Document Identifier: DSP0274

Date: 2019-09-21

Version: 0.95.0

Security Protocol and Data Model (SPDM) Specification

Information for Work-in-Progress version:

IMPORTANT: This document is not a standard. It does not necessarily reflect the views of the DMTF or its members. Because this document is a Work in Progress, this document may still change, perhaps profoundly and without notice. This document is available for public review and comment until superseded.

Provide any comments through the DMTF Feedback Portal: http://www.dmtf.org/standards/feedback

Supersedes: 0.9.0

Document Class: Normative

Document Status: Work-In-Progress

Document Language: en-US

CONTENTS

Foreword

The Platform Management Components Intercommunication (PMCI) Working Group of the DMTF prepared the *Security Protocol and Data Model (SPDM) Specification* (DSP0274). DMTF is a not-for-profit association of industry members that promotes enterprise and systems management and interoperability. For information about the DMTF, see https://www.dmtf.org.

Acknowledgments

The DMTF acknowledges these individuals' contributions to this document:

Contributors:

- Richelle Ahlvers — Broadcom Inc.
- Lee Ballard — Dell Technologies
- Patrick Caporale — Lenovo
- Yu-Yuan Chen — Intel Corporation
- Nigel Edwards — Hewlett Packard Enterprise
- Daniil Egranov — Arm Limited
- Brett Henning — Broadcom Inc.
- Jeff Hilland — Hewlett Packard Enterprise
- Yuval Itkin — Mellanox Technologies
- Theo Koulouris — Hewlett Packard Enterprise
- Luis Luciani — Hewlett Packard Enterprise
- Masoud Manoo — Lenovo
- Donald Mathews — Advanced Micro Devices, Inc
- Edward Newman — Hewlett Packard Enterprise
- Mahesh Natu — Intel Corporation
- Scott Phuong — Cisco Systems Inc.
- Jeffrey Plank — Microchip
- Viswanath Ponnuru — Dell Technologies
- Xiaoyu Ruan — Intel Corporation
- Nitin Sarangdhar — Intel Corporation
- Hemal Shah — Broadcom Inc.
- Srikanth Varadarajan — Intel Corporation

Abstract

The *Security Protocol and Data Model (SPDM) Specification* defines *messages*, data objects, and sequences for performing message exchanges between *devices* over a variety of transport and physical media. The description of message exchanges includes *authentication* of hardware identities and measurement for firmware identities. The SPDM enables efficient access to low-level security capabilities and operations. The SPDM can be used with other mechanisms, including non-PMCI- and DMTF-defined mechanisms.

Document conventions

- Document titles appear in *italics*.
- The first occurrence of each important term appears in *italics* with a link to its definition.
- ABNF rules appear in a monospaced font.

1. Scope

This specification describes how to use messages, data objects, and sequences to exchange messages between two devices over a variety of transports and physical media. This specification contains the message exchanges, sequence diagrams, message formats, and other relevant semantics for such message exchanges, including authentication of hardware identities and firmware measurement for firmware identities.

Other specifications define the mapping of these messages to different transports and physical media. This specification provides information to enable security policy enforcement but does not specify individual policy decisions.

2. Normative references

The following referenced documents are indispensable for the application of this specification. For dated or versioned references, only the edition cited (including any corrigenda or DMTF update versions) applies. For references without a date or version, the latest published edition of the referenced document (including any corrigenda or DMTF update versions) applies.

- *ISO/IEC Directives, Part 2, Principles and rules for the structure and drafting of ISO and IEC documents*, https://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype
- IETF RFC5234, *Augmented BNF for Syntax Specifications: ABNF*, January 2008, https://tools.ietf.org/html/rfc5234
- 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
- TCG Algorithm Registry, Family "2.0", Level 00 Revision 01.27, February 7, 2018 https://trustedcomputinggroup.org/resource/tcg-algorithm-registry/

ASN.1 — ISO-822-1-4

- ITU-T X.680

  Available at: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.680-201508-I!!PDF-E&type=items

- ITU-T X.681

  Available at: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.681-201508-I!!PDF-E&type=items

- ITU-T X.682

  Available at: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.682-201508-I!!PDF-E&type=items

- ITU-T X.683

  Available at: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.683-201508-I!!PDF-E&type=items

DER — ISO-8825-1

- ITU-T X.690

  Available at: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.690-201508-I!!PDF-E&type=items

X509v3 — ISO-9594-8

- ITU-T X.509

  Available at: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-201210-I!!PDF-E&type=items

ECDSA

- NIST-FIPS-186-4, Section 6

  Available at: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

- NIST P256, secp256r1; NIST P384, secp384r1; NIST P521, secp521r1: NIST-FIPS-186-4, Appendix D

  Available at: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

RSA

- As defined in TCG Algorithm Registry, Family "2.0", Level 00 Revision 01.27, February 7, 2018, Table 3: https://trustedcomputinggroup.org/wp-content/uploads/TCG_Algorithm_Registry_Rev_1.22.pdf

SHA2-256, **SHA2-384**, and **SHA2-512**

- FIPS PUB 180-4 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Secure Hash Standard (SHS)

  Available at: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

SHA3-256, **SHA3-384**, and **SHA3-512**

- FIPS PUB 202FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions

  Available at: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

3. Terms and definitions

In this document, some terms have a specific meaning beyond the normal English meaning. This clause defines those terms.

The terms "shall" ("required"), "shall not," "should"("recommended"), "should not" ("not recommended"), "may," "need not" ("not required"), "can" and "cannot" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 7. The terms in parentheses are alternatives for the preceding term, for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that ISO/IEC Directives, Part 2, Clause 7 specifies additional alternatives. Occurrences of such additional alternatives shall be interpreted in their normal English meaning.

The terms "clause," "subclause," "paragraph," and "annex" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 6.

The terms "normative" and "informative" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 3. In this document, clauses, subclauses, or annexes labeled "(informative)" do not contain normative content. Notes and examples are always informative elements.

This specification uses these terms:

Term	Definition
authentication	Process of determining whether an entity is who or what it claims to be.
authentication initiator	Endpoint that initiates the authentication process by challenging another endpoint.
byte	Eight-bit quantity. Also known as an *octet*. **Note:** SPDM specifications shall use the term byte, not octet.
certificate	Digital form of identification that provides information about an entity and certifies ownership of a particular an asymmetric key-pair.
certificate authority (CA)	Trusted third-party entity that issues certificates.
certificate chain	Series of two or more certificates. Each certificate is signed by the preceding certificate in the chain.
certificate signing request (CSR)	One of the first steps towards getting a certificate.
component	Similar to the PCI Express specification's definition, a physical entity.
device	Physical entity such as a network card or a fan.
endpoint	Logical entity that communicates with other endpoints over one or more transport protocol.
intermediate certificate	Certificate that is neither a root certificate nor a leaf certificate.
leaf certificate	Last certificate in a certificate chain.
message	See SPDM message.
message body	Portion of an SPDM message that carries additional data.
message originator	Original transmitter, or source, of an SPDM message.
most significant byte (MSB)	Highest order *byte* in a number consisting of multiple bytes.
nonce	Number that is unpredictable to entities other than its generator. The probability of the same number occurring more than once is negligible. Nonce may be generated by combining a pseudo random number of at least 64 bits, optionally concatenated with a monotonic counter of size suitable for the application.
nibble	Computer term for a four-bit aggregation, or half of a byte.
payload	Information-bearing fields of a message. These fields are separate from the fields and elements, such as address fields, framing bits, checksums, and so on, that transport the message from one point to another. In some instances, a field can be both a payload field and a transport field.
physical transport binding	Specifications that define how a base messaging protocol is implemented on a particular physical transport type and medium, such as SMBus/I^{2}C, PCI Express™ Vendor Defined Messaging, and so on.
SPDM message	Unit of communication in SPDM communications.
SPDM message payload	Portion of the message body of an SPDM message. This portion of the message is separate from those fields and elements that identify the SPDM version, the SPDM request and response codes, and the two parameters.
SPDM request message	Message that is sent to an endpoint to request a specific SPDM operation. A corresponding SPDM response message acknowledges receipt of an SPDM request message.
SPDM response message	Message that is sent in response to a specific SPDM request message. This message includes a `Response Code` field that indicates whether the request completed normally.
Platform Management Component Intercommunications (PMCI)	Name of a working group under the Distributed Management Task Force that defines standardized communication protocols, low-level data models, and transport definitions that support communications with and between management controllers and management devices that form a platform management subsystem within a managed computer system.
Requester	Original transmitter, or source, of an SPDM request message. It is also the ultimate receiver, or destination, of an SPDM response message.
Responder	Ultimate receiver, or destination, of an SPDM request message. It is also the original transmitter, or source of an SPDM response message.
root certificate	First certificate in a certificate chain, which is self-signed.
Root of Trust (RoT)	Immutable entity that forms the basis for trust in a component. For example, a hardware block or an immutable firmware executable.

4. Symbols and abbreviated terms

This specification uses these abbreviations:

Abbreviation	Definition
CA	*certificate authority*
CSR	*certificate signing request*
MSB	*most significant byte*
PMCI	Platform Management Component Intercommunications
RoT	*Root of Trust*
SPDM	Security Protocol and Data Model

5. Conventions

The following conventions apply to all SPDM specifications.

5.1. Reserved and unassigned values

Unless otherwise specified, any reserved, unspecified, or unassigned values in enumerations or other numeric ranges are reserved for future definition by the DMTF.

Unless otherwise specified, reserved numeric and bit fields shall be written as zero (`0`) and ignored when read.

5.2. Byte ordering

Unless otherwise specified, for all SPDM specifications *byte* ordering of multi-byte numeric fields or multi-byte bit fields is "Little Endian"(that is, the lowest byte offset holds the least significant byte, and higher offsets hold the more significant bytes).

5.3. SPDM data types

The SPDM data types table lists the abbreviations and descriptions for common data types that SPDM message fields and data structure definitions use. These definitions follow *DSP0240 — PLDM Base Specification*.

SPDM data types

Data type	Interpretation
`ver8`	Eight-bit encoding of the SPDM version number. Reserved and unassigned values define the encoding of the version number. For example, `7:4` is a major version number and `3:0` is a minor version number.
`bitfield8`	Byte with eight bit fields. Each bit field can be separately defined.
`bitfield16`	Two-byte word with 16-bit fields. Each bit field can be separately defined.

