followed by the cryptographic hash and optionally additional information
- Logical increment of the Measurement index implies bootstrapping of firmware stages
- When returning Measurement log, the requestor specifies the Measurement index that it needs the history for. Each event that caused changes in the Measurement hash is recorded in one Measurement block, distinguished by the step log field.

Offset	Field	Size (bytes)	Value
0	Index	1	Index. This field shall represent the index of the measurement.
1	MeasurementSpecification	1	This field is a bitmask. The value shall indicate the measurement specification that the requested Measurement follows and shall match the selected measurement specification in Algorithms message. Only one bit shall be set in the Measurement Block. •Bit 0 = DMTF. All other bits are reserved.
2	MeasurementSize	2	Size of Measurement, in bytes.
4	Measurement	Measurement Size	For format of this field is defined by MeasurementSpecification

Measurement field format in a Measurement block when the MeasurementSpecification field selects Bit 0 = DMTF

Offset	Field	Size (bytes)	Value
0	DMTFSpecMeasurement ValueType	1	•This field is composed of two parts: bit [7] indicating the representation in DMTFSpecMeasurementValue, and bits [6:0] indicating what is being measured by DMTFSpecMeasurementValue. These values are set independently. These values are interpreted as follows:[7] = 0b: Hash •[7] = 0b: Hash •[7] = 1b : Raw Bit Stream •[6:0] = 00h: immutable ROM •[6:0] = 01h: mutable firmware •[6:0] = 02h: hardware configuration, such as straps, debug modes •[6:0] = 03h : firmware configuration, e.g., configurable firmware policy All other values reserved.
1	DMTFSpecMeasurement ValueSize	2	Size of DMTFSpecMeasurementValue, in bytes. When DMTFSpecMeasurementValueType[7] = 0b: Hash, the DMTFSpecMeasurementValueSize shall be derived from the measurement hash algorithm returned in the ALGORITHM response message.
3	DMTFSpecMeasurement Value	DMTFSpec Measurement ValueSize	DMTFSpecMeasurementValueSize bytes of cryptographic hash or Raw Bit Stream, as indicated in DMTFSpecMeasurementType[7].

ERROR & ERRORCODE

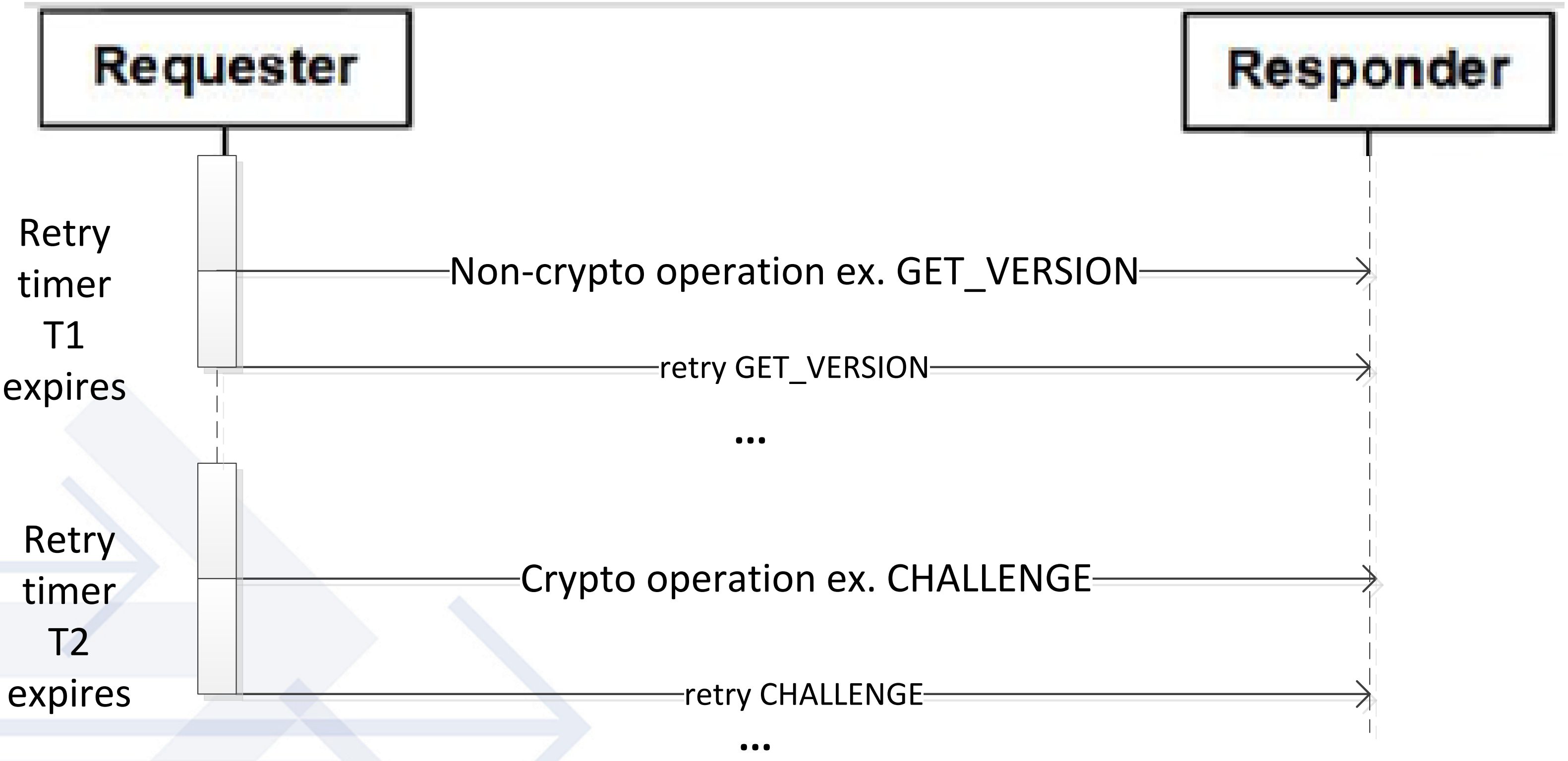
Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x7F = ERROR
2	Param1	1	Error Code. See Table 27.
3	Param2	1	Error Data. See Table 27.
4	ExtendedErrorData	0-32	Optional extended data. See Table 27.

