Security Protocol and Data Model (SPDM) Specification

Version: 1.0.3

Document Identifier: DSP0274

Date: 2026-04-03

Version History: https://www.dmtf.org/dsp/DSP0274

Supersedes: 1.0.2

Document Class: Normative

Document Status: Stabilized

Document Language: en-US

10 DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. Members and non-members may reproduce DMTF specifications and documents for uses consistent with this purpose, provided that correct attribution is given. As DMTF specifications may be revised from time to time, the particular version and release date should always be noted.

11 Implementation of certain elements of this standard or proposed standard may be subject to third-party patent rights, including provisional patent rights (herein “patent rights”). DMTF makes no representations to users of the standard as to the existence of such rights and is not responsible to recognize, disclose, or identify any or all such third-party patent right owners or claimants, nor for any incomplete or inaccurate identification or disclosure of such rights, owners, or claimants. DMTF shall have no liability to any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize, disclose, or identify any such third-party patent rights, or for such party’s reliance on the standard or incorporation thereof in its products, protocols, or testing procedures. DMTF shall have no liability to any party implementing such standards, whether such implementation is foreseeable or not, nor to any patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a patent owner for such implementations.

12 For information about patents held by third parties which have notified DMTF that, in their opinion, such patents may relate to or impact implementations of DMTF standards, visit https://www.dmtf.org/about/policies/disclosures.

13 This document's normative language is English. Translation into other languages is permitted.

CONTENTS

1 Foreword

The Security Protocols and Data Models (SPDM) Working Group of 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 DMTF, see https://www.dmtf.org.

2 Acknowledgments

DMTF acknowledges the following individuals for their 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
- Philip Hawkes — Qualcomm Inc.
- 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 Matthews — Advanced Micro Devices, Inc.
- Mahesh Natu — Intel Corporation
- Edward Newman — Hewlett Packard Enterprise
- Jim Panian — Qualcomm Inc.
- 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

3 Abstract

The *Security Protocol and Data Model (SPDM) Specification* defines *messages*, data objects, sequences, and states 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 and/or hardware identities. The SPDM enables efficient access to low-level security capabilities and operations. Other mechanisms, including non-SPDM- and DMTF-defined mechanisms, can use the SPDM.

21 # 4 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.

22 ## 4.1 Scope

23 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.

24 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.

25 ## 4.2 Normative references

26 The following 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.

- DMTF DSP0004, *Common Information Model (CIM) Metamodel*, https://www.dmtf.org/sites/default/files/standards/documents/DSP0004_3.0.1.pdf
- DMTF DSP0223, *Generic Operations*, https://www.dmtf.org/sites/default/files/standards/documents/DSP0223_1.0.1.pdf
- DMTF DSP0233, *Management Component Transport Protocol (MCTP) I3C Transport Binding Specification*, https://www.dmtf.org/sites/default/files/standards/documents/DSP0233_1.0.0.pdf
- DMTF DSP0236, *MCTP Base Specification 1.3.0*, https://dmtf.org/sites/default/files/standards/documents/DSP0236_1.3.0.pdf
- DMTF DSP0239, *MCTP IDs and Codes 1.6.0*, https://www.dmtf.org/sites/default/files/standards/documents/DSP0239_1.6.0.pdf
- DMTF DSP0240, *Platform Level Data Model (PLDM) Base Specification*, https://www.dmtf.org/sites/default/files/standards/documents/DSP0240_1.0.0.pdf
- DMTF DSP0275, *Security Protocol and Data Model (SPDM) over MCTP Binding Specification*, https://www.dmtf.org/dsp/DSP0275
- DMTF DSP1001, *Management Profile Usage Guide*, https://www.dmtf.org/sites/default/files/standards/documents/DSP1001_1.2.0.pdf

- *ISO/IEC Directives, Part 2, Principles and rules for the structure and drafting of ISO and IEC documents - 2018 (8th edition)*
- *ISO/IEC Directives, Part 2, Principles and rules for the structure and drafting of ISO and IEC documents*, https://www.iso.org/sites/directives/current/part2/index.xhtml
- IETF RFC5234, *Augmented BNF for Syntax Specifications: ABNF*, January 2008
- IETF RFC5280, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, May 2008
- *USB Authentication Specification Rev 1.0 with ECN and Errata through January 7, 2019*
- *TCG Algorithm Registry, Family "2.0", Level 00 Revision 01.27*, February 7, 2018
- **ASN.1 — ISO-822-1-4**
  - ITU-T X.680, 08/2015
  - ITU-T X.681, 08/2015
  - ITU-T X.682, 08/2015
  - ITU-T X.683, 08/2015
- **DER — ISO-8825-1**
  - ITU-T X.690, 08/2015
- **X.509 — ISO-9594-8**
  - ITU-T X.509, 10/2012
- **ECDSA**
  - Section 6, The Elliptic Curve Digital Signature Algorithm (ECDSA) in FIPS PUB 186-4 Digital Signature Standard (DSS)
  - Appendix D: Recommended Elliptic Curves for Federal Government Use in FIPS PUB 186-4 Digital Signature Standard (DSS)
- **RSA**
  - Table 3 in *TCG Algorithm Registry Family "2.0" Level 00 Revision 01.22*, February 9, 2015
- **SHA2-256**, **SHA2-384**, and **SHA2-512**
  - FIPS PUB 180-4 Secure Hash Standard (SHS)
- **SHA3-256**, **SHA3-384**, and **SHA3-512**
  - FIPS PUB 202 SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions

4.3 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 parenthesis 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.

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

31 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.

32 The terms that DSP0004, DSP0233, DSP0236, DSP0239, DSP0275, and DSP1001 define also apply to this document.

33 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*. 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 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	Physical entity similar to the PCI Express specification's definition.
device	Physical entity such as a network card or a fan.
DMTF	Formerly known as the Distributed Management Task Force, DMTF creates open manageability standards that span diverse emerging and traditional information technology (IT) infrastructures, including cloud, virtualization, network, servers, and storage. Member companies and alliance partners worldwide collaborate on standards to improve the interoperable management of IT.
endpoint	Logical entity that communicates with other endpoints over one or more transport protocols.
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.

Term	Definition
message transcript	The concatenation of a sequence of messages in the order in which an endpoint sends and receives them. The final message included in the message transcript may be truncated to allow inclusion of a signature in that message, which is computed over the message transcript.
most significant byte (MSB)	Highest order *byte* in a number consisting of multiple bytes.
Negotiated State	Set of parameters that represent the state of the communication between a corresponding pair of Requester and Responder at the successful completion of the `NEGOTIATE_ALGORITHMS` messages. These parameters may include values provided in `VERSION`, `CAPABILITIES` and `ALGORITHMS` messages. Additionally, they may include parameters associated with the transport layer. They may include other values deemed necessary by the Requester or Responder to continue or preserve communication with each other.
nibble	Computer term for a four-bit aggregation, or half of a byte.
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.
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^2C, PCI Express™ Vendor Defined Messaging, and so on.
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.
Security Protocols and Data Models (SPDM)	Working group under DMTF that is responsible for the *SPDM Specification*, which focuses on enabling authentication, attestation, and key exchange to enhance infrastructure security. In addition to developing the core *SPDM Specification*, the group collaborates with other standards organizations and developers to support alignment across the industry in the areas of component authentication, confidentiality, and integrity.
SPDM message	Unit of communication in SPDM communications.

Term	Definition
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.
trusted computing base (TCB)	Set of all hardware, firmware, and/or software components that are critical to its security, in the sense that bugs or vulnerabilities occurring inside the TCB might jeopardize the security properties of the entire system. By contrast, parts of a computer system outside the TCB must not be able to misbehave in a way that would leak any more privileges than are granted to them in accordance to the security policy. Reference: Trusted computing base

34 ## 4.4 Symbols and abbreviated terms

35 The abbreviations defined in DSP0004, DSP0223, and DSP1001 apply to this document.

36 The following additional abbreviations are used in this document.

Abbreviation	Definition
CA	*certificate authority*
CSR	*certificate signing request*
MAC	Message Authentication Code
DMTF	Formerly the *Distributed Management Task Force*
MSB	*most significant byte*
SPDM	*Security Protocol and Data Model*
TCB	*trusted computing base*

37 ## 4.5 Conventions

38 The following conventions apply to all SPDM specifications.

4.5.1 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.

4.5.2 Byte ordering

Unless otherwise specified, for all SPDM specifications *byte* ordering of multibyte numeric fields or multibyte bit fields is *little endian*. That is, the lowest byte offset holds the least significant byte, and higher offsets hold the more significant bytes.

4.5.3 Sizes and lengths

Unless otherwise specified, all sizes and lengths are in units of bytes.

4.5.4 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.

SPDM data types

Data type	Interpretation
`ver8`	Eight-bit encoding of the SPDM version number. Version encoding defines the encoding of the 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.

4.5.5 Version encoding

The `SPDMVersion` field represents the version of the specification through a combination of *Major* and *Minor* nibbles, encoded as follows:

Version encoding

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

Version	Matches	Incremented when
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 only, but the available functionality is limited to what SPDM specification Version 1.0 defines.

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

The detailed version encoding that the `VERSION` response message returns contains an additional byte that indicates specification bug fixes or development versions. See the Successful VERSION response message table.

4.5.6 Notations

SPDM specifications use the following notations:

SPDM notations

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 bit is on the left, and the least significant bit is on the right.
`1b`	A lowercase `b` after a number consisting of `0`s and `1`s indicates that the number is in binary format.
`0x12A`	A leading `0x` indicates that the number is in hexadecimal format.
`7+`	This indicates a variable length byte range that starts at byte offset 7.

4.5.7 Other conventions

Unless otherwise specified, all figures are informative.