5.4. Version encoding

The version field represents the version of the specification and contains:

- Two bytes. The *major* and *minor* nibbles.
- One byte. The detailed version.

The major and minor nibbles shall be encoded, as follows:

Version	Matches	Incremented when
Major	Major version field in the `SPDMVersion` field in the SPDM message header.	Protocol modification breaks backward compatibility.
Minor	Minor version field in the `SPDMVersion` field in the SPDM message header.	Protocol modification maintains backward compatibility.

EXAMPLE:

Version 3.7 → `0x37`

Version 1.0 → `0x10`

Version 1.2 → `0x12`

An *endpoint* that supports Version 1.2 can interoperate with an older endpoint that supports Version 1.0, but the available functionality is limited to what is defined in SPDM specification Version 1.0.

An endpoint that supports Version 1.2 and an endpoint that supports Version 3.7 are not interoperable and shall not attempt to communicate beyond `GET_VERSION`.

The detailed version byte resides in the `VERSION` response message payload and is incremented to indicate specification bug fixes.

5.5. Notations

The following notations are used for SPDM specifications:

Notation	Description
`M:N`	In field descriptions, this notation typically represents a range of byte offsets starting from byte M and continuing to and including byte N (M ≤ N). The lowest offset is on the left. The highest offset is on the right.
`[4]`	Square brackets around a number typically indicate a bit offset. Bit offsets are zero-based values. That is, the least significant bit `[LSb]` offset=0.
`[7:5]`	A range of bit offsets. The most significant is on the left, and the least significant is on the right.
`1b`	A lowercase `b` after a number consisting of 0s and 1s indicates that the number is in binary format.
`0x12A`	A leading `0x` indicates that the number is in hexadecimal format.

6. SPDM message exchanges

The messages exchanges defined in this specification are between two endpoints and are performed and exchanged through sending and receiving of SPDM messages defined in SPDM messages. The SPDM message exchanges are defined in a generic fashion that allows the messages to be communicated across different physical mediums and over different transport protocols.

The two endpoints have a role of either a Requester or Responder. All messages are paired as command/response with the Requester initiating all communication and the Responder replying to the communication.

Endpoints may implement both Requester and Responder capabilities. It is possible for a pair of endpoints to be involved with two SPDM message streams between each other with each endpoint having a Requester role and and Responder role. These two streams are mutually exclusive.

The message exchanges defined in this specification include Requesters that:

1. Discover and negotiate the security capabilities of a Responder.
2. Authenticate the identity of a Responder.
3. Retrieve the firmware measurement of a Responder.

These message exchange capabilities are built on top of well-known and established security practices across the computing industry. Brief overview for each of the message exchange capabilities are described in the following sections. Some of the message exchange capabilities are based on the security model defined in USB Authentication Specification Rev 1.0.

6.1. Security capability discovery and negotiation

This specification defines a mechanism for an Requester to discover the security capabilities of a Responder. For example, an endpoint could support multiple cryptographic hash functions that are defined in this specification.

Furthermore, the specification defines a mechanism for THE Requester and Responder to select a common set of cryptographic algorithms to use for all subsequent message exchanges before the Requester initiates another negotiation, if an overlapping set of cryptographic algorithms exist that both endpoints support.

6.2. Identity authentication

In this specification, the authenticity of a Responder is determined by digital signatures using well-established techniques based on public key cryptography. A Responder proves its identity by generating digital signatures using a private key, and the signatures can be cryptographically verified by the Requester using the public key associated with that private key.

At a high-level, the authentication of a Responder's identity involves these processes:

- **Identity provisioning**

  The process followed by device vendors during or after hardware manufacturing. A trusted root *certificate authority (CA)* generates a *root certificate* (*RootCert*) that is provisioned to the *authentication initiator* to allow the authentication initiator to verify the validity of the digital signatures generated by the endpoint during runtime authentication.

  The root CA also indirectly through the *certificate chain* endorses a per-part public/private key pair, where the private key is provisioned to or generated by the endpoint. A device carries a certificate chain, with the root being the RootCert and the leaf being the device certificate (*DeviceCert*), which contains the public key corresponding to the device private key.

- **Runtime authentication**

  The process by which an authentication initiator (Requester) interacts with a Responder in a running system. The authentication initiator can retrieve the certificate chain(s) from the Responder and send a unique challenge to the Responder. The Responder then signs the challenge with the private key. The authentication initiator verifies the signature using the public key of the Responder as well as any intermediate public keys within the certificate chain using the root certificate as the trusted anchor.

6.3. Firmware measurement

Measurement is a term that describes the process of calculating the cryptographic hash value of a piece of firmware/software and tying the cryptographic hash value with the endpoint identity through the use of digital signatures. This allows an authentication initiator to establish that the identity and measurement of the firmware/software running on the endpoint.

7. SPDM messaging protocol

The SPDM messaging protocol defines a request-response messaging model between two endpoints to perform the message exchanges outlined in SPDM message exchanges. Each SPDM request message shall be responded to with an SPDM response message as defined in this specification unless otherwise stated in this specification.

Figure 1 depicts the high-level request-response flow diagram for SPDM. An endpoint that acts as the Requester sends an SPDM request message to another endpoint that acts as the *Responder*, and the Responder returns an SPDM response message to the Requester.

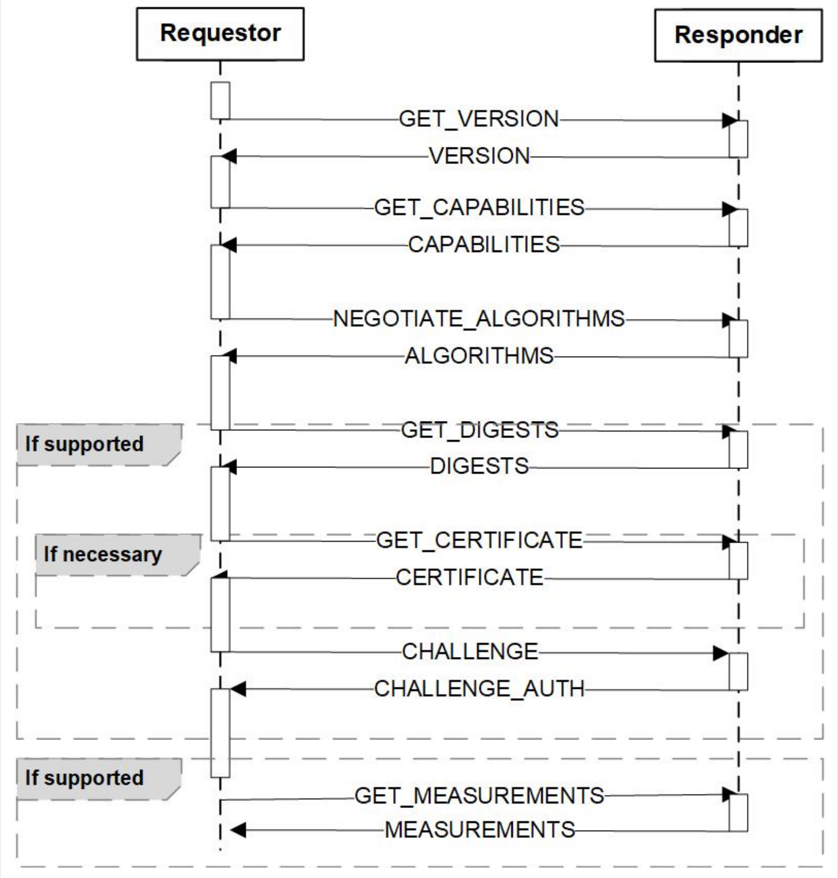

Figure 1 — SPDM messaging protocol flow

All SPDM request-response messages share a common data format, that consists of a four-byte message header and zero or more bytes message payload that is message-dependent. The following clauses describe the common message format and SPDM messages details each of the request and response messages.

The Requester shall issue `GET_VERSION`, `GET_CAPABILTIES`, and `NEGOTIATE_ALGORITHMS` request messages before issuing any other request messages.

7.1. Generic SPDM message format

The Generic SPDM message formats table defines the fields that constitute a generic SPDM message, including the message header and payload. The fields within the SPDM messages are transferred from the lowest offset first.

Generic SPDM message formats

Byte 1								Byte 2								Byte 3								Byte 4							
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
SPDM Major Version				SPDM Minor Version				Request Response Code								Param1								Param2							
SPDM message payload (zero or more bytes)																															

The Generic SPDM message field definitions table defines the fields that are part of a generic SPDM message.

Generic SPDM message field definitions

Field name	Field size (bits)	Description
SPDM Major Version	4	The major version of the SPDM Specification. An endpoint shall not communicate by using an incompatible SPDM version value. See Version encoding.
SPDM Minor Version	4	The minor version of the SPDM Specification. A specification with a given minor version extends a specification with a lower minor version as long as they share the major version. See Version encoding.
Request Response Code	8	The request message code or response code, which the SPDM request codes and SPDM response codes tables enumerate. `0x00` through `0x7F` represent response codes and `0x80` through `0xFF` represent request codes.
Param1	8	The first one-byte parameter. The contents of the parameter is specific to the Request Response Code.
Param2	8	The second one-byte parameter. The contents of the parameter is specific to the Request Response Code.
SPDM message payload	Variable	Zero or more bytes that are specific to the Request Response Code.

7.2. SPDM request codes

The SPDM request codes table defines the SPDM request codes.

All SPDM-compatible implementations shall use the following request codes.

Unsupported request codes shall return an `ERROR` response message with `ErrorCode=UnsupportedRequest`.

SPDM request codes

Request	Code value	Implementation Requirement	Message format
`GET_DIGESTS`	`0x81`	Optional	See the GET_DIGESTS request message table.
`GET_CERTIFICATE`	`0x82`	Optional	See the GET_CERTIFICATE request message table.
`CHALLENGE`	`0x83`	Optional	See the CHALLENGE request message table.
`GET_VERSION`	`0x84`	Required	See the GET_VERSION request message table.
`GET_MEASUREMENTS`	`0xE0`	Optional	See the GET_MEASUREMENTS request message table.
`GET_CAPABILITIES`	`0xE1`	Required	See the GET_CAPABILITIES request message table.
`NEGOTIATE_ALGORITHMS`	`0xE3`	Required	See the NEGOTIATE_ALGORITHMS request message table.
`RESPOND_IF_READY`	`0xFF`	Required	See the RESPOND_IF_READY request message table.