Error code	Value	Description	Error data	ExtendedErrorData
Reserved	00h	Reserved	Reserved	Reserved
InvalidRequest	01h	One or more request fields are invalid	0x00	No extended error data is provided.
Reserved	02h	Reserved	Reserved	Reserved
Busy	03h	The Responder received the request message and the Responder decided to ignore the request message, but the Responder may be able to process the request message if the request message is sent again in the future.	0x00	No extended error data is provided.

ERRORCODE contd.

Error code	Value	Description	Error data	ExtendedErrorData
UnexpectedRequest	04h	The Responder received an unexpected request message. For example, CHALLENGE before N EGOTIATE_ALGORITHMS.	0x00	No extended error data is provided.
Unspecified	05h	Unspecified error occurred.	00h	No extended error data is provided.
Reserved	06h	Reserved	00h	Reserved
UnsupportedRequest	07h	The RequestResponseCode in the Request message is unsupported.	RequestResponseCode in the Request message.	No extended error data is provided
Reserved	08h-40h	Reserved	Reserved	Reserved
MajorVersionMismatch	41h	Requested SPDM Major Version is not supported.	00h	No extended error data provided.
ResponseNotReady	42h	See RESPOND_IF_READY c lause.	00h	See Table 28.
RequestResynch	43h	Responder is requesting Requester to reissue GET_VERSION in order to resynch.	0x00	No extended error data provided.
Reserved	44h-FEh	Reserved	Reserved.	Reserved
Vendor/Other Standards Defined	FFh	Vendor or Other Standards defined	This field shall indicate the registry or standard body using one of the values in the ID column of Table 29.	See Table 30 for format definitio

Timeouts and Retries



T1 = RTT (round trip time) + ST1 (non-crypto processing time at responder)

T2 = RTT (round trip time) + CT (crypto processing time at responder)