4.6 SPDM message exchanges

The message 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 request/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 a Responder role. These two streams are mutually exclusive.

The message exchanges defined in this specification include Requesters that:

- Discover and negotiate the security capabilities of a Responder.
- Authenticate the identity of a Responder.
- Retrieve the firmware measurements of a Responder.

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

4.6.1 Security capability discovery and negotiation

This specification defines a mechanism for a 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 a Requester and Responder to select a common set of cryptographic algorithms to use for all subsequent message exchanges before another negotiation is initiated by the Requester, if an overlapping set of cryptographic algorithms exists that both endpoints support.

4.6.2 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 the identity of a Responder 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 that corresponds 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.

4.6.3 Firmware and configuration measurement

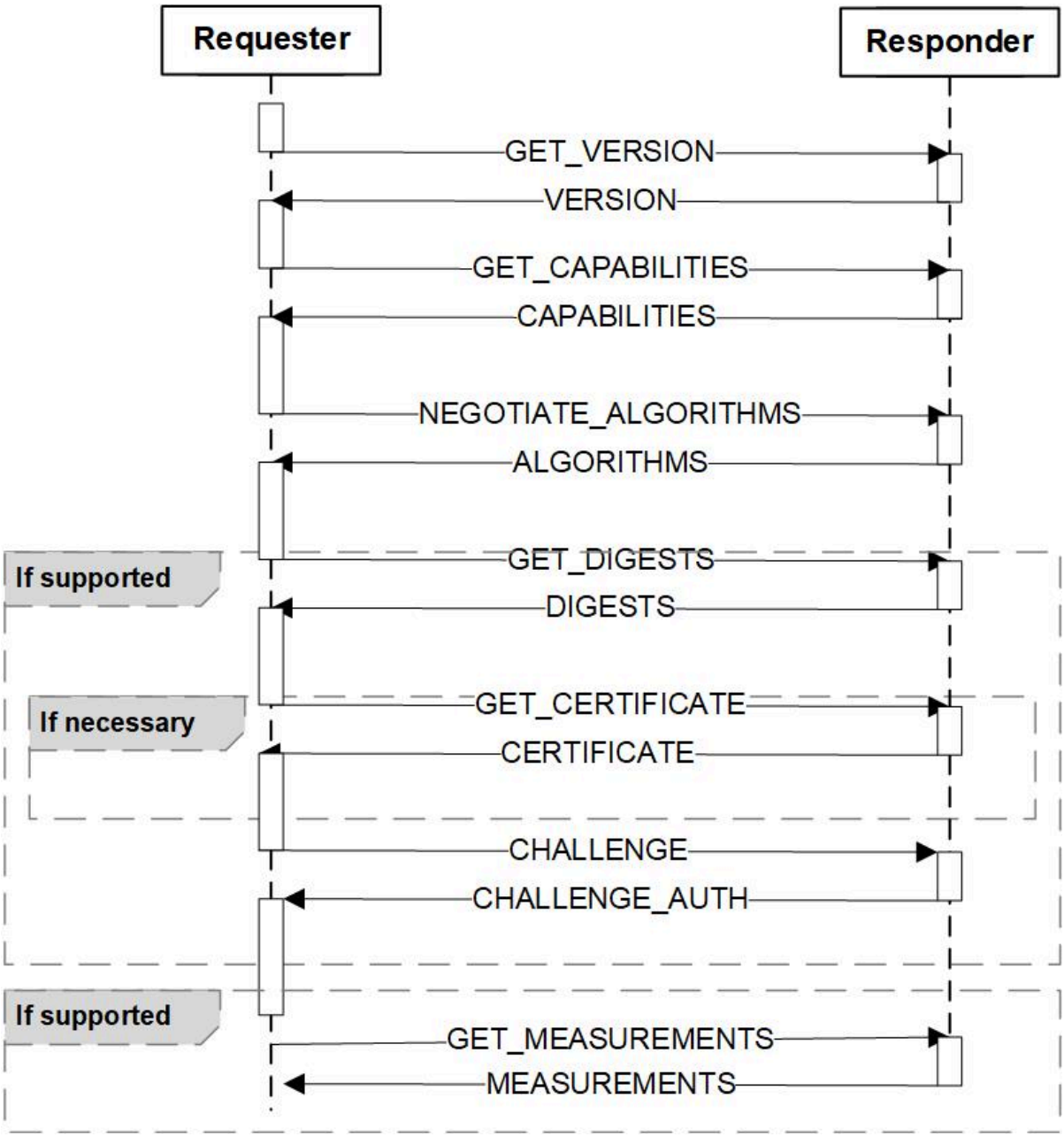
Measurement is a term that describes the process of calculating the cryptographic hash value of a piece of firmware/software or configuration data and tying the cryptographic hash value with the endpoint identity through the use of digital signatures. This allows an authentication initiator to establish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

4.7 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.

The SPDM messaging protocol flow is an example of a 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.

SPDM messaging protocol flow



87 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.

88 The Requester shall issue `GET_VERSION`, `GET_CAPABILITIES`, and `NEGOTIATE_ALGORITHMS` request messages before issuing any other request messages. The responses to `GET_VERSION`, `GET_CAPABILITIES`, and `NEGOTIATE_ALGORITHMS` may be saved by the requester so that after the next reset the requester may skip these requests.

89 ### 4.7.1 Generic SPDM message format

90 The Generic SPDM message field definitions table defines the fields that constitute a generic SPDM message, including the message header and payload.

91 **Generic SPDM message field definitions**

Byte	Bits	Length (bits)	Field name	Description
`0`	`[7:4]`	4	SPDM Major Version	The major version of the SPDM Specification. An endpoint shall not communicate by using an incompatible SPDM version value. See Version encoding.
`0`	`[3:0]`	4	SPDM Minor Version	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.
`1`	`[7:0]`	8	Request Response Code	The request message code or response code, which are enumerated in Table 4 and Table 5. `0x00` through `0x7F` represent response codes and `0x80` through `0xFF` represent request codes. In request messages, this field is considered the request code. In response messages, this field is considered the response code.
`2`	`[7:0]`	8	Param1	The first one-byte parameter. The contents of the parameter is specific to the Request Response Code.
`3`	`[7:0]`	8	Param2	The second one-byte parameter. The contents of the parameter is specific to the Request Response Code.
`4`	See Description	Variable	SPDM message payload	Zero or more bytes that are specific to the Request Response Code.

92 ### 4.7.2 SPDM request codes

93 The SPDM request codes table defines the SPDM request codes. The **Implementation Requirement** column indicates requirements on the Requester.

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

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

96 **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.
`VENDOR_DEFINED_REQUEST`	`0xFE`	Optional	See the VENDOR_DEFINED_REQUEST request message table.
Reserved	`0x80`, `0x85` - `0xDF`, `0xE2`, `0xE4` - `0xFD`		SPDM implementations compatible with this version shall not use the reserved request codes.

97 ### 4.7.3 SPDM response codes

98 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 SPDM response codes.

99 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.

100 The SPDM response codes table defines the response codes for SPDM. The **Implementation Requirement** column indicates requirements on the Responder.

101 **SPDM response codes**

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

4.8 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 response messages independently.

4.8.1 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. If the Requester has sent a request to a Responder and wants to send a subsequent request to the same Responder, then the Requester shall wait to send the subsequent request until after 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, from the transport layer, that transmission of the request message failed.

A Requester may send simultaneous request messages to different Responders.

4.8.2 Requirements for Responders

A Responder is not required to process more than one request message at a time.

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.

If a Responder is working on a request message from a Requester, the Responder may respond with `ErrorCode=Busy`.

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

4.8.3 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, but that is outside of the SPDM specification.

4.8.3.0.1 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.

4.8.3.1 Timing specification table

The **Ownership** column in the Timing specification for SPDM messages table specifies whether the timing parameter applies to the Responder or Requester.

Timing specification for SPDM messages

Timing parameter	Ownership	Value	Units	Description
RTT	Requester	See the description.	ms	Worst-case round-trip transport timing. The maximum 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- or media-specific.
ST1	Responder	100	ms	Shall be the maximum amount of time the Responder has to provide a response to requests that do not require cryptographic processing, such as the `GET_CAPABILITIES`, `GET_VERSION`, or `NEGOTIATE_ALGORITHMS` request messages.
T1	Requester	RTT + ST1	ms	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`.

Timing parameter	Ownership	Value	Units	Description
CT	Responder	$2^{CTExponent}$	µs	The `CAPABILITIES` message reports the cryptographic timeout, in microseconds. `CTExponent` is reported in . This timing parameter shall be the maximum amount of time the Responder has to provide any response requiring cryptographic processing, such as the `GET_MEASUREMENTS` or `CHALLENGE` request messages.
T2	Requester	RTT + CT	µs	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`.
RDT	Responder	$2^{RDTExponent}$	µs	Recommended delay, in microseconds. When the Responder cannot complete cryptographic processing response within the `CT` time, it shall provide `RDTExponent` as part of the `ERROR` response. See the ResponseNotReady extended error data table for the `RDTExponent` value. For details, see `ErrorCode=ResponseNotReady` in the ResponseNotReady extended error data table.

Timing parameter	Ownership	Value	Units	Description
WT	Requester	RDT	µs	Amount of time that the Requester should wait before issuing the `RESPOND_IF_READY` request message. 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 transmission time of the `ERROR` from the Responder to Requester when calculating `WT`. For details, see `RDT`.
WT_{Max}	Requester	(RDT * RDTM) - RTT	µs	Maximum wait time the Requester has to issue `RESPOND_IF_READY` request unless the Requester issued a successful `RESPOND_IF_READY` request message 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 `RDTM` value appears in the ResponseNotReady extended error data. The Responder should ensure that `WT Max` does not result in less than `WT` in determination of `RDTM`. For details, see `ErrorCode=ResponseNotReady` in the ResponseNotReady extended error data table.