Reserved	`0x80,`
	`0x85-0xDF,`
	`0xE2,`
	`0xE4-0xFE`
		SPDM implementations compatible with this version shall not use the reserved request codes.	

7.3. SPDM response codes

The Request Response Code field in the SPDM response message shall specify the appropriate response code for a request. All SPDM-compatible implementations shall use the following response codes.

On a successful completion of an SPDM operation, the specified response message shall be returned. Upon an unsuccessful completion of an SPDM operation, the `ERROR` response message shall be returned.

The SPDM response codes table defines the response codes for SPDM.

SPDM response codes

Response	Value	Implementation requirement	Message format
`DIGESTS`	`0x01`	Required for endpoints that support `GET_DIGESTS` request message.	See the GET_DIGESTS request message table.
`CERTIFICATE`	`0x02`	Required for endpoints that support `GET_CERTIFICATE` request message.	See the GET_CERTIFICATE request message table.
`CHALLENGE_AUTH`	`0x03`	Required for endpoints that support `CHALLENGE` request message.	See the CHALLENGE request message table.
`VERSION`	`0x04`	Required for all Responders.	See the Successful VERSION response message table.
`MEASUREMENTS`	`0x60`	Required for endpoints that support `GET_MEASUREMENTS` request message.	See the Successful MEASUREMENTS response message table.
`CAPABILITIES`	`0x61`	Required for all endpoints.	See the Successful CAPABILITIES response message table.
`ALGORITHMS`	`0x63`	Required for all endpoints.	See the NEGOTIATE_ALGORITHMS request message table.
`ERROR`	`0x7F`	See the ERROR response message table.	

Reserved	`0x00,`
	`0x05`
	`0x5F,`
	`0x62,`
	`0x64-0x7E`
		SPDM implementations compatible with this version shall not use the reserved response codes.	

8. Concurrent SPDM message processing

This clause describes the specifications and requirements for handling concurrent overlapping SPDM request messages.

If an endpoint can act as both a Responder and Requester, it shall be able to send request messages and responses independently.

8.1. Requirements for Requesters

A Requester shall not have multiple outstanding requests to the same Responder, with the exception of `GET_VERSION` addressed in GET_VERSION request message and VERSION response message. The SPDM Requester shall wait to issue a subsequent request to a Responder until the Requester completes one of the following actions:

- Receives the response from the Responder for the outstanding request.
- Times out waiting for a response.
- Receives an indication that transmission of the request message failed.

A Requester may send simultaneous request messages to different Responders.

8.2. Requirements for Responders

A Responder that is not ready to accept a new request message shall either respond with an `ERROR` response message with `ErrorCode=Busy` or silently discard the request message.

A Responder is not required to process more than one request message at a time. If a Responder is working on a request message from a Requester, the Responder shall be allowed to respond with `ErrorCode=ResponseNotReady`.

If a Responder allows simultaneous communications with multiple Requesters, the Responder is expected to distinguish the Requesters by using mechanisms that are outside the scope of this specification.

8.3. Timing requirements

The Timing specification for SPDM messages table shows the timing specifications for Requesters and Responders.

If the Requester does not receive a response within **T1** or **T2** time accordingly, the Requester may retry a request message. A retry of a request message shall be a complete retransmission of the original SPDM request message.

The Responder shall not retry SPDM response messages. It is understood that the transport protocol(s) may retry failed packages, but that is outside of the SPDM specification.

8.3.0.1. Timing measurements

A Requester shall measure timing parameters, applicable to it, from the end of a successful transmission of an SPDM request to the beginning of the reception of the corresponding SPDM response. A Responder shall measure timing parameters, applicable to it, from the end of the reception of the SPDM request to the beginning of transmission of the response.

8.3.1. Timing specification table

In the Timing specification for SPDM messages table, the **Ownership** column shows whether the timing parameter applies to the Responder or Requester.

Timing specification for SPDM messages

Timing parameter	Ownership	Value	Description
RTT	Requester	See the **Description**.	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	100ms	This shall be the maximum amount of time the Responder has to provide a response that does not require cryptographic processing, such as `GET_CAPABILITIES`, `GET_VERSION`, or `NEGOTIATE_ALGORITHMS`.
T1	Requester	RTT+ST1	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	$2^{CTExponent}$	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	This shall be the minimum amount of time the Requester shall wait before issuing a retry for requests that require cryptographic processing. For more details, see `CT`.
RDT	Responder	$2^{RDTExponent}$	Recommended delay, in microseconds. When the Responder cannot complete cryptographic processing response within the `CT` time, it shall provide this minimum delay parameter as part of the `ERROR` response. See the ResponseNotReady extended error data table for the `RDTExponent` value. For details, see `ErrorCode=ResponseNotReady`.
WT	Requester	RDT	Amount of time the Requester should wait before issuing the `RESPOND_IF_READY` request. The Requester shall measure this time parameter from the receipt of the `ERROR` response to the transmission of `RESPOND_IF_READY` request. The Requester may take into account the transmission time of the `ERROR` from the Responder to Requester when calculating `WT`. For details, see `RDT`.
WT_{Max}	Requester	(RDT*RDTM)−RTT	Maximum wait time the Requester has to to issue the `RESPOND_IF_READY` request unless the Requester issued a successful `RESPOND_IF_READY` request earlier. After this time, the Responder can drop the response. The Requester shall account for the transmission time of the `ERROR` response from the Responder to Requester when calculating WT_{Max}. The ResponseNotReady extended error data table shows the value of `RDTM`. The Responder should ensure WT_{Max} does not result less than `WT` in determination of `RDTM`. For details, see `ErrorCode=ResponseNotReady`.

9. SPDM messages

SPDM messages can be divided into the following categories, supporting different aspects of security exchanges between a Requester and Responder:

1. Capability discovery and negotiation.
2. Hardware identity authentication.
3. Firmware measurement.

9.1. Capability discovery and negotiation

All endpoints shall support `GET_VERSION`, `GET_CAPABILITIES` and `NEGOTIATE_ALGORITHMS` both as a Requester and a Responder. Figure 2 shows the high-level request-response flow and sequence for the capability discovery and negotiation.

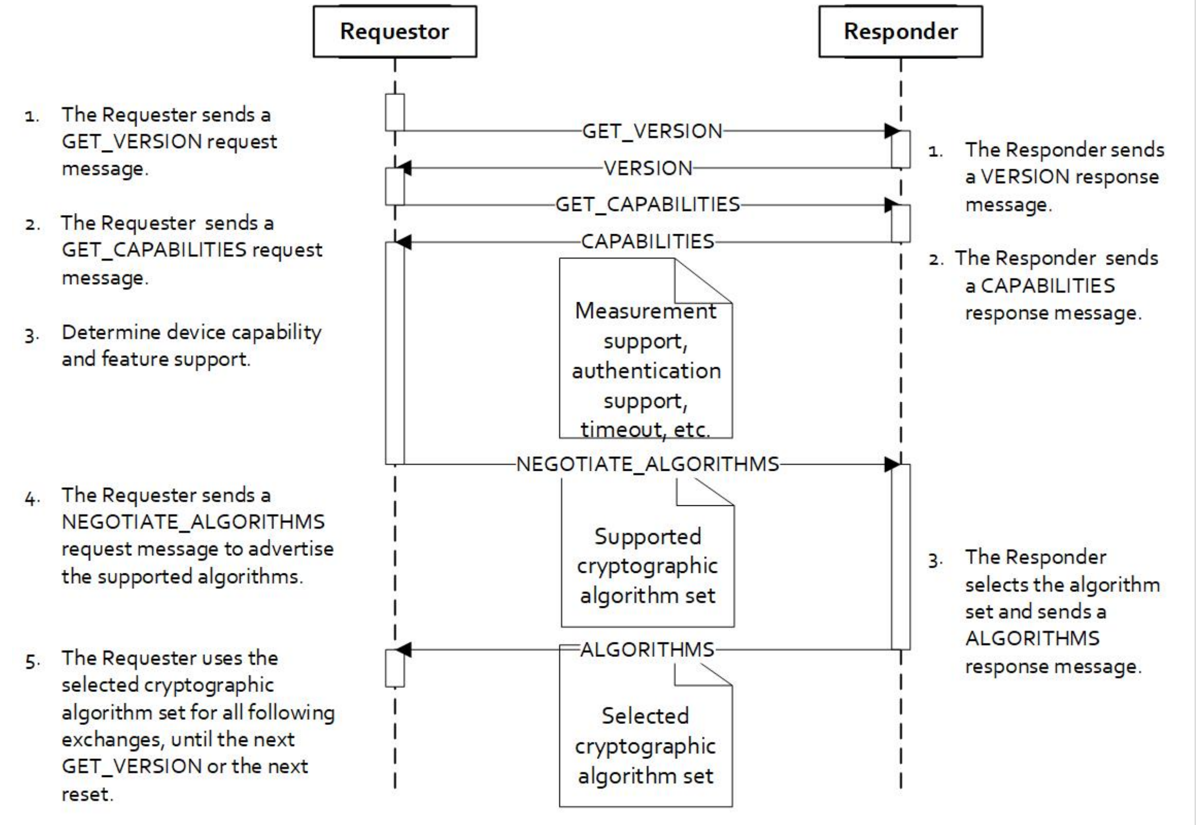
Requestor
Responder
1. The Requester sends a GET_VERSION request message.
GET_VERSION
1. The Responder sends a VERSION response message.
VERSION
GET_CAPABILITIES
2. The Requester sends a GET_CAPABILITIES request message.
CAPABILITIES
2. The Responder sends a CAPABILITIES response message.
3. Determine device capability and feature support.
Measurement support, authentication support, timeout, etc.
NEGOTIATE_ALGORITHMS
4. The Requester sends a NEGOTIATE_ALGORITHMS request message to advertise the supported algorithms.
Supported cryptographic algorithm set
3. The Responder selects the algorithm set and sends a ALGORITHMS response message.
ALGORITHMS
5. The Requester uses the selected cryptographic algorithm set for all following exchanges, until the next GET_VERSION or the next reset.
Selected cryptographic algorithm set

Figure 2 — Capability discovery and negotiation flow

9.1.1. GET_VERSION request message and VERSION response message

This request message shall retrieve an endpoint's security version. The GET_VERSION request message table shows the `GET_VERSION` request message format and the Successful VERSION response message table shows the `VERSION` response message format.