Timeouts and Retries
Error (ResponseNotReady); RESPOND_IF_READY

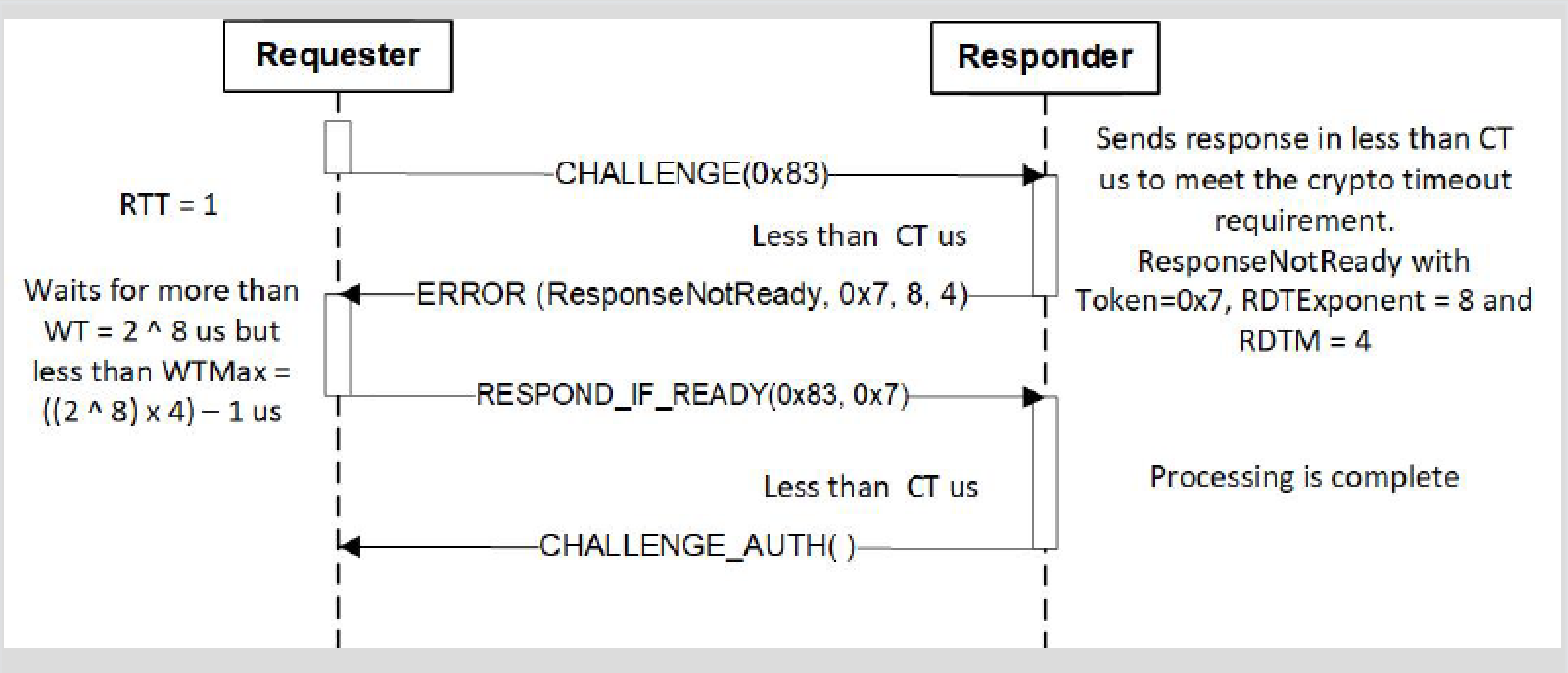

Requester
Responder
CHALLENGE(0x83)
RTT = 1
Sends response in less than CT us to meet the crypto timeout requirement. ResponseNotReady with Token=0x7, RDTExponent = 8 and RDTM = 4
Less than CT us
Waits for more than WT = 2 ^ 8 us but less than WTMax = ((2 ^ 8) x 4) – 1 us
ERROR (ResponseNotReady, 0x7, 8, 4)
RESPOND_IF_READY(0x83, 0x7)
Processing is complete
Less than CT us
CHALLENGE_AUTH( )

Figure 9 — RESPOND_IF_READY flow leading to completion

Timing Specification Part 1

Timing Parameter	Ownership	Value	Units	Description
RTT	Requester	See Description	See Description	This is the worst case round trip transport timing. The max value shall be the worst case total time for the complete transmission and delivery of an SPDM message round trip at the transport layer(s). The actual value for this parameter is transport/media specific.
ST1	Responder	100	ms	This shall be the maximum amount of time the Responder has to provide a response to requests that do not require cryptographic processing, such as GET_CAPABILITIES, GET_VERSION or NEGOTIATE_ALGORITHMS.
T1	Requester	RTT + ST1	ms	This shall be the minimum amount of time the Requester shall wait before issuing a retry for requests that do not require cryptographic processing. For details, see ST1.
CT	Responder	$2^{CTExponent}$	us	This is the cryptographic timeout in microseconds. CTExponent is reported in the CAPABILITIES message. This timing parameter shall be the maximum amount of time the Responder has to provide any response requiring cryptographic processing, such as GET_MEASUREMENTS and CHALLENGE.
T2	Requester	RTT + CT	us	This shall be the minimum amount of time the Requester shall wait before issuing a retry for requests that require cryptographic processing. For details, see CT.

Timing Specification Part 2

Timing Parameter	Ownership	Value	Units	Description
RDT	Responder	$2^{RDTExponent}$	us	This is the Recommended Delay in microseconds. When the Responder is unable to complete cryptographic processing response within the CT time, it shall provide RDTExponent as part of the ERROR Response. See Table 28 for the RDTExponent value. For details, see ErrorCode=ResponseNotReady.
WT	Requester	RDT	us	This is the amount of time the Requester should wait before issuing RESPOND_IF_READY request. The Requester shall measure this time parameter from the reception of the ERROR response to the transmission of RESPOND_IF_READY request. The Requester may take into account the transission time of the ERROR from the Responder to Requester when calculating WT. For details, see RDT.
WT_{Max}	Requester	(RDT * RDTM) - RTT	us	This is the maximum wait time the Requester has to to issue RESPOND_IF_READY request unless the Requester issued a successful RESPOND_IF_READY earlier. After this time the Responder is allowed to drop the response. The Requester shall take into account the transmission time of the ERROR from the Responder to Requester when calculating WTMax. The value of RDTM is given in Table 28. The Responder should ensure WT<sub>Max</sub> does not result less than WT in determination of RDTM. For details, see ErrorCode=ResponseNotReady.

Retries and timing

- SPDM requests may be retried; SPDM responses may not.
- There are two SPDM request retry timers:
  - T1 – for SPDM requests that do not involve cryptographic processing
  - T2 – for SPDM requests that require cryptographic processing
- The T1 retry timer is the sum of:
  - RTT – worst-case round-trip time, which is transport layer specific
  - ST1 – amount of time responder can take to process non crypto requests
- The T2 retry timer is the sum of:
  - RTT
  - CT – amount of time responder can take to process crypto requests

Retries and timing

- An SPDM request that requires cryptographic processing may take longer than expected at the responder
- A responder may provide a Response Not Ready error within T2 and let the requester know when to try again with a Respond if Ready request
- With Response Not Ready, the responder provides RDTexponent, a recommended delay to the requester; and RDTM, a multiplier
- The requester may retry with Respond if Ready within a window bounded by WT and WTMax:
  - WT = 2**RDTexponent
  - WTMax = (Multiplier * WT) - RTT

Future Work

- SPDM1.1 Scope
  - Protection: Key establishment and agreement / Encryption / Integrity
  - Mutual Authentication
- SPDM1.x Scope
  - Measurement log
  - Set certificate command
  - Measurement manifest (Local attestation)