4.9 SPDM messages

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

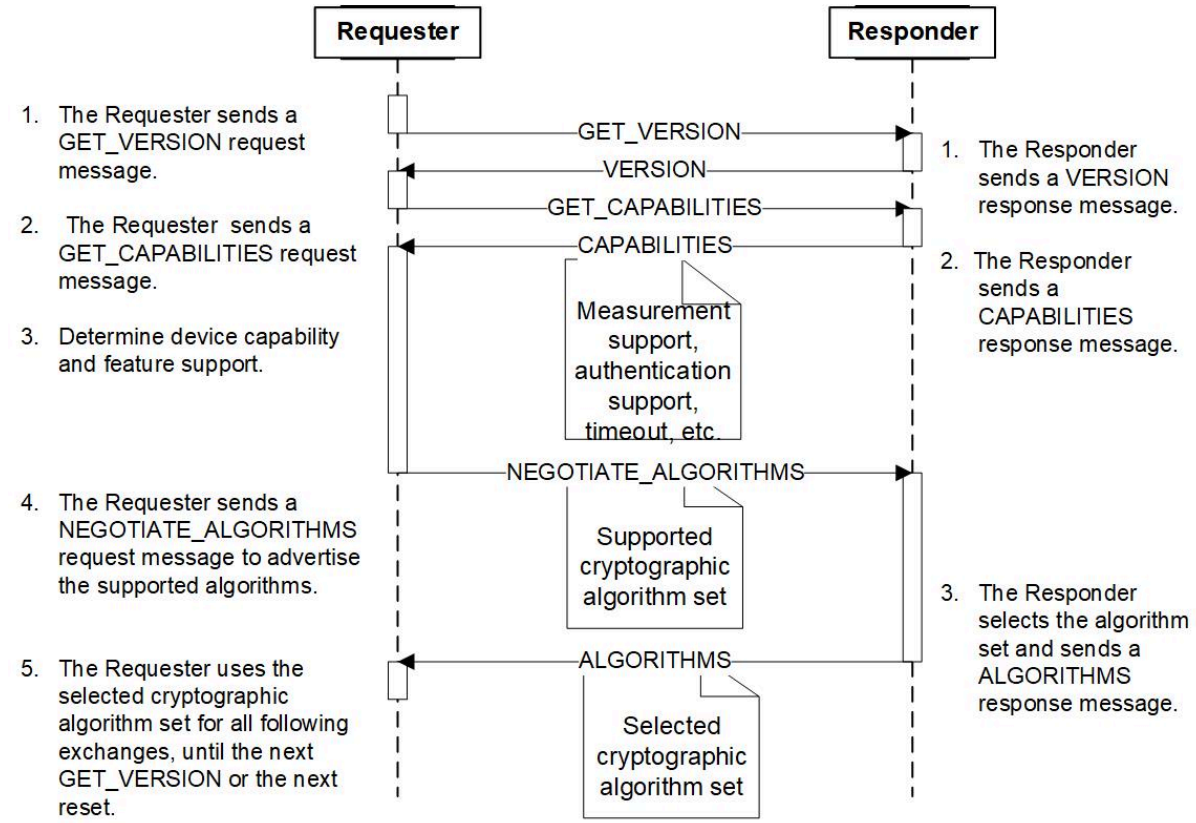
- Capability discovery and negotiation
- Responder identity authentication
- Request ordering and message transcript computation rules for M1 and M2

4.9.1 Capability discovery and negotiation

All Requesters and Responders shall support `GET_VERSION`, `GET_CAPABILITIES` and `NEGOTIATE_ALGORITHMS`.

The Capability discovery and negotiation flow shows the high-level request-response flow and sequence for the capability discovery and negotiation:

Capability discovery and negotiation flow



4.9.1.1 GET_VERSION request message and VERSION response message

This request message shall retrieve the SPDM version of an endpoint. 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.

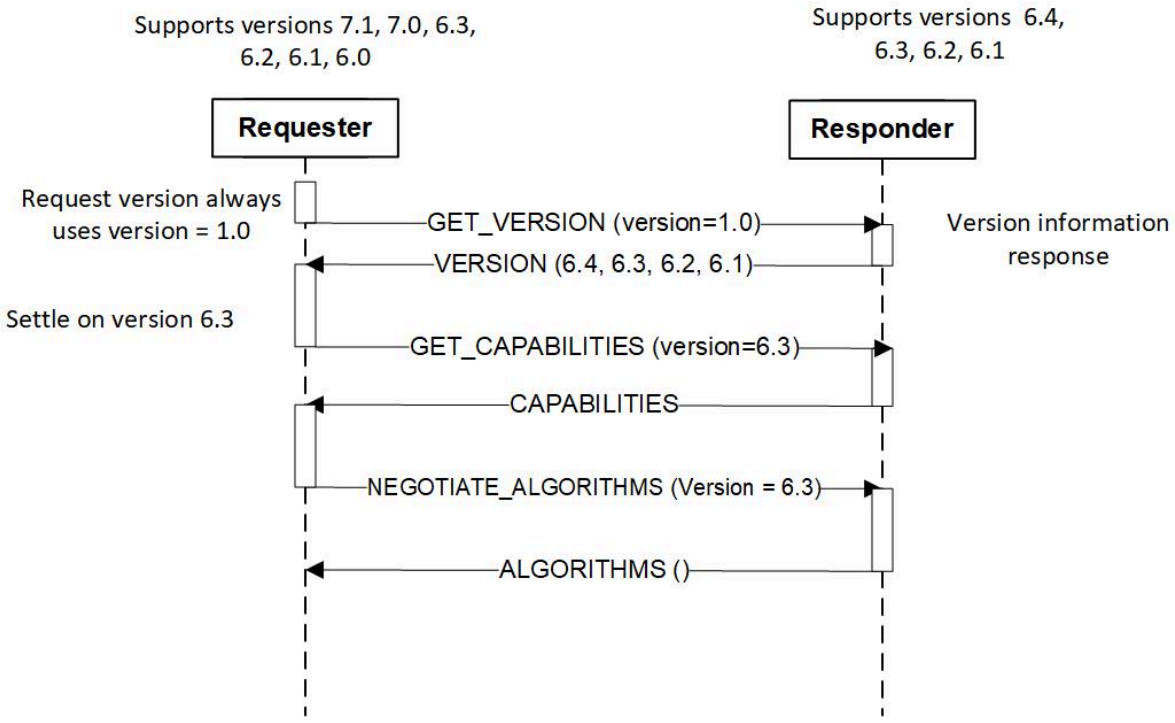
The Requester shall begin the discovery process by sending a `GET_VERSION` request message with major version 0x1. All Responders must always support `GET_VERSION` request message with major version `0x1` and provide a `VERSION` response containing all supported versions, as the GET_VERSION request message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

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 an `ERROR` message except for `ErrorCode`s specified in this clause.

A Requester may issue a `GET_VERSION` request message to a Responder at any time, which is as an exception to Requirements for Requesters for the case where a Requester must restart the protocol due to an internal error or reset.

After receiving a `GET_VERSION` request, the Responder shall cancel all previous requests from the same Requester. Additionally, this message shall clear or reset the previously *Negotiated State*, if any, in both the Requester and its corresponding Responder.

Discovering the common major version



138 **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

139 **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).

Offset	Field	Size (bytes)	Value
6	`VersionNumberEntry1:n`	2 x n	16-bit version entry. See the GET_VERSION request message table.

140 **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.

141 #### 4.9.1.2 GET_CAPABILITIES request message and CAPABILITIES response message

142 This request message shall retrieve the security capabilities of an endpoint.

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

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

145 The Flag fields definitions table shows the flag fields definitions.

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

147 **GET_CAPABILITIES request message**

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`

Offset	Field	Size (bytes)	Value
1	`RequestResponseCode`	1	`0xE1=GET_CAPABILITIES`
2	`Param1`	1	Reserved
3	`Param2`	1	Reserved

148 **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`. See the Timing specification for SPDM messages table. The equation for `CT` shall be 2^{CT} microseconds (µs). For example, if `CTExponent` is 10, `CT` is 2^{10}=1024 µs.
6	`Reserved`	2	Reserved
8	`Flags`	4	See the SPDM data types table.

149 **Flag fields definitions**

Byte	Bit	Field	Value
0	0	CACHE_CAP	If set, the Responder shall support 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 shall support GET_DIGESTS and GET_CERTIFICATE messages.
0	2	CHAL_CAP	If set, Responder shall support CHALLENGE request message.
0	4:3	MEAS_CAP	MEASUREMENT capabilities of the Responder. 00b. The Responder shall not support MEASUREMENTS capabilities. 01b. The Responder shall support MEASUREMENTS but cannot generate signatures. 10b. The Responder shall support MEASUREMENTS and can generate signatures. 11b. Reserved

Byte	Bit	Field	Value
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

4.9.1.3 NEGOTIATE_ALGORITHMS request message and ALGORITHMS response message

This request message shall negotiate cryptographic algorithms. A Requester shall not issue a 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_VERSION until it receives a successful ALGORITHMS response message.