In all future SPDM versions, the `GET_VERSION` and `VERSION` response messages will be backward compatible with all previous versions.

If the Requester supports multiple SPDM Major Versions, the Requester shall begin the discovery process by sending a `GET_VERSION` request message with major version 0x1. All Responders must always support `GET_VERSION` request message with major version 0x1 and provide a `VERSION` response containing all supported versions as described in The GET_VERSION request message table.

The Requester shall consult the `VERSION` response to select a common (typically highest) version supported. The Requester shall use the selected version in all future communication of other requests. A Requester shall not issue other requests until it has received a successful `VERSION` response and has identified a common version supported by both sides. A Responder shall not respond to `GET_VERSION` request message with `ErrorCode=ResponseNotReady`.

A Requester may issue `GET_VERSION` request message at any time to a Responder, which is as an exception to the rules in Requirements for Requesters for the case where a Requester must restart the protocol because of internal error or reset. After receiving a `GET_VERSION` request the Responder shall cancel all previous requests from the same Requester. '

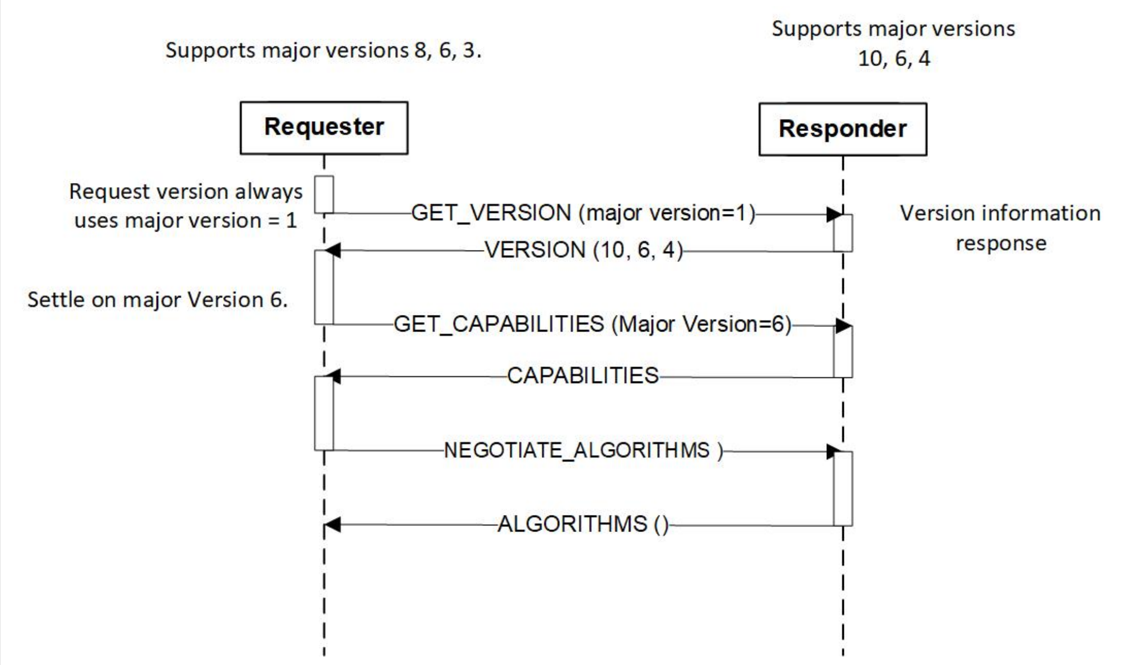
Supports major versions 8, 6, 3.
Supports major versions 10, 6, 4
Requester
Responder
Request version always uses major version = 1
GET_VERSION (major version=1)
Version information response
VERSION (10, 6, 4)
Settle on major Version 6.
GET_CAPABILITIES (Major Version=6)
CAPABILITIES
NEGOTIATE_ALGORITHMS)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

Figure 3 — Discovering common major version

GET_VERSION request message

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0x84=GET_VERSION`
2	`Param1`	1	Reserved
3	`Param2`	1	Reserved

Successful VERSION response message

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. See the GET_VERSION request message table.

VersionNumberEntry 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.
[11:8]	`MinorVersion`	Version of the specification with changes that are compatible with functions in earlier minor versions of this major version specification.
[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.
[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.

9.1.2. GET_CAPABILITIES request message and CAPABILITIES response message

This request message shall retrieve an endpoint's security capabilities.

The GET_CAPABILITIES request message table shows the `GET_CAPABILITIES` request message format.

The Successful CAPABILITIES response message table shows the `CAPABILITIES` response message format.

The Flag fields definitions table shows the flag fields definitions.

A Responder shall not respond to `GET_CAPABILITIES` request message with ErrorCode=`ResponseNotReady`.

GET_CAPABILITIES request message

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0x61=CAPABILITIES`
2	`Param1`	1	Reserved
3	`Param2`	1	Reserved

Successful CAPABILITIES response message

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	Shall be the exponent of base 2, which is used to calculate `CT` as described in the Timing specification for SPDM messages table. 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 the SPDM data types table.
12	Reserved	2	Reserved

Flag fields definitions

Byte	Bit	Field	Value
0	`0`	Reserved	Reserved
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 support `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

9.1.3. NEGOTIATE_ALGORITHMS request message and ALGORITHMS response message

This request message shall negotiate cryptographic algorithms. A Requester shall not issue an `NEGOTIATE_ALGORITHMS` request message until it receives a successful `CAPABILITIES` response message.

A Requester shall not issue any other SPDM requests with the exception of `GET_CAPABILITIES` until it receives a successful `ALGORITHMS` response message with exactly one asymmetric algorithm and exactly one hashing algorithm.

The NEGOTIATE_ALGORITHMS request message table shows the `NEGOTIATE_ALGORITHMS` request message format.

The Successful ALGORITHMS response message table shows the `ALGORITHMS` response message format.

NEGOTIATE_ALGORITHMS request message

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

Offset	Field	Size (bytes)	Value
4	`Length`	2	Length of the entire request message packet, in bytes. Length shall be less than 64 bytes.
6	`MeasurementSpecification`	1	Bit mask. The values for this field shall be those defined in the `MeasurementSpecification` field of the GET_MEASUREMENTS request message and MEASUREMENTS response message. The Requester may set more than bit to indicate multiple measurement specification support.
7	Reserved	1	Reserved
8	`BaseAsymAlgo`	4	Bit mask that lists 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
12	`BaseHashAlgo`	4	Bit mask listing Requester-supported SPDM-enumerated cryptographic hashing algorithms . - Bit 0. TPM_ALG_SHA_256 - Bit 1. TPM_ALG_SHA_384 - Bit 2. TPM_ALG_SHA_512 - Bit 3. TPM_ALG_SHA3_256 - Bit 4. TPM_ALG_SHA3_384 - Bit 5. TPM_ALG_SHA3_512
16	Reserved	8	Reserved
24	`ExtAsymCount`	1	Number of Requester-supported extended asymmetric key signature algorithms (=A). A+E shall be less than or equal to 8.
25	`ExtHashCount`	1	Number of Requester-supported extended hashing algorithms (=E). A+E shall be less than or equal to 8.
26	Reserved	2	Reserved for future use.
28	`ExtAsym`	4*A	List of Requester-supported extended asymmetric key signature algorithms. - First byte in each entry is the enumeration for the encoding for `ExtAsym`: ○ 0 – DMTF ○ 1 – TCG - Second byte is reserved. - Other two bytes represent the algorithm encoding. At this time, the DMTF namespace has no algorithms defined. TCG algorithms are enumerated in TCG Algorithm Registry.
28+4 *A	`ExtHash`	4*E	List of extended hashing algorithms supported by Requester. - First byte in each entry is enumeration for the encoding for `ExtHash`: ○ 0 – DMTF ○ 1 – TCG - Second byte is reserved. - Other two bytes represent the algorithm encoding. At this time, the DMTF namespace has no algorithms defined. TCG algorithms are enumerated in the TCG Algorithm Registry.
28+4*A+4*E	Reserved	Length−28−4*A−4*E	Reserved for future expansion. Consult the `Length` field (offset 4) to determine the number of bytes in the request message.

Successful ALGORITHMS response message

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 packet, 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 defined in the `MeasurementSpecification` field of the Successful CERTIFICATE response message.
7	`MeasurementHashAlgo`	1	Bit mask listing SPDM-enumerated hashing algorithm for measurements. M represents the length of the measurement hash field in measurement block structure. See the CHALLENGE request message table. 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. TPM_ALG_SHA_256, M=32 - Bit 1. TPM_ALG_SHA_384, M=48 - Bit 2. TPM_ALG_SHA_512, M=64 - Bit 3. TPM_ALG_SHA3_256, M=32 - Bit 4. TPM_ALG_SHA3_384, M=48

Offset	Field	Size (bytes)	Value
			- Bit 5. 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`.
8	`BaseAsymSel`	4	Bit mask listing SPDM-enumerated asymmetric key signature algorithm selected. A Responder that returns `AUTH_CAP=0` and `MEAS_CAP=0` shall set this field 0. Other Responders shall set no more than one bit.
12	`BaseHashSel`	4	Bit mask listing SPDM-enumerated hashing algorithm selected. A Responder that returns `AUTH_CAP=0` and `MEAS_CAP=0` shall set this field to `0`. Other Responders shall set no more than one bit.
16	Reserved	8	Reserved.
24	`ExtAsymSelCount`	1	Number of extended asymmetric key signature algorithms selected. Shall be either `0` or `1`. (=A) A Requester that returns `AUTH_CAP=0` and `MEAS_CAP=0` shall set this field to `0`.
25	`ExtHashSelCount`	1	Number of extended hashing algorithms selected. Shall be either `0` or `1`. (=H) A Requester that returns `AUTH_CAP=0` and `MEAS_CAP=0` shall set this field to `0`.
26	Reserved	2	Reserved
28	`ExtAsymSel`	4*A	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 during all subsequent applicable response messages to the Requester until the next `ALGORITHMS` response message. - First byte is enumeration for the encoding for `ExtAsymSel`: - 0 - DMTF - 1 - TCG - Second byte is reserved. - The other two bytes represent the algorithm encoding. At this time, the DMTF namespace has no algorithms defined. TCG algorithms are enumerated in TCG Algorithm Registry.
28+4*A	`ExtHashSel`	4*H	Selected extended hashing algorithm. The Responder shall use this hashing algorithm during all subsequent response messages to the Requester until the next `ALGORITHMS` response message. The Requester shall use this hashing algorithm during all subsequent applicable request messages to the Responder until the next `ALGORITHMS` response message. The length of the nonce and salt fields exchanged during subsequent applicable request messages and response messages shall match the length of the selected hash, until the next ALGORITHM response message. - First byte is enumeration for the encoding for `ExtHashSel`: - 0 - DMTF - 1 - TCG - Second byte is reserved. - Other two bytes represent the algorithm encoding. At this time, the DMTF namespace has no algorithms defined. TCG algorithms are enumerated in TCG Algorithm Registry.
28+4*A+4*H	Reserved	Length —28 —	Reserved for 4*A through 4*H future expansion. Consult the length field (offset 4) to determine the total number of bytes in the response message.