SPDM 1.1: Session Key Exchange Protocols

July 2020

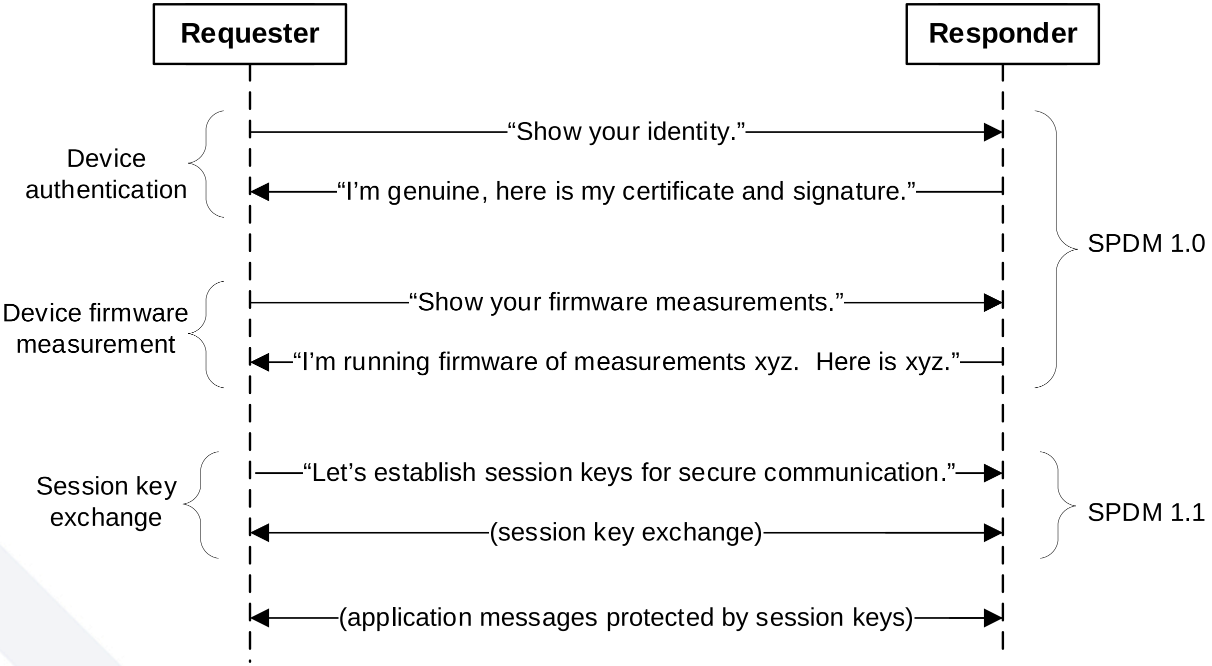
Disclaimer

- The information in this presentation represents a snapshot of work in progress within the DMTF.
- This information is subject to change without notice. The standard specifications remain the normative reference for all information.
- For additional information, see the DMTF website.
- This information is a summary of the information that will appear in the specifications. See the specifications for further details.

SPDM 1.1 Feature Additions

- Sessions
  - Key exchange
    - Exchange keys to enable encryption by the transport
    - SIGMA and Pre-shared key options
    - Suitable for adoption by many industry transport layers
  - Key confirmation
  - Key update
  - Key schedule
- Mutual authentication
- Derivation of additional keys

Session Key Exchange


Requester
Responder
Device authentication
"Show your identity."
"I'm genuine, here is my certificate and signature."
SPDM 1.0
Device firmware measurement
"Show your firmware measurements."
"I'm running firmware of measurements xyz. Here is xyz."
Session key exchange
"Let's establish session keys for secure communication."
(session key exchange)
SPDM 1.1
(application messages protected by session keys)

Objective: Establish session keys that are known to only Requester and Responder

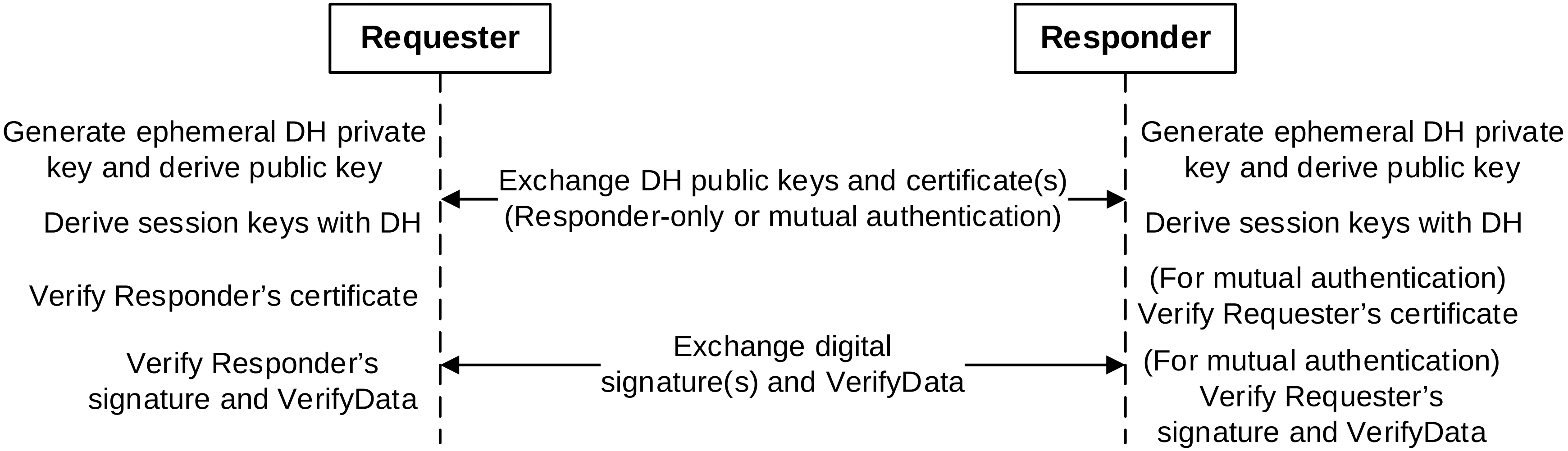
- Either endpoint may abort a session at any time.
- Authentication happens with session key exchange.
- Requester authenticates Responder. Optionally, Responder may authenticate Requester.