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

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, in bytes. Length shall be less than 64 bytes.
6	`MeasurementSpecification`	1	Bit mask. The `MeasurementSpecification` field of the GET_MEASUREMENTS request message and MEASUREMENTS response message shall define the values for this field. The Requester may set more than one bit to indicate multiple measurement specification support.
7	`Reserved`	1	Reserved

Offset	Field	Size (bytes)	Value
8	*BaseAsymAlgo*	4	Bit mask listing Requester-supported SPDM-enumerated asymmetric key signature algorithms for the purposes of signature verification by the Requester. Byte 0 Bit 0. TPM_ALG_RSASSA_2048 Byte 0 Bit 1. TPM_ALG_RSAPSS_2048. The salt length should be the same as the output length of the selected hash function. Byte 0 Bit 2. TPM_ALG_RSASSA_3072 Byte 0 Bit 3. TPM_ALG_RSAPSS_3072. The salt length should be the same as the output length of the selected hash function. 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. The salt length should be the same as the output length of the selected hash function. Byte 0 Bit 7. TPM_ALG_ECDSA_ECC_NIST_P384 Byte 1 Bit 0. TPM_ALG_ECDSA_ECC_NIST_P521 All other values reserved.

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 Extended algorithm field format table describes the format of this field.
32+4*A	*ExtHash*	4*E	List of the extended hashing algorithms supported by Requester. The Extended algorithm field format table describes the format of this field.

Successful ALGORITHMS response message

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0x63=ALGORITHMS`

Offset	Field	Size (bytes)	Value
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 `MeasurementSpecification` field of the Measurement block format table defines the values in this field.
7	`Reserved`	1	Reserved

Offset	Field	Size (bytes)	Value
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. 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. 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

159

Offset	Field	Size (bytes)	Value
			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 for the purposes of signature generation by the Responder. A Responder that returns `CHAL_CAP=0` and `MEAS_CAP!=2` shall set this field to `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 to `0`. Other Responders shall set no more than one bit.
20	`Reserved`	12	Reserved.
32	`ExtAsymSelCount`	1	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 to `0`.

Offset	Field	Size (bytes)	Value
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 to `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 by using this algorithm. The Responder shall use this asymmetric signature algorithm for all subsequent applicable response messages to the Requester. The Extended algorithm field format table describes the format of this field.
36+4*A'	`ExtHashSel`	4*E'	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 Extended algorithm field format table describes the format of this field.

160 **Extended algorithm field format**

Offset	Field	Description
0	Registry ID	Shall represent the registry or standards body. The **ID** column in the Registry or standards body ID table describes the value of this field.
1	Reserved	Reserved
[2:3]	Algorithm ID	Shall indicate the desired algorithm. The registry or standards body owns the value of this field. For details, see the Registry or standards body ID table.

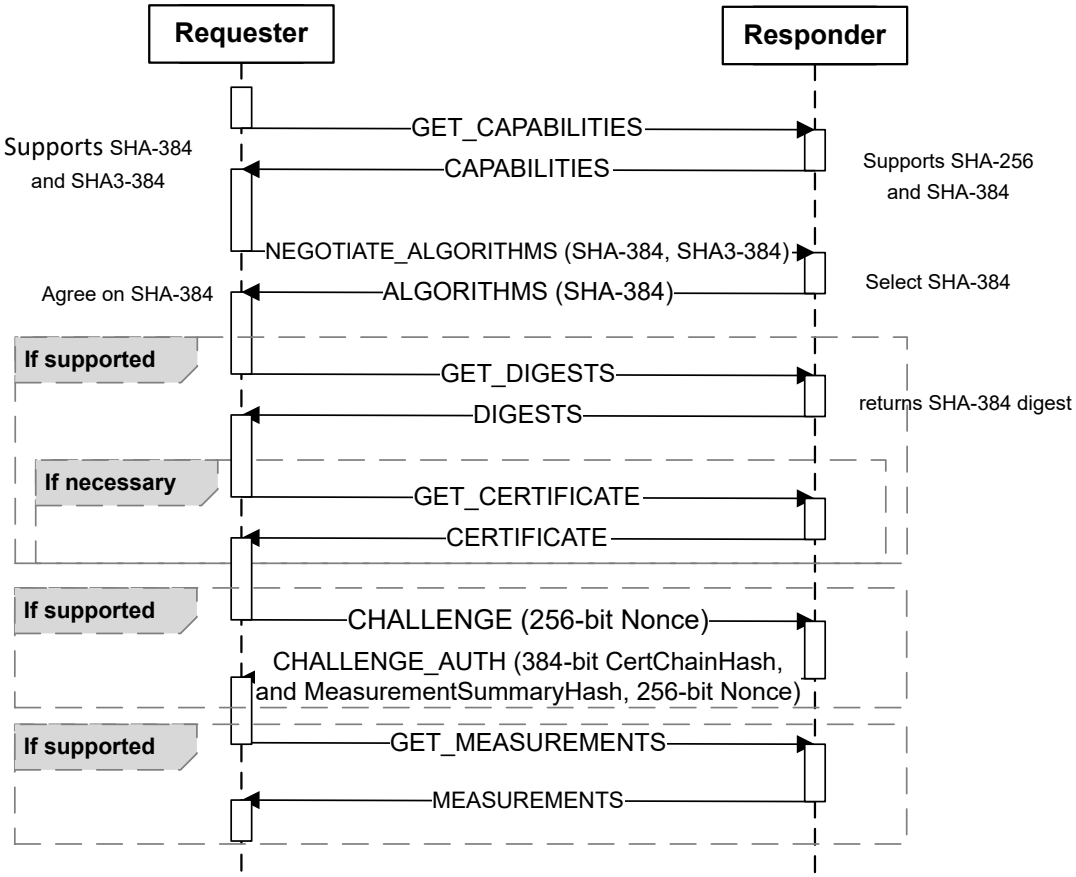
161 A Responder shall not select both an SPDM-enumerated asymmetric key signature algorithm and an extended asymmetric key signature algorithm. A Responder shall not select both an SPDM-enumerated hashing algorithm and an extended hashing algorithm.

162 This clause illustrates how two endpoints negotiate a base hashing algorithm.

163 In Hashing algorithm selection: Example 1, endpoint A issues `NEGOTIATE_ALGORITHMS` request message and endpoint B selects an algorithm of which both endpoints are capable.

164 **Hashing algorithm selection: Example 1**

165



166 The SPDM protocol accounts for the possibility that both endpoints may issue `NEGOTIATE_ALGORITHMS` request messages independently of each other. In this case, the 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.

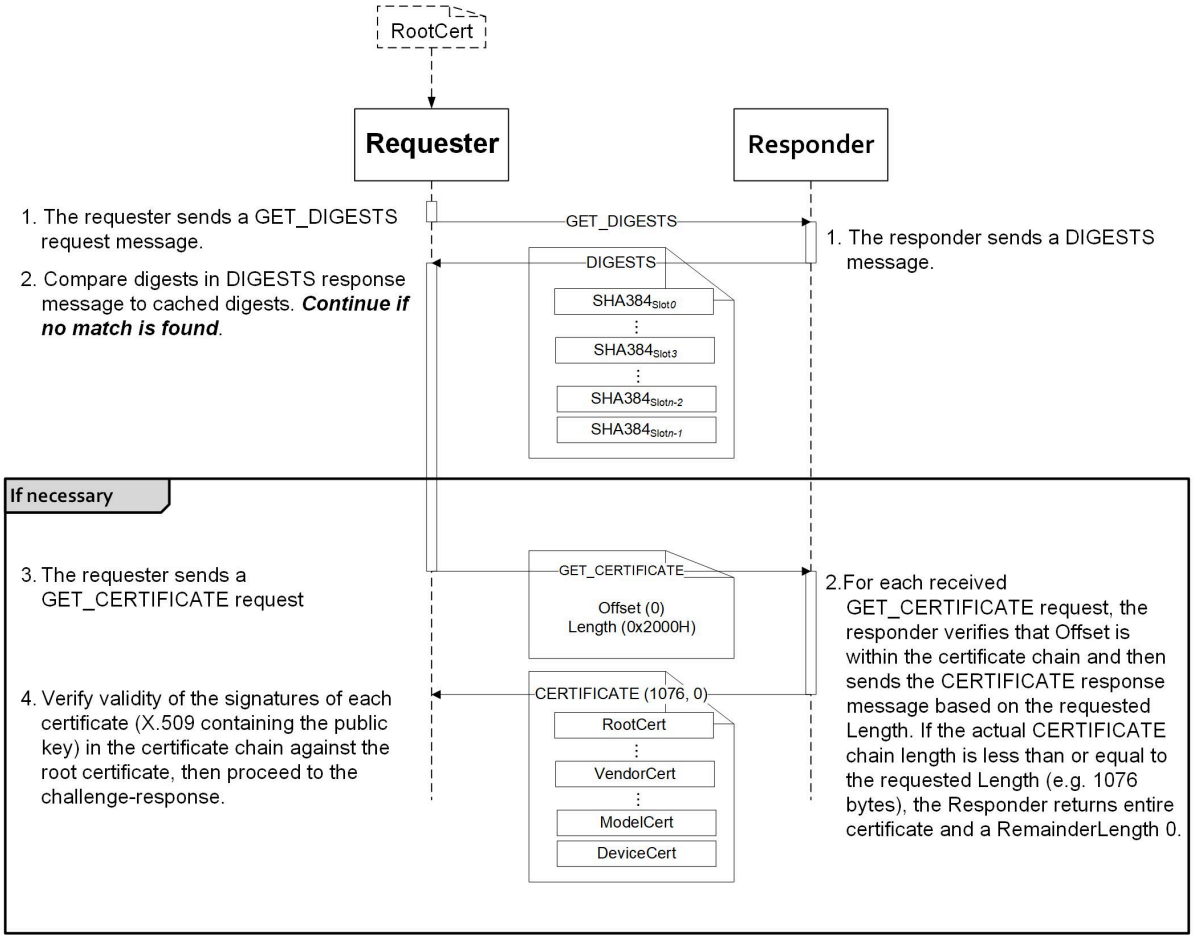
167 ### 4.9.2 Responder identity authentication

168 This clause describes request messages and response messages associated with the identity authentication operations of a Responder. All request messages in this clause shall be supported by a Responder that returns `CERT_CAP=1` and/or `CHAL_CAP=1` in the `CAPABILITIES` response message.