This clause illustrates how two endpoints negotiate base hashing algorithm.

In Figure 4, endpoint A issues `NEGOTIATE_ALGORITHMS` request message and endpoint B selects an algorithm of which both endpoints are capable.

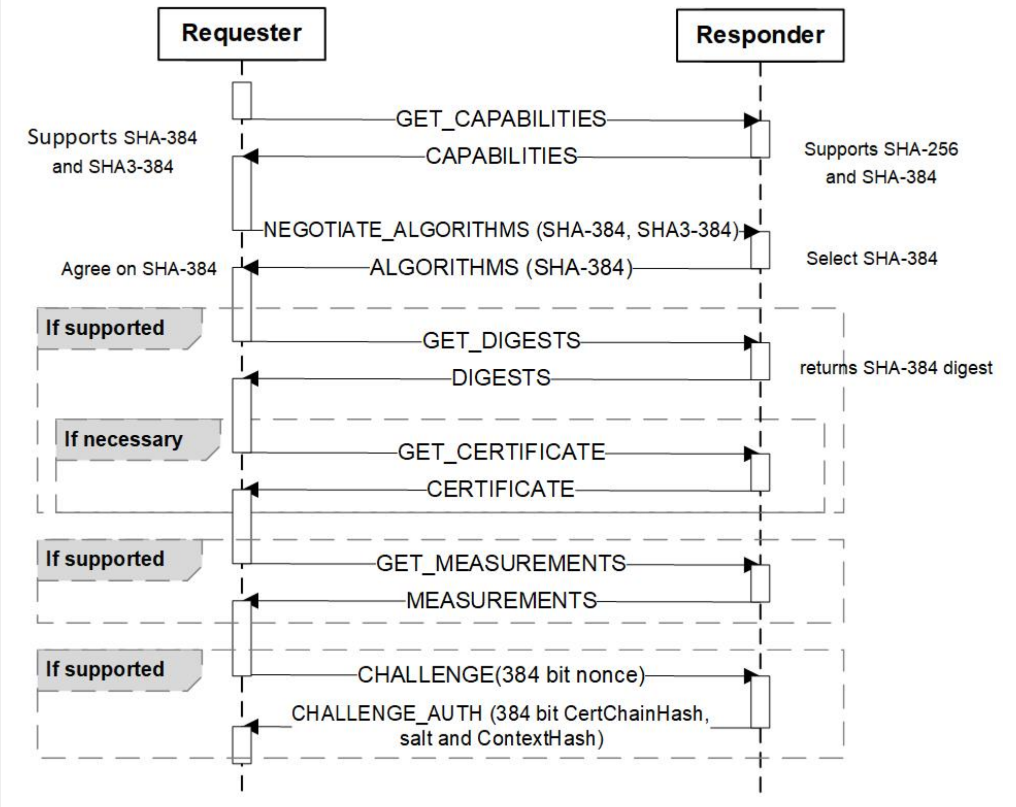

Figure 4 — Hashing algorithm selection: Example 1

SPDM protocol accounts for the possibility that both endpoints may issue `NEGOTIATE_ALGORITHMS` request message independent of each other. In this case, endpoint A Requester and endpoint B Responder communication pair may select a different algorithm compared to the endpoint B Requester and endpoint A Responder communication pair.

9.2. Responder identity authentication

This clause describes request messages and response messages associated with the Responder's identity authentication operations. All request messages in this clause shall be supported by a Responder that returns `AUTH_CAP=1` in the `CAPABILITIES` response message.

Figure 5 shows the high-level request-response message flow and sequence for Responder's identity authentication for *certificate* retrieval.

Figure 6 shows the high-level request-response message flow and sequence for Responder's authentication for runtime challenge-response.

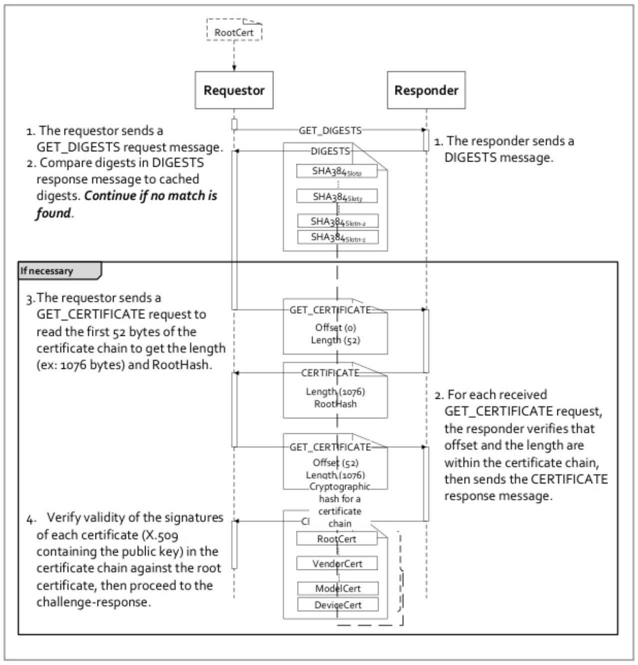
RootCert
Requestor
Responder
1. The requestor sends a GET_DIGESTS request message.
2. Compare digests in DIGESTS response message to cached digests. Continue if no match is found.
GET_DIGESTS
1. The responder sends a DIGESTS message.
DIGESTS
SHA384Slot0
SHA384Slot1
SHA384Slotn-2
SHA384Slotn-1
If necessary
3. The requestor sends a GET_CERTIFICATE request to read the first 52 bytes of the certificate chain to get the length (ex: 1076 bytes) and RootHash.
GET_CERTIFICATE
Offset (0)
Length (52)
CERTIFICATE
Length (1076)
RootHash
2. For each received GET_CERTIFICATE request, the responder verifies that offset and the length are within the certificate chain, then sends the CERTIFICATE response message.
GET_CERTIFICATE
Offset (52)
Length (1076)
Cryptographic hash for a certificate chain
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.
RootCert
VendorCert
ModelCert
DeviceCert

Figure 5 — Responder authentication: example certificate retrieval flow

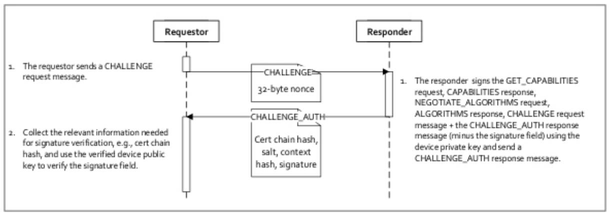
Requestor
Responder
1. The requestor sends a CHALLENGE request message.
CHALLENGE
32-byte nonce
1. The responder signs the GET_CAPABILITIES request, CAPABILITIES response, NEGOTIATE_ALGORITHMS request, ALGORITHMS response, CHALLENGE request message + the CHALLENGE_AUTH response message (minus the signature field) using the device private key and send a CHALLENGE_AUTH response message.
CHALLENGE_AUTH
2. Collect the relevant information needed for signature verification, e.g., cert chain hash, and use the verified device public key to verify the signature field.
Cert chain hash, salt, context hash, signature

Figure 6 — Responder authentication: runtime challenge-response flow

The `GET_DIGESTS` request message and `DIGESTS` response message may optimize the amount of data required to be transferred from the Responder to the Requester, due to the potentially large size of a certificate chain. The cryptographic hash values of each of the certificate chains stored on an endpoint is returned with the `DIGESTS` response message, such that the Requester can cache the previously retrieved certificate chain hash values to detect any change to the certificate chains stored on the device before issuing the `GET_CERTIFICATE` request message.

For the runtime challenge-response flow, the signature field in the `CHALLENGE_AUTH` response message payload shall be signed using the device private key over the `GET_VERSION` request, `VERSION` response, `GET_CAPABILITIES` request, `CAPABILITIES` response, `NEGOTIATE_ALGORITHMS` request, `ALGORITHMS` response, `CHALLENGE` request message and the `CHALLENGE_AUTH` response message except for the signature field, to ensure cryptographic binding between a specific request message from a specific Requester and a specific response message from a specific Responder. Inclusion of `GET_VERSION` request, `VERSION` response, `GET_CAPABILITIES` request, `CAPABILITIES` response, `NEGOTIATE_ALGORITHMS` request and `ALGORITHMS` response allows the Responder to detect the presence of an active adversary attempting to downgrade cryptographic algorithms or SPDM Major Versions. Furthermore, a nonce generated by the Requester protects the challenge-response from replay attacks, whereas a nonce generated by the Responder prevents the Responder from signing over arbitrary data dictated by the Requester. The signature computation is restarted with the latest GET_VERSION request received.

9.2.1. Certificates and certificate chains

Each Responder that supports identity authentication shall carry at least one certificate chain. A certificate chain contains an ordered list of certificates, presented as the binary (byte) concatenation of the fields shown in the CHALLENGE request message table. Each certificate shall be in ASN.1 DER-encoded X509v3 format. The ASN.1 DER encoding of each individual certificate can be analyzed to determine its length. The minimum number of certificates within a chain shall be one, in which case the single certificate is the device-specific certificate. The Responder shall contain a single public-private key pair per supported algorithm for its hardware identity, regardless of how many certificate chains are stored on the device. The Responder selects a single asymmetric key signature algorithm per Requester.