SPDM 1.1 specifies the following session key exchange schemes:

1. **SIGMA** option: based on ephemeral Diffie-Hellman and digital signatures.
2. **Pre-shared secret** option: based on a pre-shared secret known to both endpoints.

*SIGMA: http://webee.technion.ac.il/~hugo/sigma-pdf.pdf

SIGMA Option for Session Key Exchange


Requester
Responder
Generate ephemeral DH private key and derive public key
Derive session keys with DH
Verify Responder's certificate
Verify Responder's signature and VerifyData
Exchange DH public keys and certificate(s) (Responder-only or mutual authentication)
Exchange digital signature(s) and VerifyData
Generate ephemeral DH private key and derive public key
Derive session keys with DH
(For mutual authentication) Verify Requester's certificate
(For mutual authentication) Verify Requester's signature and VerifyData

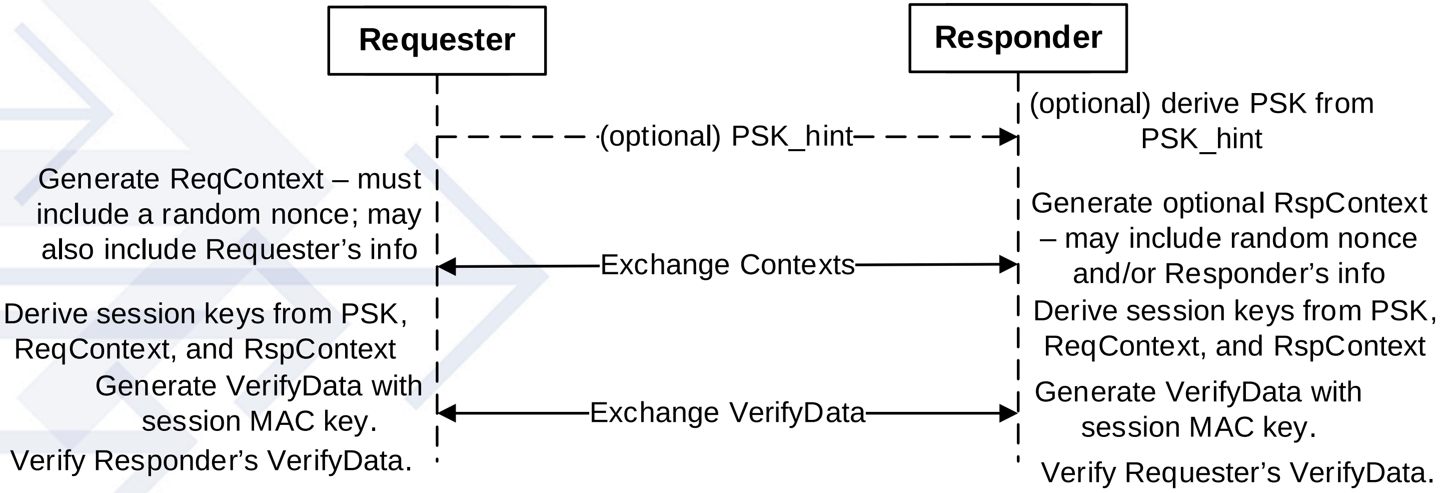
- Diagram above illustrates high-level sequence; arrows do not map to actual commands
- Based on SIGMA and TLS 1.3 handshake protocols
- Session key agreement uses Diffie-Hellman scheme (ECDHE or FFDHE)
- Features mutual or one-way (Responder to Requester) authentication
- Features forward secrecy
- Features session key confirmation through VerifyData exchanges
- Requester capabilities: RSA and/or ECC, HMAC, RNG
- Responder capabilities: RSA or ECC, HMAC, RNG (if mutual authentication or forward secrecy is required)
- Responder examples: graphics card, SSD, FPGA

Pre-Shared Key Option: Introduction

- Pre-shared key (PSK) is a secret known to both the Requester and the Responder, before the session key exchange flow is executed
- Provisioning of PSK is out of scope of SPDM 1.1.
    - Implementer’s policy is also out of scope of SPDM 1.1.
- Responder benefits: low cost (HMAC + unique device secret or secure storage for PSK)
- Responder examples: integrated webcam, integrated fingerprint scanner, devices soldered on board, CPU, GPU, NIC
- Requester capabilities: HMAC, RNG, secure storage

Pre-Shared Key Option for Session Key Exchange

- Diagram below illustrates high-level sequence; arrows do not map to actual commands.
- Some provisioning schemes require Requester to send PSK_hint to Responder during session key exchange flow, so the Responder can derive PSK. Content of PSK_hint depends on the underlying PSK provisioning scheme and is out of scope of SPDM.
- Requester context and Responder context are described in diagram below.
- Features session key confirmation through VerifyData exchanges
- Session keys are derived from PSK and contexts.