169 The Responder authentication: Example certificate retrieval flow shows the high-level request-response message flow and sequence for the identity authentication for *certificate* retrieval of a Responder.

170 **Responder authentication: Example certificate retrieval flow**

171



172 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.

173 For the runtime challenge-response flow, the signature field in the `CHALLENGE_AUTH` response message payload shall contain the signature generated by using the device private key over the hash of the message transcript. See the Request ordering and message transcript computation rules for M1/M2 table.

174 This ensures cryptographic binding between a specific request message from a specific Requester and a specific response message from a specific Responder and enables the Requester to detect the presence of an active adversary attempting to downgrade cryptographic algorithms or SPDM versions.

175 Furthermore, a Requester-generated nonce protects the challenge-response from replay attacks, whereas a Responder-generated nonce prevents the Responder from signing over arbitrary data that the Requester dictates. The signature computation is restarted with the latest `GET_VERSION` request received.

176 #### 4.9.2.1 Certificates and certificate chains

177 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 that the Certificate chain format shows.

178 Each certificate in a chain shall be a DER-encoded ASN.1 X.509 v3 certificate where RFC5280 defines the ASN.1 X.509 v3 certificate format and where X.690 defines the DER encoding of ASN.1 structures. 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.

179 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 eight 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 identifies the certificate chains from the eight slots.

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

181 **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 X.509 v3 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.

182 #### 4.9.2.2 GET_DIGESTS request message and DIGESTS response message

183 This request message shall be used to retrieve the certificate chain digests.

184 The GET_DIGESTS request message table shows the `GET_DIGESTS` request message format.

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

186 The digests in the Successful DIGESTS response message table shall be big endian, and the digest shall be computed over the certificate chain as shown in Certificate chain format.

187 **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

188 **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.) 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 (n^{th}) certificate chain.

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

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 PortionLength values in all previous `GET_CERTIFICATE` responses.

Offset	Field	Size (bytes)	Value
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 value is the smaller of the following values: Capacity of the internal buffer of the Requester for receiving the certificate chain of the Responder. The `RemainderLength` of the preceding `GET_CERTIFICATE` response. For the first `GET_CERTIFICATE` request, the Requester should use the capacity of the receiving buffer of the Requester. If `offset=0` and `length=0xFFFF`, the Requester is requesting the entire chain.

195 **Successful CERTIFICATE response message**

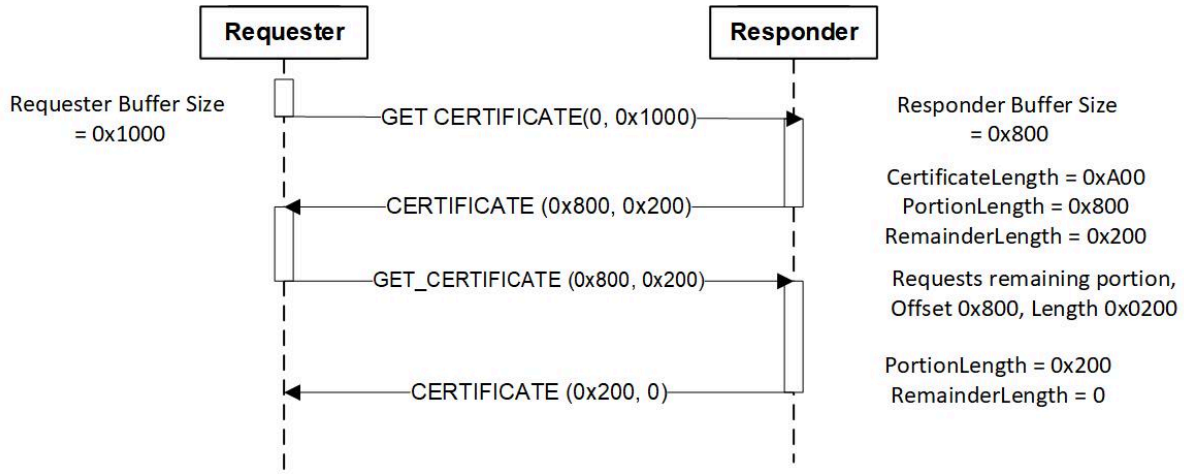
Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0x02=CERTIFICATE`
2	`Param1`	1	Slot number of the certificate chain returned.
3	`Param2`	1	Reserved.

Offset	Field	Size (bytes)	Value
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 internal buffer of the Responder.
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, as described in Certificates and certificate chains.

196 The Responder unable to return full length data flow 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.

197 **Responder unable to return full length data flow**

198



199 #### 4.9.2.4 Leaf certificate

200 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.

201 Early expiration of any certificate in the certificate chain can cause the entire certificate chain to be rejected. Certificates from the Root CA Certificate to the certificate that endorses the leaf certificate should have a `notAfter` date that is the same as or later than the leaf certificate. Alternatively for certificate chains stored in slots 1-7, there might exist the ability to update the certificate chain to refresh the validity period.

202 ##### 4.9.2.4.1 Required fields

Field	Description
`Version`	Version of the encoded certificate shall be present and shall be `3` (encoded as value `2`).
`Serial Number`	CA-assigned serial number shall be present with a positive integer value.
`Signature Algorithm`	Signature algorithm that CA uses shall be present.
`Issuer`	CA distinguished name shall be specified.
`Subject Name`	Subject name shall be present and shall represent the distinguished name associated with the leaf certificate.
`Validity`	Certificates may include this attribute. See RFC5280 for further details.
`Subject Public Key Info`	Device public key and the algorithm shall be present.
`Extended Key Usage`	Shall be present and key usage bit for digital signature shall be set.

203 ##### 4.9.2.4.2 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 with a specific format to represent the device information. The use of the `otherName` field also provides flexibility for other alliances to provide device specific information as part of the device certificate. See the Definition of otherName using the DMTF OID.

4.9.2.4.3 Definition of otherName using the DMTF OID

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

-- All printable characters except ":" --
DMTF-device-string                     ::= UTF8String (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 A (normative) Leaf certificate example shows an example leaf certificate.

4.9.2.5 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 certificate chain of the Responder that shall be used for authentication.

Offset	Field	Size (bytes)	Value
3	`Param2`	1	Requested measurement summary hash Type: `0x0` . No measurement summary hash. `0x1=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 `0x0` .
4	`Nonce`	32	The Requester should choose a random value.

211 **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. The Requester can use this value 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.

Offset	Field	Size (bytes)	Value
4	`CertChainHash`	H	Hash of the certificate chain. It is used for authentication. This field is big endian. The Requester can use this value 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`, this field is computed as the hash(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.

Offset	Field	Size (bytes)	Value
38 + 2H + `OpaqueLength`	`Signature`	S	S is the size of the asymmetric-signing algorithm output that the Responder selected through the last `ALGORITHMS` response message to the Requester. The CHALLENGE_AUTH signature generation and CHALLENGE_AUTH signature verification clauses, respectively, define the signature generation and verification processes.

4.9.2.6 CHALLENGE_AUTH signature generation

To complete the `CHALLENGE_AUTH` signature generation process, the Responder shall complete these steps:

1. The Responder shall construct M1 and the Requester shall construct M2 message transcripts. See the Request ordering and message transcript computation rules for M1/M2 table.

   where:

   `Concatenate()` is the standard concatenation function that is performed only after a successful completion response on the entire request and response contents.

   - If a response contains `ErrorCode=ResponseNotReady`

     Concatenation function is performed on the contents of both the original request and the response received during `RESPOND_IF_READY`.

   - If a response contains `ErrorCode~=ResponseNotReady`

     No concatenation function is performed on the contents of both the original request and response.

2. The Responder shall generate:

   ```
   Signature = Sign(SK, Hash(M1));
   ```

   where:

   - `Sign`

     Asymmetric signing algorithm that the Responder selected through the last `ALGORITHMS` response message that the Responder sent.

The Successful ALGORITHMS response message table describes the `BaseAsymSel` and `ExtAsymSel` fields.

- `SK`

  Private key associated with the leaf certificate of the Responder in `slot=Param1` of the `CHALLENGE` request message.

- `Hash`

  Hashing algorithm the Responder selected through the last `ALGORITHMS` response message that the Responder sent.

  The Successful ALGORITHMS response message table describes the `BaseHashSel` and `ExtHashSel` fields.

If the signing algorithm first hashes the message before generating the signature, the signing algorithm's hashing step shall be skipped.

4.9.2.7 CHALLENGE_AUTH signature verification

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.

To complete the `CHALLENGE_AUTH` signature verification process, the Requester shall complete this step:

1. The Requester shall perform:

```
Verify(PK, Hash(M2), Signature);
```

where:

- `Verify`

  Asymmetric verification algorithm that the Responder selected through the last `ALGORITHMS` response message that the Requester received.

  The Successful ALGORITHMS response message table describes the `BaseAsymSel` and `ExtAsymSel` fields.

- `PK`

  Public key associated with the leaf certificate of the Responder with `slot=Param1` of the `CHALLENGE` request message.

- `Hash`

243 Hashing algorithm the Responder selected through the last sent `ALGORITHMS` response message as received by the Requester.