Certificate chains are stored in locations called slots. Each slot shall either be empty or contain one complete certificate chain. A product shall not contain more than 8 slots. Slot 0 is populated by default. Additional slots may be populated through the supply chain such as by a platform integrator or by an end user such as the IT administrator. A slot mask is used to identify the certificate chains from the 8 slots.

In this document, `H` refers to the output size (bytes) of the hash algorithm agreed upon in `NEGOTIATE_ALGORITHMS`.

Certificate chain format

Offset	Field	Size	Description
0	`Length`	2	Total length of the certificate chain, in bytes, including all fields in this table. This field is little endian.
2	Reserved	2	Reserved.
4	`RootHash`	H	Digest of the Root Certificate. Note that Root Certificate is ASN.1 DER-encoded for this digest. This field is big endian.
4+H	`Certificates`	Length−(4+H)	One or more ASN.1 DER-encoded X509v3 certificates where the first certificate is signed by the Root Certificate or is the Root Certificate itself and each subsequent certificate is signed by the preceding certificate. The last certificate is the *Leaf Certificate*. This field is big endian.

9.2.2. GET_DIGESTS request message and DIGESTS response message

This request message shall retrieve the certificate chain digests.

The GET_DIGESTS request message table shows the `GET_DIGESTS` request message format. The digests in the GET_DIGESTS request message table are in big endian.

The Successful DIGESTS response message table shows the `DIGESTS` response message format.

GET_DIGESTS request message

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0x81=GET_DIGESTS`
2	`Param1`	1	Reserved
3	`Param2`	1	Reserved

Successful DIGESTS response message

Offset	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.) Number of digests returned shall be equal to 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 (n^{th}) certificate chain.

9.2.3. GET_CERTIFICATE request message and CERTIFICATE response message

This request message shall retrieve the certificate chains.

The GET_CERTIFICATE request message table shows the `GET_CERTIFICATE` request message format.

The Successful CERTIFICATE response message table shows the `CERTIFICATE` response message format.

The Responder should, at a minimum, save the public key of the leaf certificate and associate with each of the digests returned by `DIGESTS` message response. The Requester sends one or more `GET_CERTIFICATE` requests to retrieve the Responder's certificate chain.

GET_CERTIFICATE request message

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 the `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. 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 response message

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.
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`	PortionLength	Requested contents of target certificate chain, formatted in DER. This field is big endian.

Figure 7 shows the high-level request-response message flow for Responder response when it cannot return the entire data requested by the Requester in the first response.

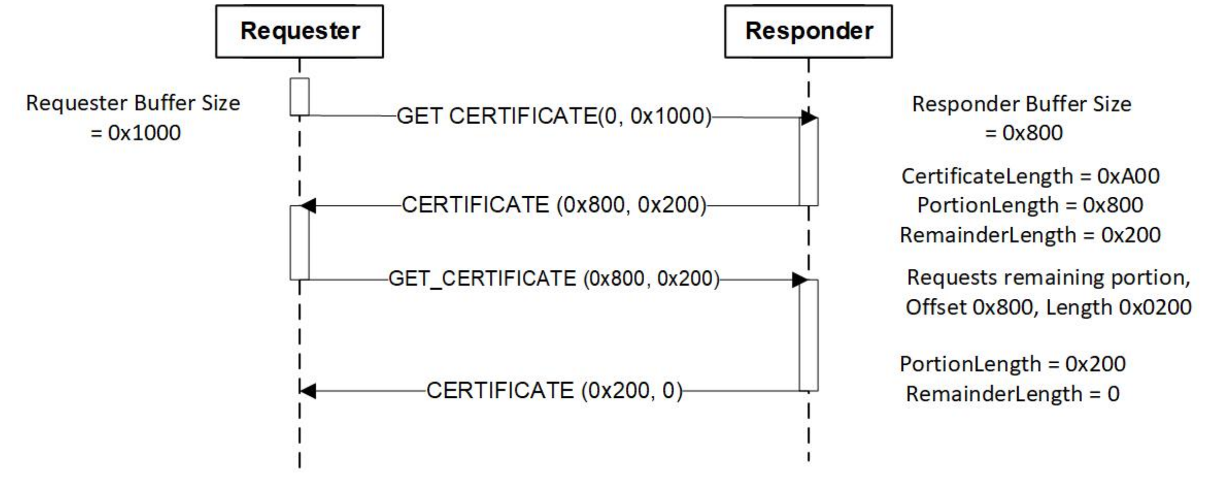
Requester
Responder
Requester Buffer Size = 0x1000
GET CERTIFICATE(0, 0x1000)
Responder Buffer Size = 0x800
CERTIFICATE (0x800, 0x200)
CertificateLength = 0xA00
PortionLength = 0x800
RemainderLength = 0x200
GET_CERTIFICATE (0x800, 0x200)
Requests remaining portion, Offset 0x800, Length 0x0200
CERTIFICATE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

Figure 7 — Responder unable to return full length data flow

9.2.4. Leaf certificate

The SPDM endpoints for authentication must be provisioned with DER-encoded X.509 v3 format certificates. The leaf certificate must be signed by a trusted CA and provisioned to the device. For endpoint devices to verify the certificate, the following required fields must be present. In addition, to provide device information, use the Subject Alternative Name certificate extension `OtherName` field.

9.2.4.1. Required fields

Field	Description
Version	Version of encoded certificate. Shall be present and shall be `3` or `2`.
Serial Number	CA-assigned serial number. Shall be present with a positive integer value.
Signature Algorithm	Signature algorithm used by CA. Shall be present.
Issuer	CA-distinguished name. Shall be specified.
Subject Name	Subject name. Shall be present. Shall represent the distinguished name associated with the leaf certificate.
Validity	Certificate may include this attribute. If validity attribute is present, the value for `notBefore` field should be assigned the generalized `19700101000000Z` time value and `notAfter` field should be assigned the generalized `99991231235959Z` time value.
Subject Public Key Info	Device public key and the algorithm. Shall be present.
Extended Key Usage	Extended key usage. Shall be present and key usage bit for digital signature shall be set.

9.2.4.2. Optional fields

Field	Description
Basic Constraints	If present, the CA value shall be `FALSE`.
Subject Alternative Name `OtherName`	In some cases, it might be desirable to provide device specific information as part of the device certificate. DMTF chose the `OtherName` field to be used with a specific format to represent the device information. The use of the `OtherName` field also provides flexibility that enables other alliances to provide information about device specific information as part of the device certificate.

9.2.4.3. Definition of otherName using the DMTF OID

```
DMTFOtherName :== SEQUENCE {
    type-id  [0] id-DMTF-device-info
    value    [1] ub-DMTF-device-info
}
-- OID for DMTF device info --
id-DMTF-device-info  OBJECT IDENTIFIER ::== { 1 3 6 1 4 1 412 274 1 }

-- All printable characters except ":" --
DMTF-device-string   PrintableString   ::== (ALL EXCEPT ":")

-- Device Manufacturer --
DMTF-manufacturer                      ::== DMTF-device-string

-- Device Product --
DMTF-product                           ::== DMTF-device-string

-- Device Serial Number --
DMTF-serialNumber                      ::== DMTF-device-string

-- Device information string  --
ub-DMTF-device-info  UTF8String        ::== (DMTF-manufacturer":"DMTF-product":"DMTF-serialNumber)
```

Annex B shows an example leaf certificate.

9.2.5. CHALLENGE request message and CHALLENGE_AUTH response message

This request message shall authenticate an endpoint through the challenge-response protocol.

The CHALLENGE request message table shows the `CHALLENGE` request message format.

The Successful CHALLENGE_AUTH response message table shows the `CHALLENGE_AUTH` response message format.

CHALLENGE request message

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	Reserved
4	`Nonce`	32	The Requester should choose a random value.

Successful CHALLENGE_AUTH response message

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.
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	`MinSPDMVersion`	1	Minimum SPDM version supported by this endpoint.
5	`MaxSPDMVersion`	1	Maximum SPDM version supported by this endpoint.
6	`CAPABILITIES`	1	Set to `01h` for this specification. All other values reserved.
7	Reserved	1	Reserved
8	`CertChainHash`	H	Hash of the certificate chain used for authentication. This field is big endian.
8+H	`Nonce`	32	Responder-selected random value.
8+2H	`MeasurementSummaryHash`	H	Hash computed using Concatenation(Measurement 1, Measurement 2, ..., Measurement N) of all measurements supported by the device. The measurements include all supported indices by the device.
8+3H	`OpaqueLength`	2	Size of the `OpaqueData` field. The value shall not be greater than 1024 bytes.
10+3H	`OpaqueData`	`OpaqueLength`	Free-form field, if present. The Responder may include Responder-specific information and/or information defined by its transport.
10+3H+`OpaqueLength`	`Signature`	S	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 Signature generation and Signature verification clauses, respectively.

9.2.6. Signature generation

1. The Responder shall construct M1:

   ```
   M1=Concatenate(GET_VERSION, VERSION, GET_CAPABILITIES, CAPABILITIES, NEGOTIATE_ALGORITHMS, ALGORITHMS, CHALLENGE, CHALLENGE_AUTH)
   ```

   where:

   | Value | Description | |:------|:------------| | `Concatenate ( )` | Standard concatenation function. | | `GET_VERSION` | Entire contents of the last successful `GET_VERSION` request message processed by the Responder. | | `VERSION` | Entire contents of the associated response message sent by the Responder. Constructing M1 may require that the Responder preserve the contents of these earlier messages. | | `GET_CAPABILITIES` | Entire contents of the last successful `GET_CAPABILITIES` request message processed by the Responder. | | `CAPABILITIES` | Entire contents of the associated response message sent by the Responder. Constructing M1 may require that the Responder preserve the contents of these earlier messages. | | `NEGOTIATE_ALGORITHMS` | Entire contents of the last successful `NEGOTIATE_ALGORITHMS` request message processed by the Responder. | | `ALGORITHMS` | Entire contents of the associated response message sent by the Responder. Constructing M1 may require that the Responder preserve the contents of these earlier messages. | | `CHALLENGE` | Entire contents of the `CHALLENGE` request message under consideration, as seen by the Responder. | `CHALLENGE_AUTH` | Entire `CHALLENGE_AUTH` response message without the signature bytes, as sent by the Responder. |

2. The Responder shall generate:

   ```
   Signature=Sign(SK, Hash1(M1))
   ```

   where:

   - `Sign` is the asymmetric signing algorithm the Responder selected via the last `ALGORITHMS` response message sent by the Responder. See `BaseAsymSel` or `ExtAsymSel` fields in the CHALLENGE request message table.
   - `Hash1` is the hashing algorithm the Responder selected via the last `ALGORITHMS` response message sent by the Responder. See `BaseHashSel` or `ExtHashSel` fields in the CHALLENGE request message table.
   - `SK` is the private Key associated with the Responder's leaf certificate in slot=Param1 of `CHALLENGE` request message.