Requester

Responder

(optional) PSK_hint

(optional) derive PSK from PSK_hint

Generate ReqContext – must include a random nonce; may also include Requester's info

Generate optional RspContext – may include random nonce and/or Responder's info

Exchange Contexts

Derive session keys from PSK, ReqContext, and RspContext

Derive session keys from PSK, ReqContext, and RspContext

Generate VerifyData with session MAC key.

Exchange VerifyData

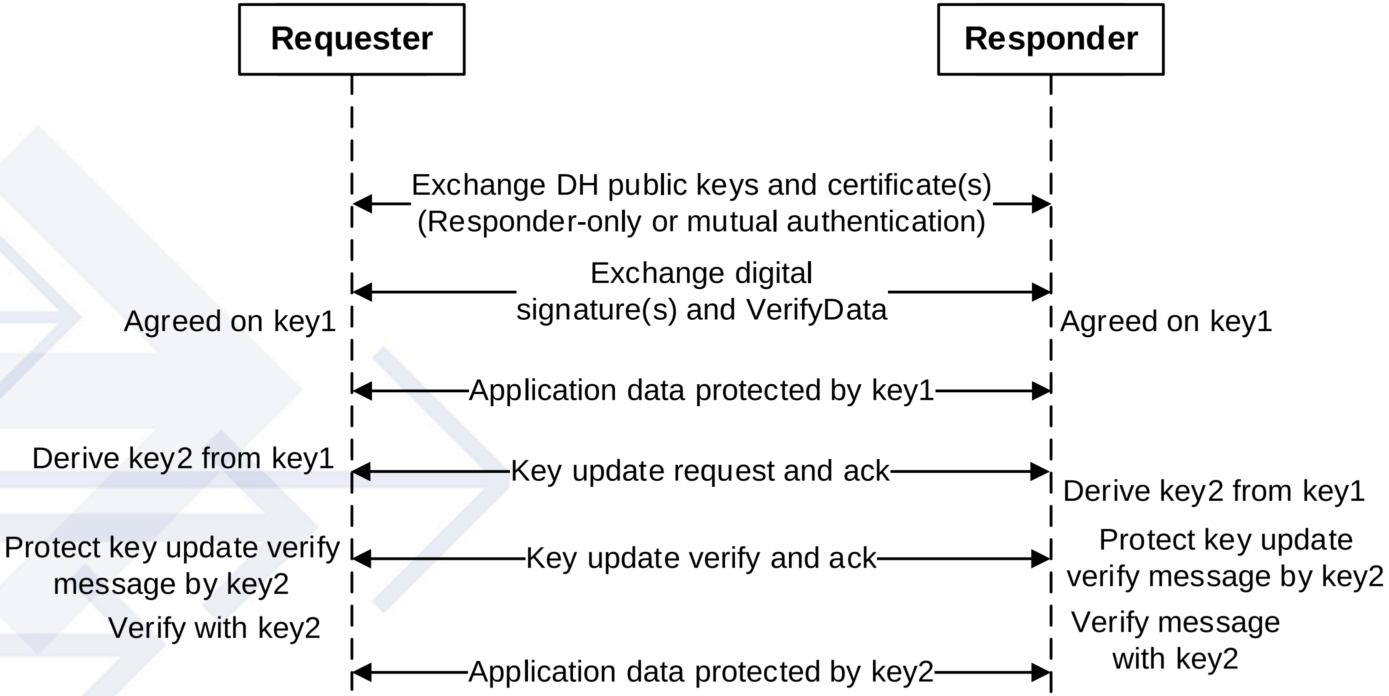
Generate VerifyData with session MAC key.

Verify Responder's VerifyData.

Verify Requester's VerifyData.