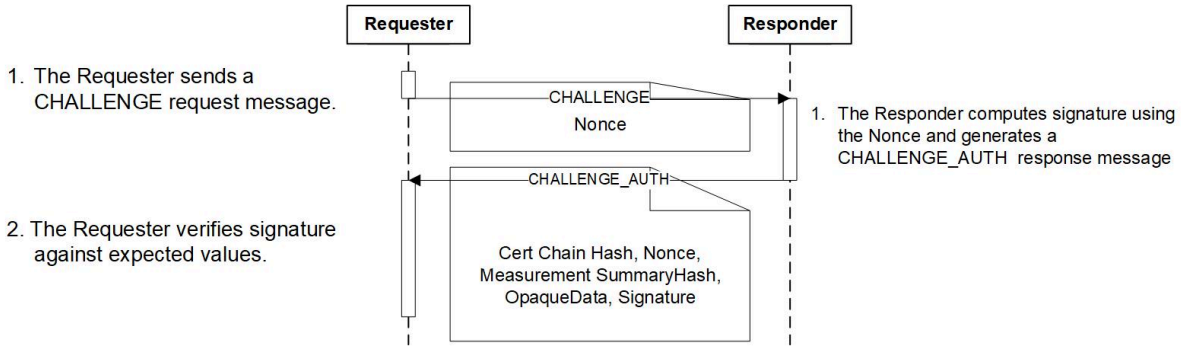
244 The Successful ALGORITHMS response message table describes the `BaseHashSel` and `ExtHashSel` fields.

245 If the verification algorithm first hashes the message before generating the signature, the verification algorithm's hashing step shall be skipped.

246 The Responder authentication: Runtime challenge-response flow shows the high-level request-response message flow and sequence for the authentication of the Responder for runtime challenge-response.

247 **Responder authentication: Runtime challenge-response flow**

248



249

4.10 Request ordering and message transcript computation rules for M1 and M2

250 The Request ordering and message transcript computation rules for M1/M2 table defines how the message transcript is constructed for M1 and M2, which are used in signature calculation and verification in the `CHALLENGE_AUTH` response message.

251 The possible request orderings after reset are:

- `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`

252 After the Requester receives a successful `CHALLENGE_AUTH` response or the Requester sends a `GET_MEASUREMENTS` request, M1 and M2 shall be set to null. Immediately after reset, M1 and M2 shall be null. If a Requester sends a `GET_VERSION` message, the Requester and Responder shall reset M1 and M2 to null and recommence construction of M1 and M2 starting with the new `GET_VERSION` message.

253 **Request ordering and message transcript computation rules for M1/M2**

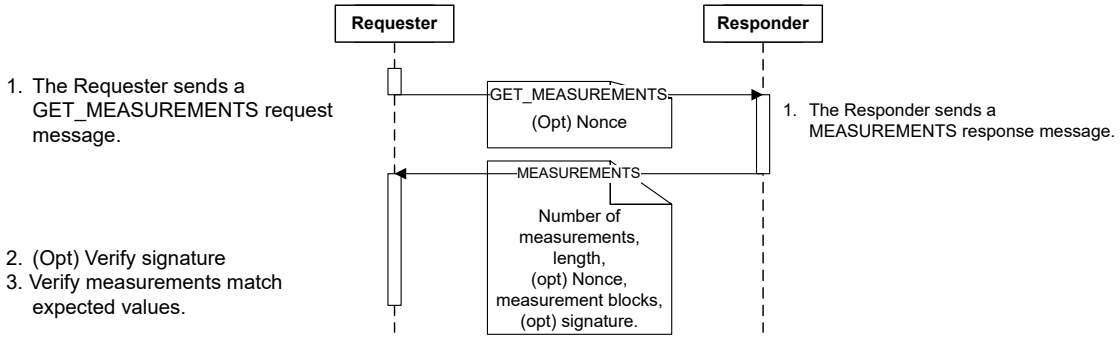
Requests	Implementation requirements	M1/M2=Concatenate (A, B, C)
Reset	NA	M1/M2=null
`GET_VERSION` issue	The Requester may choose to issue this request any time to allow the Requester and Responder to determine an agreed upon Negotiated State. A Requester may detect out of sync condition typically when either the signature verification fails or the Responder provides an unexpected error response.	M1/M2=null
`GET_VERSION` , `GET_CAPABILITIES` , `NEGOTIATE_ALGORITHMS`	Requester shall always issue these requests in this order.	`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 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`
`GET_DIGESTS` , `GET_CERTIFICATE`	Requester shall always issue these requests in this order after `NEGOTIATE_ALGORITHMS` request completion or immediately after reset, if it chose to skip the previous three requests.	`B=Concatenate(GET_DIGESTS, DIGEST, GET_CERTIFICATE, CERTIFICATE)`
`GET_DIGESTS` , `GET_CERTIFICATE`	Requester may choose to skip both requests after a new reset if it can use previously cached response to these requests.	`B=null`
`GET_DIGESTS` , `GET_CERTIFICATE`	Requester may choose to skip `GET_CERTIFICATE` request after a new reset if it can use the previously cached `CERTIFICATE` response.	`B=(GET DIGESTS, DIGEST)`
`CHALLENGE`	Requester shall issue this request to complete security verification of current requests and responses. The Signature bytes of `CHALLENGE_AUTH` shall not be included in C.	`C=(CHALLENGE, CHALLENGE_AUTH\Signature)` . See the CHALLENGE request message table.
`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_DIGESTS` directly after 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`

4.10.1 Firmware and other measurements

This clause describes request messages and response messages associated with endpoint measurement. All request messages in this clause shall be supported by an endpoint that returns `MEAS_CAP=01b` or `MEAS_CAP=10b` in `CAPABILITIES` response.

The Firmware measurement retrieval flow shows the high-level request-response flow and sequence for endpoint 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 resent. The mechanisms employed for resetting the Responder are outside the scope of this specification.

Firmware measurement retrieval flow



4.10.1.1 GET_MEASUREMENTS request message and MEASUREMENTS response message

This request message shall retrieve firmware measurements. If the Responder has set `CHAL_CAP`, a Requester should not send this message until it has received at least one successful `CHALLENGE_AUTH` response message from the responder. The successful `CHALLENGE_AUTH` response may have been received before the last reset.

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`

Offset	Field	Size (bytes)	Value
2	`Param1`	1	Request attributes. See the GET_MEASUREMENTS request attributes table.
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. See the GET_MEASUREMENTS request attributes table.

265 **GET_MEASUREMENTS request attributes**

Bits	Value	Description
`0`	`1`	If the Responder can generate a signature as shown in `CAPABILITIES` message, the value of this bit 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`	Responders that cannot generate a signature as shown in the `CAPABILITIES` message shall use the value of this bit. For Responders that can generate signatures, the value of this bit shall indicate that 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.

Bits	Value	Description
[7:1]	Reserved	Reserved

266 **Successful MEASUREMENTS response message**

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. 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= MeasurementRecordLength	Concatenation of all measurement blocks that correspond to the requested Measurement operation. Measurement block defines the measurement block structure.
8 + L	Nonce	32	The Responder should choose a random value.

Offset	Field	Size (bytes)	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.

4.10.1.2 Measurement block

Each measurement block that the `MEASUREMENTS` response message defines shall contain a four-byte descriptor, offsets 0 through 3, followed by the measurement data that correspond 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	Index. Shall represent the index of the measurement.

Offset	Field	Size (bytes)	Value
1	`MeasurementSpecification`	1	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, as specified in the Measurement field format when MeasurementSpecification field is Bit 0 = DMTF table. All other bits are reserved.
2	`MeasurementSize`	2	Size of `Measurement`, in bytes.
4	`Measurement`	MeasurementSize	The `MeasurementSpecification` defines the format of this field.

271 **4.10.1.3 DMTF specification for the Measurement field of a measurement block**

272 The present clause is the specification for the format of the `Measurement` field in a measurement block when the `MeasurementSpecification` field selects Bit 0=DMTF. This format is specified in Measurement field format when MeasurementSpecification field is Bit 0 = DMTF.

273 **Measurement field format when MeasurementSpecification field is Bit 0 = DMTF**

Offset	Field	Size (bytes)	Value
0	`DMTFSpecMeasurementValueType`	1	Composed of: Bit [7] indicates the representation in `DMTFSpecMeasurementValue`. Bits [6:0] indicate what is being measured by `DMTFSpecMeasurementValue`. These values are set independently and are interpreted as follows: `[7]=0b`. Hash. `[7]=1b`. Raw bit stream. `[6:0]=00h`. Immutable ROM. `[6:0]=0x1`. Mutable firmware. `[6:0]=02h`. Hardware configuration, such as straps, debug modes. `[6:0]=03h`. Firmware configuration, such as, configurable firmware policy. All other values reserved.
1	`DMTFSpecMeasurementValueSize`	2	Size of `DMTFSpecMeasurementValue`, in bytes. When `DMTFSpecMeasurementValueType[7]=0b`, the `DMTFSpecMeasurementValueSize` shall be derived from the measurement hash algorithm that the `ALGORITHM` response message returns.
3	`DMTFSpecMeasurementValue`	`DMTFSpecMeasurementValueSize`	`DMTFSpecMeasurementValueSize` bytes of cryptographic hash or raw bit stream, as indicated in `DMTFSpecMeasurementValueType[7]`.

4.10.1.4 MEASUREMENTS signature generation

To complete the `MEASUREMENTS` signature generation process, the Responder shall complete these steps:

1. The Responder shall construct L1 and the Requester shall construct L2 over their observed messages:

```
L1/L2 = Concatenate(GET_MEASUREMENTS_REQUEST1, MEASUREMENTS_RESPONSE1, ...,
                    GET_MEASUREMENTS_REQUESTn-1, MEASUREMENTS_RESPONSEn-1,
                    GET_MEASUREMENTS_REQUESTn, MEASUREMENTS_RESPONSEn)
```

where:

- `Concatenate()`

  Standard concatenation function.