9.2.7. Signature verification

1. The Requester shall create M2 as:

```
M2=Concatenate (GET_VERSION, VERSION, GET_CAPABILITIES, CAPABILITIES,
`NEGOTIATE_ALGORITHMS`, ALGORITHMS, CHALLENGE, CHALLENGE_AUTH)
```

where

- `Concatenate( )` is the standard concatenation function.
- `GET_VERSION` is the entire contents of the last successful `GET_VERSION` request message processed by the Responder. `VERSION` is the entire contents of the associated response message sent by the Responder. Constructing M1 may require that the Responder preserve the contents of these prior messages.
- `GET_CAPABILITIES` is the entire contents of the last successful `GET_CAPABILITIES` request message sent by the Requester. `CAPABILITIES` is the entire contents of the associated response message received by the Requester. Constructing M2 may require that the Requester preserve the contents of these previous messages.
- `NEGOTIATE_ALGORITHMS` is the entire contents of the last successful `NEGOTIATE_ALGORITHMS` request message sent by the Requester. `ALGORITHMS` is the entire contents of the associated response message received by the Requester. Constructing M2 may require that the Requester preserve the contents of these previous messages.
- `CHALLENGE` is the entire contents of the `CHALLENGE` request message under consideration as sent by the Requester. `CHALLENGE_AUTH` is the entire `CHALLENGE_AUTH` response message without the signature field, as received by the Requester.

  Modifications to the previous request messages or the corresponding response messages by an active person-in-the-middle adversary or media error result in `M2!=M1` and lead to verification failure.

2. The Requester shall perform:

```
Verify(PK, Hash2(M2), Signature)
```

where:

- `PK` is the public key associated with the leaf certificate of the Responder with `slot=Param1` of the `CHALLENGE` request message.
- `Verify` is the asymmetric verification algorithm the Responder selected through the last `ALGORITHMS` response message as received by the Requester. See the `BaseAsymSel` or `ExtAsymSel` field in the CHALLENGE request message table.
- `Hash2` is the hashing algorithm the Responder selected via the last `ALGORITHMS` response message sent as received by the Requester. See the `BaseHashSel` or `ExtHashSel` field in the CHALLENGE request message table.

9.3. Firmware measurement

This clause describes request messages and response messages associated with endpoint firmware measurement. All request messages in this clause shall be supported by an endpoint that returns `MEAS_CAP=1` in `CAPABILITIES` response.

Figure 8 shows the high-level request-response flow and sequence for endpoint firmware measurement. If `MEAS_FRESH_CAP` bit in the `CAPABILITIES` response message returns 0, and the Requester requires fresh measurements, the Responder must be reset before `GET_MEASUREMENTS` is reset. The mechanisms employed for resetting the Responder are outside the scope of this specification.

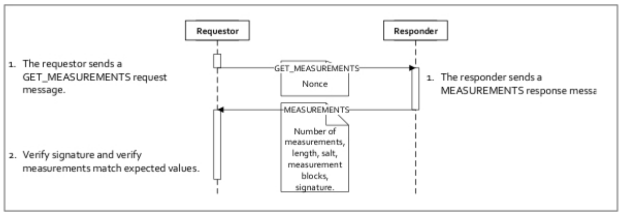
Requestor
Responder
1. The requestor sends a GET_MEASUREMENTS request message.
GET_MEASUREMENTS
Nonce
1. The responder sends a MEASUREMENTS response messa
MEASUREMENTS
Number of measurements, length, salt, measurement blocks, signature.
2. Verify signature and verify measurements match expected values.

Figure 8 — Firmware measurement retrieval flow

9.3.1. GET_MEASUREMENTS request message and MEASUREMENTS response message

This request message shall retrieve firmware measurements.

The GET_MEASUREMENTS request message table shows the `GET_MEASUREMENTS` request message format.

The GET_MEASUREMENTS request attributes table shows the `GET_MEASUREMENTS` request message attributes.

The Successful MEASUREMENTS response message table shows the `MEASUREMENTS` response message format.

GET_MEASUREMENTS request message

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0xE0=GET_MEASUREMENTS`
2	`Param1`	1	Request attributes. See the GET_MEASUREMENTS request attributes table.
3	`Param2`	1	Measurement operation. • A `0x0` value shall request the total number of measurements. • A `0xFF` value returns all measurements. • A value from `0x1` to `0xFE` shall request the measurement at that value.
4	`Nonce`	32	The Requester should choose a random value.

GET_MEASUREMENTS request attributes

Bit(s)	Value	Description

Bit(s)	Value	Description
0	1	If the Responder can generate a signature as indicated in `CAPABILITIES` message, this bit's value shall indicate that the Responder is 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 that can generate signatures, 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 may be absent in the request.
[7:1]	Reserved	Reserved

Successful MEASUREMENTS response message

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0x60=MEASUREMENTS`
2	`Param1`	1	When 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 Number of measurement blocks in **MeasurementRecord**. If the requested measurement operation is `0`, this field shall be `0`.
8	`MeasurementRecord`	L=N*(M+4)	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.
8+2H	`OpaqueLength`	2	Size of the `OpaqueData` field. The value shall not be greater than 1024 bytes.
10+2H	`OpaqueData`	`OpaqueLength`	Free-form field, if present. The Responder may include Responder-specific information and/or information defined by its transport.
10+L+2H+`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.

9.3.2. Measurement block

Each Measurement block defined in the `MEASUREMENTS` response message shall contain a four-byte descriptor (offsets 0-3), followed by the Measurement Data corresponding to a particular *Measurement Index* and *Measurement Type*. The blocks are ordered by `Index`.

The Measurement block format table shows the format for a measurement block:

Measurement block format

Offset	Field	Size (bytes)	Value
0	`Index`	1	For `MeasurementType`=0-3, Index represents the firmware stage and incrementing of index represents bootstrapping of firmware stages. For example, index 0 firmware measures index 1 firmware, and so on.
1	`MeasurementType`	1	• [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
2	`MeasurementSpecification`	1	A bit mask. The value shall indicate the measurement specification that the requested measurement follows and shall match the selected measurement specification in the `Algorithms` message. See the Successful ALGORITHMS response message table. Only one bit shall be set in the Measurement Block. • Bit 0=DMTF All other bits are reserved.
3	`MeasurementSize`	1	Size of measurement, in bytes. When `MeasurementType[7]=0b`: Hash, the `MeasurementSize` shall be derived from the measurement hash algorithm returned in the `ALGORITHM` response message.
4	`Measurement`	The measurement size	`MeasurementSize` bytes of cryptographic hash or raw bit stream, as indicated in `MeasurementType[7]`.

9.3.3. Signature generation

Symbols that end with `1` represent the Responder-observed messages.

1. The Responder shall construct L1:

```
L1=Concatenate(GET_MEASUREMENTS_REQUEST1, MEASUREMENTS_RESPONSE_WITHOUT_SIGNATURE1)
```

   where:

   - `Concatenate ( )` is the standard concatenation function.
   - `GET_MEASUREMENTS_REQUEST1` is the entire `MEASUREMENTS` request message under consideration, as seen by the Responder.

    - `MEASUREMENTS_RESPONSE_WITHOUT_SIGNATURE1` is the entire `MEASUREMENTS` response message without the signature bytes, as sent by the Responder.

2. The Responder shall generate:

```
Signature=Sign(SK, Hash1(L1))
```

   where:

    - `Sign` is the asymmetric signing algorithm the Responder selected through the last `ALGORITHMS` response message sent by the Responder. See `BaseAsymSel` or `ExtAsymSel` fields in the CHALLENGE request message table.
    - `Hash1` is the hashing algorithm the Responder selected through the last `ALGORITHMS` response message sent by the Responder. See `BaseHashSel` or `ExtHashSel` fields in the CHALLENGE request message table.
    - `SK` is the private Key associated with the Responder's slot 0 leaf certificate.

9.3.4. Signature verification

Symbols that end with `2` represent the Requester-observed messages.

1. The Requester shall create L2 as:

```
L2= Concatenate(GET_MEASUREMENTS_REQUEST2, MEASUREMENTS_RESPONSE_WITHOUT_SIGNATURE2)
```

   where:

    - `Concatenate ( )` is the standard concatenation function.
    - `GET_MEASUREMENTS _REQUEST2` is the entire contents of the `MEASUREMENTS` request message under consideration, as sent by the Requester.
    - `MEASUREMENTS_RESPONSE_WITHOUT_SIGNATURE2` is the entire contents of the `MEASUREMENTS` response message without the signature bytes, as received by the Requester.

2. The Requester shall perform:

```
Verify(PK, Hash2(L2), Signature)
```

   where:

    - `PK` is the public key associated with the slot 0 certificate of the Responder. `PK` is extracted from the `CERTIFIATES` response.
    - `Verify` is the asymmetric verification algorithm the Responder selected through the last `ALGORITHMS` response message as received by the Requester. See `BaseAsymSel` or `ExtAsymSel` fields in the CHALLENGE request message table.
    - `Hash2` is the hashing algorithm the Responder selected through the last `ALGORITHMS` response message sent as received by the Requester. See the `BaseHashSel` or `ExtHashSel` fields in the CHALLENGE request message table.

9.4. ERROR response message

For an SPDM operation that results in an error, the Responder shall send an `ERROR` response message to the Requester.

The ERROR response message table shows the `ERROR` response format.

The Error code and error data table shows the detailed error code, error data, and extended error data.