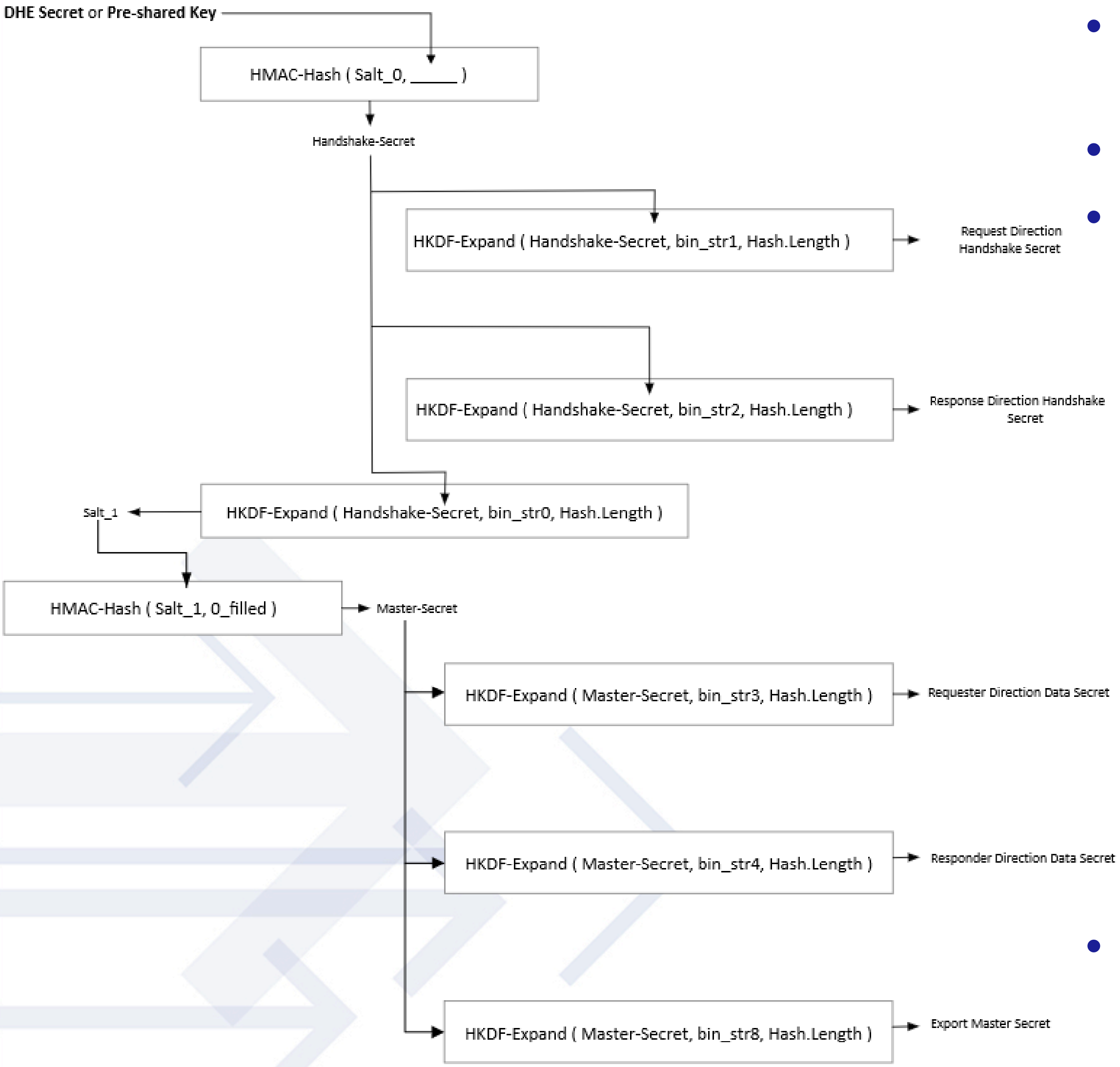
Key Update

- Either Requester or Responder may initialize a key update, updating the session keys to new values, for reasons such as counter overflow.
- No SIGMA or PSK. The new keys are derived from the current keys.
- The new keys are confirmed by both endpoints before used for protecting application data.



Requester
Responder
Exchange DH public keys and certificate(s)
(Responder-only or mutual authentication)
Exchange digital
signature(s) and VerifyData
Agreed on key1
Agreed on key1
Application data protected by key1
Derive key2 from key1
Key update request and ack
Derive key2 from key1
Protect key update verify
message by key2
Key update verify and ack
Protect key update
verify message by key2
Verify with key2
Verify message
with key2
Application data protected by key2

Key Schedule


DHE Secret or Pre-shared Key
HMAC-Hash (Salt_0, _____)
Handshake-Secret
HKDF-Expand (Handshake-Secret, bin_str1, Hash.Length)
Request Direction Handshake Secret
HKDF-Expand (Handshake-Secret, bin_str2, Hash.Length)
Response Direction Handshake Secret
Salt_1
HKDF-Expand (Handshake-Secret, bin_str0, Hash.Length)
HMAC-Hash (Salt_1, 0_filled)
Master-Secret
HKDF-Expand (Master-Secret, bin_str3, Hash.Length)
Requester Direction Data Secret
HKDF-Expand (Master-Secret, bin_str4, Hash.Length)
Responder Direction Data Secret
HKDF-Expand (Master-Secret, bin_str8, Hash.Length)
Export Master Secret

- Based on HMAC-Hash and HKDF-Expand
- Input: DHE secret or PSK
- Output:
  - **Handshake Secrets**: used to protect handshake messages
  - **Data Secrets**: used to protect application message
  - **Export Master Secret**: additional key derived for custom usages defined by vendor
- Different secrets for the two directions of communication, respectively

References

- PMCI Standards: where to find all the specs, white papers and presentations
  - https://www.dmtf.org/standards/pmci
- SPDM
  - DSP0274 (Security Protocol and Data Model (SPDM)): https://www.dmtf.org/dsp/DSP0274
  - DSP0275 (Security Protocol and Data Model (SPDM) over MCTP Binding Specification): https://www.dmtf.org/dsp/DSP0275
  - DSP0276 (Secured Messages using SPDM over MCTP Binding Specification): https://www.dmtf.org/dsp/DSP0276
  - DSP0277 (Secured Messages using SPDM Specification): https://www.dmtf.org/standards/pmci when released
  - DSP2058 (Security Protocol and Data Model (SPDM) Architecture White Paper): https://www.dmtf.org/standards/pmci when released