- `GET_MEASUREMENTS_REQUEST1`

  Entire first `GET_MEASUREMENTS` request message under consideration, where the Requester has not requested a signature on that specific `GET_MEASUREMENTS` request.

- `MEASUREMENTS_RESPONSE1`

  Entire `MEASUREMENTS` response message without the signature bytes that the Responder sent in response to `GET_MEASUREMENTS_REQUEST1`.

- `GET_MEASUREMENTS_REQUESTn-1`

  Entire last consecutive `GET_MEASUREMENTS` request message under consideration, where the Requester has not requested a signature on that specific `GET_MEASUREMENTS` request.

- `MEASUREMENTS_RESPONSEn-1`

  Entire `MEASUREMENTS` response message without the signature bytes that the Responder sent in response to `GET_MEASUREMENTS_REQUESTn-1`.

- `GET_MEASUREMENTS_REQUESTn`

  Entire first `GET_MEASUREMENTS` request message under consideration, where the Requester has requested a signature on that specific `GET_MEASUREMENTS` request.

  n is a number greater than or equal to `1`.

  When *n* equals `1`, the Requester has not made any `GET_MEASUREMENTS` requests without signature prior to issuing a `GET_MEASUREMENTS` request with signature.

- `MEASUREMENTS_RESPONSEn`

  Entire `MEASUREMENTS` response message without the signature bytes that the Responder sent in response to `GET_MEASUREMENTS_REQUESTn`.

Any communication between Requester and Responder other than a `GET_MEASUREMENTS` request or response resets L1/L2 computation to null.

2. The Responder shall generate:

```
Signature = Sign( SK, Hash( L1));
```

where:

- `Sign`

  Asymmetric signing algorithm that the Responder selected through the last `ALGORITHMS` response message that the Responder sent.

  The Successful ALGORITHMS response message table describes the `BaseAsymSel` and `ExtAsymSel` fields.

- `SK`

  Private key associated with the slot 0 leaf certificate of the Responder.

- `Hash`

  Hashing algorithm that the Responder selected through the last `ALGORITHMS` response message that the Responder sent.

  The Successful ALGORITHMS response message table describes the `BaseAsymSel` and `ExtAsymSel` fields.

If the signing algorithm first hashes the message before generating the signature, the signing algorithm's hashing step shall be skipped.

4.10.1.5 MEASUREMENTS signature verification

To complete the `MEASUREMENTS` signature verification process, the Requester shall complete this step:

1. The Requester shall perform:

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

where:

- `PK`

  Public key associated with the slot 0 certificate of the Responder.

`PK` is extracted from the `CERTIFICATES` response.

- `Verify`

  Asymmetric verification algorithm that the Responder selected through the last `ALGORITHMS` response message that the Requester received.

  The Successful ALGORITHMS response message table describes the `BaseAsymSel` and `ExtAsymSel` fields.

- `Hash`

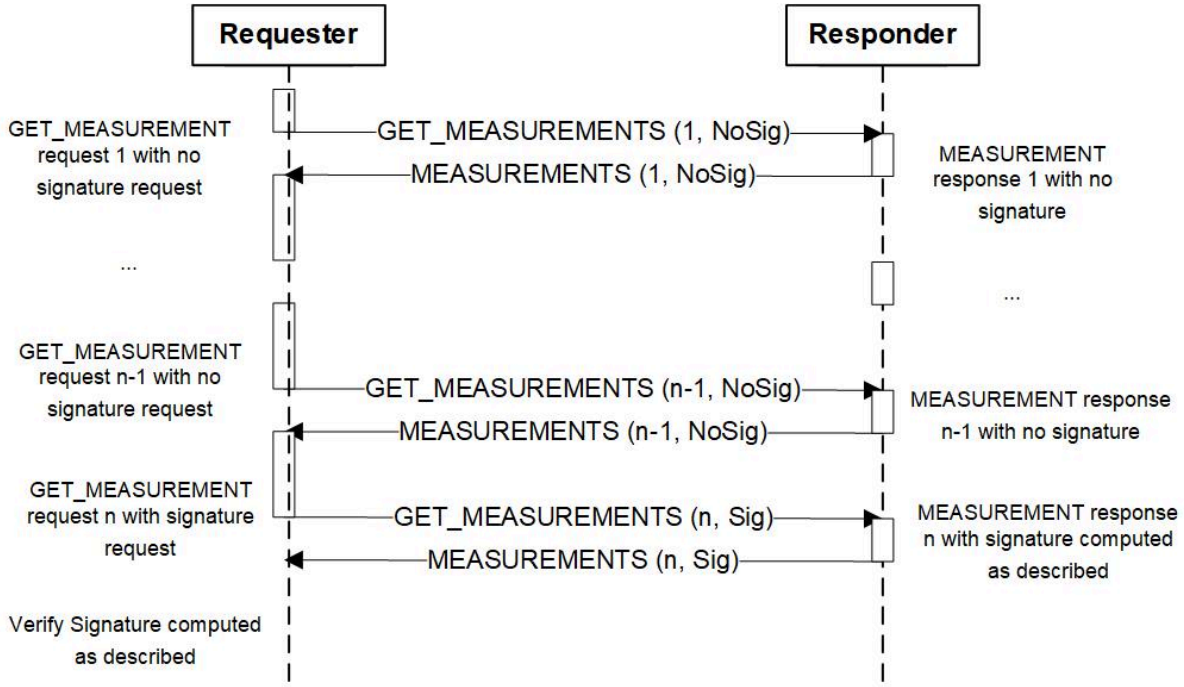
  Hashing algorithm the Responder selected through the last sent `ALGORITHMS` response message that the Requester sent.

  The Successful ALGORITHMS response message table describes the `BaseAsymSel` and `ExtAsymSel` fields.

If the verification algorithm first hashes the message before generating the signature, the verification algorithm's hashing step shall be skipped.

The Measurement signature computation example shows an example of a typical Requester Responder protocol where the Requester issues 0 to *n*-1 `GET_MEASUREMENTS` requests without a signature, followed by a single `GET_MEASUREMENTS` request *n* with a signature.

Measurement signature computation example



4.10.2 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 Error code and error data.
3	`Param2`	1	Error Data. See Error code and error data.
4	`ExtendedErrorData`	0-32	Optional extended data. See Error code and error data.

Error code and error data

Error code	Value	Description	Error data	ExtendedErrorData
Reserved	`0x00`	Reserved	Reserved	Reserved
`InvalidRequest`	`0x01`	One or more request fields are invalid	`0x00`	No extended error data is provided.
Reserved	`0x02`	Reserved	Reserved	Reserved

Error code	Value	Description	Error data	ExtendedErrorData
`Busy`	`0x03`	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.
`UnexpectedRequest`	`0x04`	The Responder received an unexpected request message. For example, `CHALLENGE` before `NEGOTIATE_ALGORITHMS`.	`0x00`	No extended error data is provided.
`Unspecified`	`0x05`	Unspecified error occurred.	`0x00`	No extended error data is provided.
Reserved	`0x06`	Reserved	`0x00`	Reserved
`UnsupportedRequest`	`0x07`	The `RequestResponseCode` in the request message is unsupported.	`RequestResponseCode` in the request message.	No extended error data is provided
Reserved	`0x08` - `0x40`	Reserved	Reserved	Reserved
`MajorVersionMismatch`	`0x41`	Requested SPDM Major Version is not supported.	`0x00`	No extended error data provided.
`ResponseNotReady`	`0x42`	See the RESPOND_IF_READY request message.	`0x00`	See the ResponseNotReady extended error data table.
`RequestResynch`	`0x43`	Responder is requesting Requester to reissue `GET_VERSION` to resynchronize.	`0x00`	No extended error data provided.
Reserved	`0x44` - `0xFE`	Reserved	Reserved.	Reserved
Vendor/Other Standards Defined	`0xFF`	Vendor or Other Standards defined	Shall indicate the registry or standard body using one of the values in the **ID** column in 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.

332 **ResponseNotReady extended error data**

Offset	Field	Size (bytes)	Value
0	`RDTExponent`	1	Exponent expressed in logarithmic (base 2 scale) to calculate `RDT` time in µs after which the Responder can provide successful completion response. For example, the raw value 8 indicates that the Responder will be ready in 2^8=256 µs. Responder should use `RDT` to avoid continuous pinging and issue the `RESPOND_IF_READY` request message 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 `RESPOND_IF_READY` request message.

Offset	Field	Size (bytes)	Value
3	RDTM	1	Multiplier used to compute WT_{Max} in µs 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.

333 **Registry or standards body ID**

334 For algorithm encoding in extended algorithm fields, unless otherwise specified, consult the respective registry or standards body.

ID	Vendor ID length (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	VendorID is identified by using TCG Vendor ID Registry. For extended algorithms, see TCG Algorithm Registry.
0x2	2	USB	VendorID is identified by using the vendor ID assigned by USB.
0x3	2	PCI-SIG	VendorID is identified using PCI-SIG Vendor ID.

ID	Vendor ID length (bytes)	Registry or standards body name	Description
`0x4`	4	IANA	The Private Enterprise Number (PEN) assigned by the Internet Assigned Numbers Authority (IANA) identifies the vendor.
`0x5`	4	HDBaseT	`VendorID` is identified by using HDBaseT HDCD entity.
`0x6`	2	MIPI	The Manufacturer ID assigned by MIPI identifies the vendor.