The ResponseNotReady extended error data table shows the `ResponseNotReady` extended error data.

The Registry or standards body ID table shows the registry or standards body ID.

The ExtendedErrorData format definition for vendor or other standards-defined ERROR response message table shows the `ExtendedErrorData` format definition for vendor or other standards-defined `ERROR` response message.

ERROR response message

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

Error code and error data

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 cannot respond now, but may be able to respond in the future	`0x00`	No extended error data is provided.
`UnexpectedRequest`	`04h`	The Responder received an unexpected request message. For example, `CHALLENGE` before `NEGOTIATE_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

Error code	Value	Description	Error data	ExtendedErrorData
`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.	No extended error data provided.	
`ResponseNotReady`	`42h`	See `RESPOND_IF_READY` clause.	See the ResponseNotReady extended error data table.	
Reserved	`43h-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 the Registry or standards body ID table.	See the ExtendedErrorData format definition for vendor or other standards-defined ERROR response message table for format definition.

ResponseNotReady extended error data

Offset	Field	Size (bytes)	Value
0	`RDTExponent`	1	Exponent expressed in logarithmic (base 2 scale) to calculate `RDT` time in uS after which the Responder will be able to provide successful completion response. For example, the raw value 8 indicates that the Responder will be ready in 2^8=256 uS. Responder should use `RDT` to avoid continuous pinging and issue the RESPOND_IF_READY request message table after `RDT` time. For timing requirement details, See the Timing specification for SPDM messages table.
1	`RequestCode`	1	The request code that triggered this response.
2	`Token`	1	The opaque handle that the Requester shall pass in with the the RESPOND_IF_READY request message table request message.
3	`RDTM`	1	Multiplier used to compute `WT<sub>Max</sub>` in uS to indicate the response may be dropped after this delay. The multiplier shall always be greater than 1. The Responder may also stop processing the initial request if the same Requester issues a different request. For timing requirement details, See the Timing specification for SPDM messages table.

Registry or standards body ID

ID	Vendor ID Len (bytes)	Registry or standards body name	Description
0x7D	2	USB	Vendor is identified using USB's vendor ID.
0x7E	2	PCI-SIG	Vendor is identified using PCI-SIG Vendor ID.
0x7F	4	IANA	Vendor is identified using the Internet Assigned Numbers Authority's Private Enterprise Number (PEN).

ExtendedErrorData format definition for vendor or other standards-defined ERROR response message

Byte Offset	Len	Field name	Description
0	1	Len	Length of the `VendorID` field. If the `ERROR` is vendor defined, the value of this field shall equal the `Vendor ID Len` as described in the Registry or standards body ID table of the corresponding registry or standard body name. If the `ERROR` is defined by a registry or a standard, this field shall be zero, which also indicates the `VendorID` field is not present. The registry or standards body name in the `ERROR` is indicated in the `Error Data` field, such as `Param2`, and is one of the values in the **ID** column of the Registry or standards body ID table.
1	Len	VendorID	The value of this field shall indicate the Vendor ID, as assigned by the registry or standards body. The length of this field is provided in the Registry or standards body ID table. This field shall be in little endian format. The registry or standards body name in the `ERROR` is indicated in the `Error Data` field, such as `Param2`, and is one of the values in the **ID** column of the Registry or standards body ID table.
1+Len	Variable	OpaqueErrorData	Defined by the vendor or other standards.

9.5. RESPOND_IF_READY request message

This request message shall ask for the response to the original request upon receipt of `ResponseNotReady` error code. If the response to the original request is ready, the Responder shall return that response message. If the response to the original request is not ready, the Responder shall return the `ERROR` response message, set `ErrorCode`=`ResponseNotReady` and return the same token as the previous `ResponseNotReady` response message.

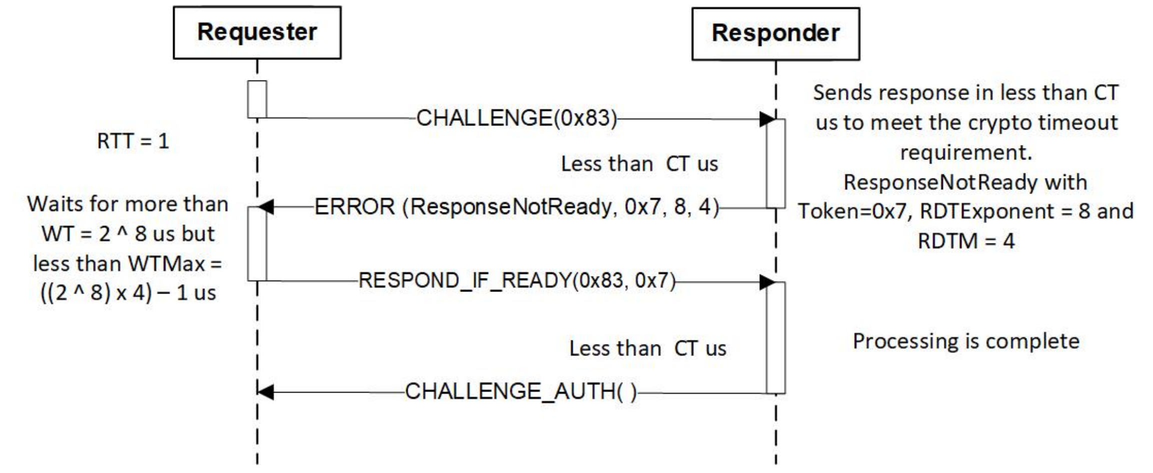
Requester
Responder
CHALLENGE(0x83)
RTT = 1
Sends response in less than CT us to meet the crypto timeout requirement.
Less than CT us
Waits for more than WT = 2 ^ 8 us but less than WTMax = ((2 ^ 8) x 4) – 1 us
ERROR (ResponseNotReady, 0x7, 8, 4)
ResponseNotReady with Token=0x7, RDTExponent = 8 and RDTM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

Figure 9 — RESPOND_IF_READY flow leading to completion

The RESPOND_IF_READY request message table shows the `RESPOND_IF_READY` request message format.

RESPOND_IF_READY request message

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0xFF=RESPOND_IF_READY`
2	`RequestCode`	1	The original request code that triggered the `ResponseNotReady` error code response. Shall match the request code returned as part of the `ResponseNotReady` extended error data.
3	`Token`	1	The token that was returned as part of the `ResponseNotReady` extended error data.

10. SPDM messaging control and discovery examples

11. ANNEX A (informative)

The 1.0 revision of this specification will address scenarios where `GET_VERSION`, `VERSION`, `GET_CAPABILITIES`, `CAPABILITIES`, `NEGOTIATE_ALGORITHMS` and `ALGORITHMS` messages can be optional.

In the v1.0 release, figures and tables will be renumbered.

12. ANNEX B - Leaf certificate example

Certificate:

```
Data:
    Version: 3 (0x2)
    Serial Number: 8 (0x8)
    Signature Algorithm: ecdsa-with-SHA256
    Issuer: C=CA, ST=NC, L=city, O=ACME, OU=ACME Devices, CN=CA
    Validity
        Not Before: Jan  1 00:00:00 1970 GMT
        Not After : Dec 31 23:59:59 9999 GMT
    Subject: C=US, ST=NC, O=ACME Widget Manufacturing, OU=ACME Widget Manufacturing Unit, CN=w0123456789
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            RSA Public-Key: (2048 bit)
            Modulus:
                00:ba:67:47:72:78:da:28:81:d9:81:9b:db:88:03:
                e1:10:a4:91:b8:48:ed:6b:70:3c:ec:a2:68:a9:3b:
                5f:78:fc:ae:4a:d1:1c:63:76:54:a8:40:31:26:7f:
                ff:3e:e0:bf:95:5c:4a:b4:6f:11:56:ca:c8:11:53:
                23:e1:1d:a2:7a:a5:f0:22:d8:b2:fb:43:da:dd:bd:
                52:6b:e6:a5:3f:0f:3b:60:b8:74:db:56:08:d9:ee:
                a0:30:4a:03:21:1e:ee:60:ad:e4:00:7a:6e:6b:32:
                1c:28:7e:9c:e8:c3:54:db:63:fd:1f:d1:46:20:9e:
                ef:80:88:00:5f:25:db:cf:43:46:c6:1f:50:19:7f:
                98:23:84:38:88:47:5d:51:8e:11:62:6f:0f:28:77:
                a7:20:0e:f3:74:27:82:70:a7:96:5b:1b:bb:10:e7:
                95:62:f5:37:4b:ba:20:4e:3c:c9:18:b2:cd:4b:58:
                70:ab:a2:bc:f6:2f:ed:2f:48:92:be:5a:cc:5c:5e:
                a8:ea:9d:60:e8:f8:85:7d:c0:0d:2f:6a:08:74:d1:
                2f:e8:5e:3d:b7:35:a6:1d:d2:a6:04:99:d3:90:43:
                66:35:e1:74:10:a8:97:3b:49:05:51:61:07:c6:08:
                01:1c:dc:a8:5f:9e:30:97:a8:18:6c:f9:b1:2c:56:
                e8:67
            Exponent: 65537 (0x10001)
            X509v3 extensions:
```

```
        X509v3 Basic Constraints:
            CA:FALSE
        X509v3 Key Usage:
            Digital Signature, Non Repudiation, Key Encipherment
        X509v3 Subject Alternative Name:
            othername:1.3.6.1.4.1.412.274.1;UTF8STRING:ACME:WIDGET:0123456789
        Signature Algorithm: ecdsa-with-SHA256
        Signature Value:
            30:45:02:21:00:fc:8f:b0:ad:6f:2d:c3:2a:7e:92:6d:29:1d:
            c7:fc:0d:48:b0:c6:39:5e:c8:76:d6:40:9a:12:46:c3:39:0e:
            36:02:20:1a:ea:3a:59:ca:1e:bc:6d:6e:61:79:af:a2:05:7c:
            7d:da:41:a9:45:6d:cb:04:49:43:e6:0b:a8:8d:cd:da:e
```

13. Change log

Version	Date	Description
0.9.0	2019-05-30	First draft version.

14. Bibliography

DMTF DSP4014, *DMTF Process for Working Bodies 2.6*, https://www.dmtf.org/sites/default/files/standards/documents/DSP4014_2.6.pdf

SPDMSpecification🗑