Backup

SNIA STORAGE SECURITY SUMMIT

Wednesday, May 11, 2022 • Virtual

SPDM Storage Security

Extending SPDM to Storage Use Cases

Presented by Brett Henning (Broadcom) and Scott Phuong (Cisco)

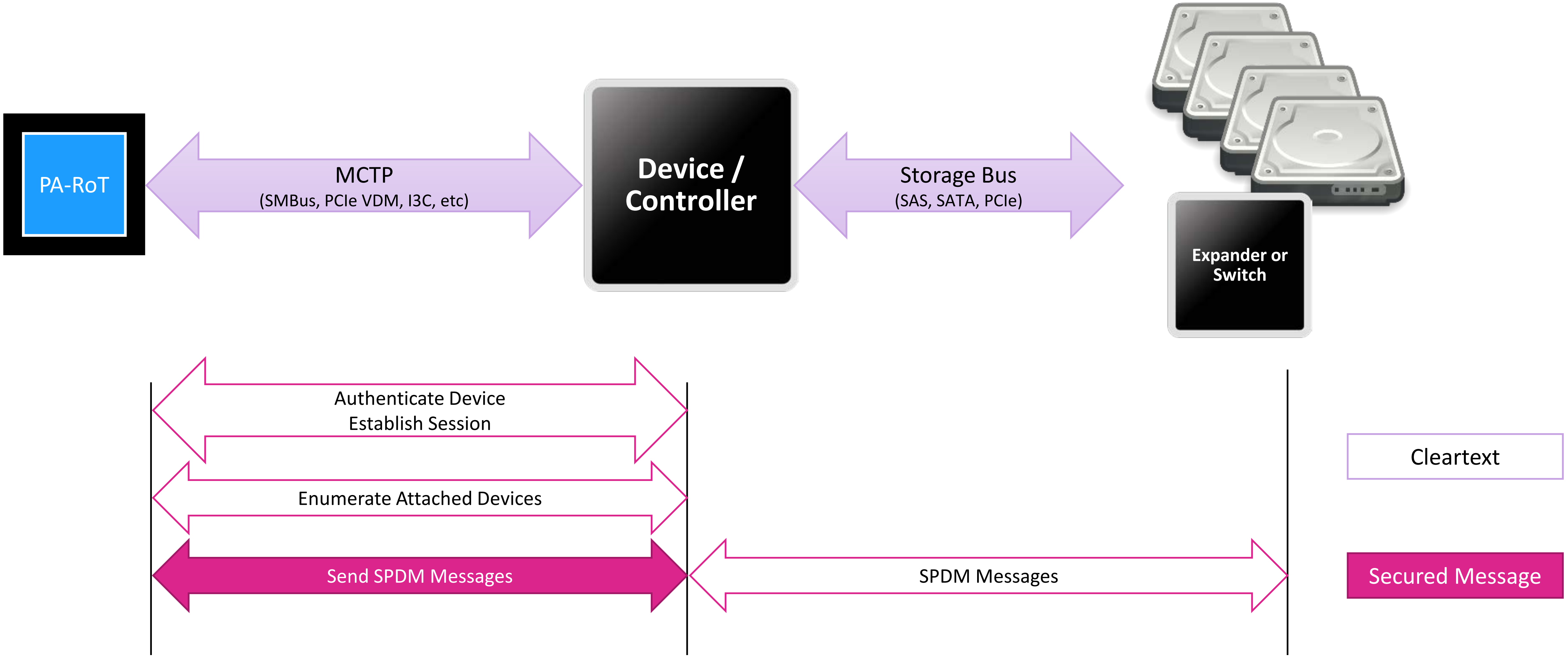
A SNIA Event

SNIA Legal Notice

- The material contained in this tutorial is copyrighted by the SNIA unless otherwise noted.
- Member companies and individual members may use this material in presentations and literature under the following conditions:
  - Any slide or slides used must be reproduced in their entirety without modification
  - The SNIA must be acknowledged as the source of any material used in the body of any document containin material from these presentations.
- This presentation is a project of the SNIA.
- Neither the author nor the presenter is an attorney and nothing in this presentation is intended to be or should be construed as legal advice or an opinion of counsel. If you need legal advice or a legal opinion please contact your attorney.
- The information presented herein represents the author's personal opinion and current understanding of the relevant issues involved. The author, the presenter, and the SNIA do not assume any responsibility or liability for damages arising out of any reliance on or use of this information.

  NO WARRANTIES, EXPRESS OR IMPLIED. USE AT YOUR OWN RISK.

Extending SPDM to Storage



PA-RoT

MCTP
(SMBus, PCIe VDM, I3C, etc)

Device / Controller

Storage Bus
(SAS, SATA, PCIe)

Expander or Switch

Authenticate Device
Establish Session

Enumerate Attached Devices

Send SPDM Messages

SPDM Messages

Cleartext

Secured Message

Storage Protocol Support

- SPDM is supported in storage protocols
  - SECURITY PROTOCOL IN/OUT support SPDM
    - In SCSI Primary Commands – 6
    - SECURITY PROTOCOL E8h

SNIA® STORAGE SECURITY SUMMIT

Wednesday, May 11, 2022 • Virtual

Please take a moment to rate this session.

Your feedback is important to us.