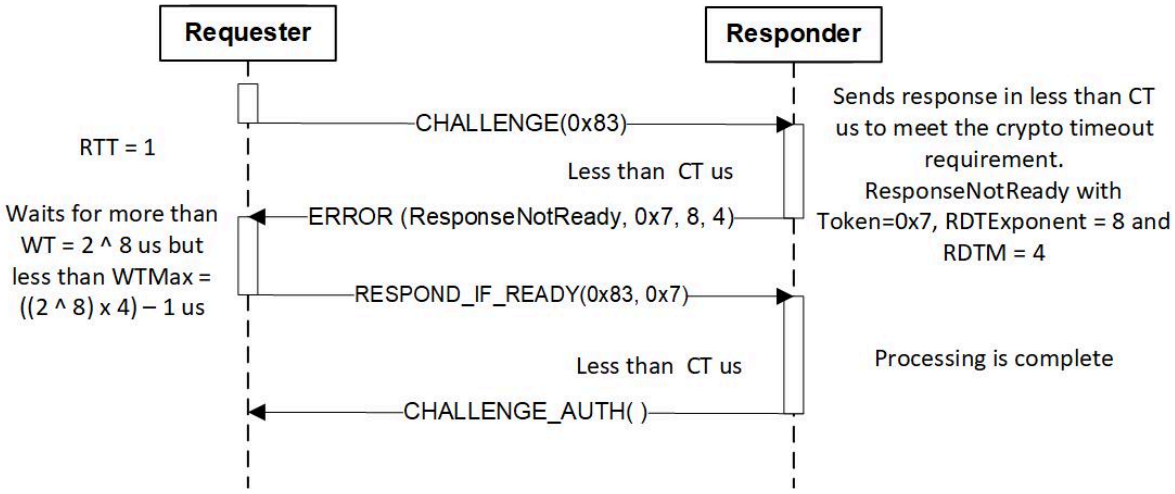
335 **ExtendedErrorData format definition for vendor or other standards-defined ERROR response message**

Byte offset	Length	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 the Registry or standards body ID table describes, of the corresponding registry or standard body name. If the `ERROR` is defined by a registry or a standard, this field shall be zero (`0`), which also indicates that the `VendorID` field is not present. The `Error Data` field in the `ERROR` message indicates the registry or standards body name, such as `Param2`, and is one of the values in the **ID** column in the Registry or standards body ID table.

Byte offset	Length	Field name	Description
1	Len	VendorID	The value of this field shall indicate the Vendor ID, as assigned by the registry or standards body. The Registry or standards body ID table describes the length of this field. Shall be in little endian format. The registry or standards body name in the `ERROR` is indicated in the `ErrorData` field, such as `Param2`, and is one of the values in the **ID** column in the Registry or standards body ID table.
1 + Len	Variable	OpaqueErrorData	Defined by the vendor or other standards.

4.10.3 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.



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`

Offset	Field	Size (bytes)	Value
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.

4.10.4 VENDOR_DEFINED_REQUEST request message

A Requester intending to define a unique request to meet its need can use this request message. The VENDOR_DEFINED_REQUEST request message table defines the format.

The Requester should send this request message only after sending `GET_VERSION`, `GET_CAPABILITIES` and `NEGOTIATE_ALGORITHMS` request sequence.

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

VENDOR_DEFINED_REQUEST request message

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0xFE=VENDOR_DEFINED_REQUEST`
2	`Reserved`	1	Reserved
3	`Reserved`	1	Reserved
4	`StandardID`	2	Shall indicate the registry or standards body by using one of the values in the **ID** column in the Registry or standards body ID table.

Offset	Field	Size (bytes)	Value
6	`Len`	1	Length of the `Vendor ID` field. If the `VendorDefinedRequest` is standard defined, Len shall be `0`. If the `VendorDefinedRequest` is vendor-defined, Len shall equal `Vendor ID Len`, as the Registry or standards body ID table describes.
7	`VendorID`	Len	Vendor ID, as assigned by the registry or standards body. Shall be in little endian format.
7 + Len	`ReqLength`	2	Length of the `VendorDefinedReqPayload`.
7 + Len + 2	`VendorDefinedReqPayload`	ReqLength	The standard or vendor shall use this field to send the request payload.

346 Other DMTF specifications may define VENDOR_DEFINED_REQUEST with `StandardID` set to 0. See `VendorDefinedReqPayload` and `VendorDefinedRespPayload` defined by DMTF specifications for more information.

347 4.10.5 VENDOR_DEFINED_RESPONSE response message

348 A Responder can use this response message in response to `VENDOR_DEFINED_REQUEST`. The VENDOR_DEFINED_RESPONSE response message table defines the format.

349 The VENDOR_DEFINED_RESPONSE response message table shows the `VENDOR_DEFINED_RESPONSE` response message format.

350 **VENDOR_DEFINED_RESPONSE response message**

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0x7E=VENDOR_DEFINED_RESPONSE`
2	`Reserved`	1	Reserved
3	`Reserved`	1	Reserved

Offset	Field	Size (bytes)	Value
4	`StandardID`	2	Shall indicate the registry or standard body using one of the values in the **ID** column in the Registry or standards body ID table.
6	`Len`	1	Length of the `Vendor ID` field. If the `VendorDefinedRequest` is standards-defined, length shall be `0`. If the `VendorDefinedRequest` is vendor-defined, length shall equal `Vendor ID Len`, as the Registry or standards body ID table describes.
7	`VendorID`	Len	Shall indicate the Vendor ID, as assigned by the registry or standards body. Shall be in little endian format.
7 + Len	`RespLength`	2	Length of the `VendorDefinedRespPayload`
7 + Len + 2	`VendorDefinedRespPayload`	RespLength	Standard or vendor shall use this value to send the response payload.

351

4.10.6 `VendorDefinedReqPayload` and `VendorDefinedRespPayload` defined by DMTF specifications

352 Other DMTF specifications may define `VENDOR_DEFINED_REQUEST` and `VENDOR_DEFINED_RESPONSE` messages with `StandardID` set to 0 ("DMTF", as defined in the Registry or standards body ID table) and `Len` set to 0. In this case, `VENDOR_DEFINED_REQUEST` and `VENDOR_DEFINED_RESPONSE` messages shall specify the underlying DMTF specification that defines them. A DMTF specification which defines the data model of `VendorDefinedReqPayload` for `VENDOR_DEFINED_REQUEST` and the data model of `VendorDefinedRespPayload` for `VENDOR_DEFINED_RESPONSE` shall follow the Format of `VendorDefinedReqPayload` and `VendorDefinedRespPayload` when `StandardID` is DMTF table.

353 **Format of `VendorDefinedReqPayload` and `VendorDefinedRespPayload` when `StandardID` is DMTF**

Byte offset	Field	Size (bytes)	Description
0	DSPNumber	2	Shall be the DMTF specification's DSP number in a 16-bit integer. For example, DSP0287 would use 0x011F.

Byte offset	Field	Size (bytes)	Description
2	DSPVersion	2	Shall be the version number of the DMTF specification whose DSP number is populated in the `DSPNumber` field. The format of the version number shall follow the VersionNumberEntry definition table.
4	VendorPayload	Variable	Shall be the actual payload data defined by the DMTF specification whose DSP number is populated in the `DSPNumber` field.

354

5 ANNEX A (normative) Leaf certificate example

355 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
```

6 ANNEX B (informative) Change log

6.1 Version 1.0.0 (2019-10-16)

- Initial Release

6.2 Version 1.0.1 (2021-02-16)

- Minor typographical fixes
- USB Authentication Specification 1.0 link updated
- Tables are no longer numbered. They are now named.
- Fix internal document links in SPDM response codes table.
- Added sentence to paragraph 92 to clarify on the potential to skip messages after a reset.
- Removed text at paragraph 156.
- Certificate digests in `DIGEST` calculation clarified at paragraph 190.
- Format of certificate in `CertChain` parameter of `CERTIFICATE` message clarified.
- Validity period of X.509v3 certificate clarified in Required Fields
- `Subject Alternative Name otherName` field in Optional fields references DMTF OID section.
- `DMTFOtherName` definition changed to properly meet ASN.1 syntax.
- Text in figures are now searchable.
- Fix improper reference in `DMTFSpecMeasurementValue` field in "Measurement field format when MeasurementSpecification field is Bit 0 = DMTF" table.
- Corrected example of a leaf certificate in Annex A.
- Minor edits to figures for clarity.

6.3 Version 1.0.2 (2023-10-08)

- Removed `PSS_CAP` from the `CAPABILITIES` response (corrects error introduced in DSP0274 1.0.1).
- Fixed Leaf Certificate X.509 version field description (corrects error introduced in DSP0274 1.0.1).
- Clarified that messages are only hashed once before being signed and verified.
- Added clause that sizes and lengths are in units of bytes.
- Fix broken references.
- Clarified in Registry or standards body ID that the registry specifies the value used for the `VendorID` field.
- Clarified that `ERROR` is only allowed in response to `GET_VERSION` in cases explicitly defined in this specification.
- Added `VendorDefinedReqPayload` and `VendorDefinedRespPayload` defined by DMTF specifications clauses.

- Added normative information in the Flag fields definitions table.
- Stated that all figures are informative unless otherwise specified explicitly.

6.4 Version 1.0.3 (2026-04-03)

- Minor editorial updates
- Correct `ReqLength` to `RespLength` in VENDOR_DEFINED_RESPONSE response message.
- Clarified whether algorithms are used for signature generation or signature verification.
- Clarified that recommendation for `CHALLENGE_AUTH` before `GET_MEASUREMENTS` is conditional on Responder's `CHAL_CAP` value.
- Added states and hardware measurements to the scope in Abstract
- Provide guidance that validity periods in certificate chains should all be similar to the Leaf certificate or have another update strategy.
- Clarified that the salt length should be the same as the output length of the selected hash function.
- Clarified encoding and format of certificates.

7 Bibliography

DMTF DSP4014, *DMTF Process for Working Bodies 2.6*.