Document Identifier: DSP0274

Date: 2019-10-30

Version: 1.1.0a

Security Protocol and Data Model (SPDM) Specification

Information for Work-in-Progress version:

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 This document is available for public review and comment until superseded.

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

Supersedes: 0.99.0a

Document Class: Normative

Document Status: Work In Progress

Document Language: en-US

CONTENTS

Figures

Tables

1 Foreword

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

2 Acknowledgments

The DMTF acknowledges these individuals' contributions to this document:

Contributors:

- Richelle Ahlvers — Broadcom Inc.
- Lee Ballard — Dell Technologies
- Patrick Caporale — Lenovo
- Yu-Yuan Chen — Intel Corporation
- Nigel Edwards — Hewlett Packard Enterprise
- Daniil Egranov — Arm Limited
- 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 Mathews — 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, and sequences for performing message exchanges between *devices* over a variety of transport and physical media. The description of message exchanges includes *authentication* of hardware identities and measurement for firmware identities. The SPDM enables efficient access to low-level security capabilities and operations. Other mechanisms, including non-PMCI- and DMTF-defined mechanisms, can use the SPDM.

3.1 Scope

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

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.

3.2 Normative references

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.

- *ISO/IEC Directives, Part 2, Principles and rules for the structure and drafting of ISO and IEC documents - 2018 (8th edition)*
- 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 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*, https://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype
- IETF RFC5234, *Augmented BNF for Syntax Specifications: ABNF*, January 2008
- *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, 08/2015
- **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
- **Transport Layer Security 1.3**
  - TLS 1.3 RFC 8446

3.3 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.

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

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

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

This specification uses these terms:

Term	Definition
application data	Data that is transferred over a secured session whose definition and format is outside the scope of this specification. Application layer, in general, is the layer above SPDM.
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*.
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.
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 DMTF, the 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 protocol.
intermediate certificate	Certificate that is neither a root certificate nor a leaf certificate.
leaf certificate	Last certificate in a certificate chain.
message	See SPDM message.

Term	Definition
message body	Portion of a SPDM message that carries additional data.
message originator	Original transmitter, or source, of a SPDM message.
message transcript	The concatenation of a sequence of messages in the order in which they are sent and received by an endpoint. 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.
Platform Management Component Intercommunications (PMCI)	Working group under the Distributed Management Task Force that defines standardized communication protocols, low-level data models, and transport definitions that support communications with and between management controllers and management devices that form a platform management subsystem within a managed computer system.
Requester	Original transmitter, or source, of a SPDM request message. It is also the ultimate receiver, or destination, of a SPDM response message.
Responder	Ultimate receiver, or destination, of a SPDM request message. It is also the original transmitter, or source of a SPDM response message.
root certificate	First certificate in a certificate chain, which is self-signed.
session keys	Session Keys are any secrets, derived cryptographic keys or any cryptographic information bound to the session.

Term	Definition
SPDM message	Unit of communication in SPDM communications.
SPDM message payload	Portion of the message body of a 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 a 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: https://en.wikipedia.org/wiki/Trusted_computing_base

3.4 Symbols and abbreviated terms

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

The following additional abbreviations are used in this document.

Abbreviation	Definition
CA	*certificate authority*
MAC	Message Authentication Code
DMTF	Formerly the *Distributed Management Task Force*
MSB	*most significant byte*
PMCI	*Platform Management Component Intercommunications*
SPDM	Security Protocol and Data Model
TCB	*trusted computing base*
AEAD	Authenticated Encryption with Associated Data

3.5 Conventions

The following conventions apply to all SPDM specifications.

3.5.1 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.

3.5.2 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.

3.5.3 Byte ordering

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

3.5.4 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.

3.5.5 Version encoding

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

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

EXAMPLE:

Version 3.7 → `0x37`

Version 1.0 → `0x10`

Version 1.2 → `0x12`

An *endpoint* that supports Version 1.2 can interoperate with an older endpoint that supports Version 1.0 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.

3.5.6 Notations

SPDM specifications use the following 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.
`[M:N]`	A range of bit offsets where M is greater than or equal to N. 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.

Notation	Description
`N+`	This indicates a variable length byte range that starts at byte offset N.

4 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 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.
- Construct a secure communication channel for the transmission or reception of application data.

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.

4.1 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.

4.2 Identity authentication

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

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

- **Identity provisioning**

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

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

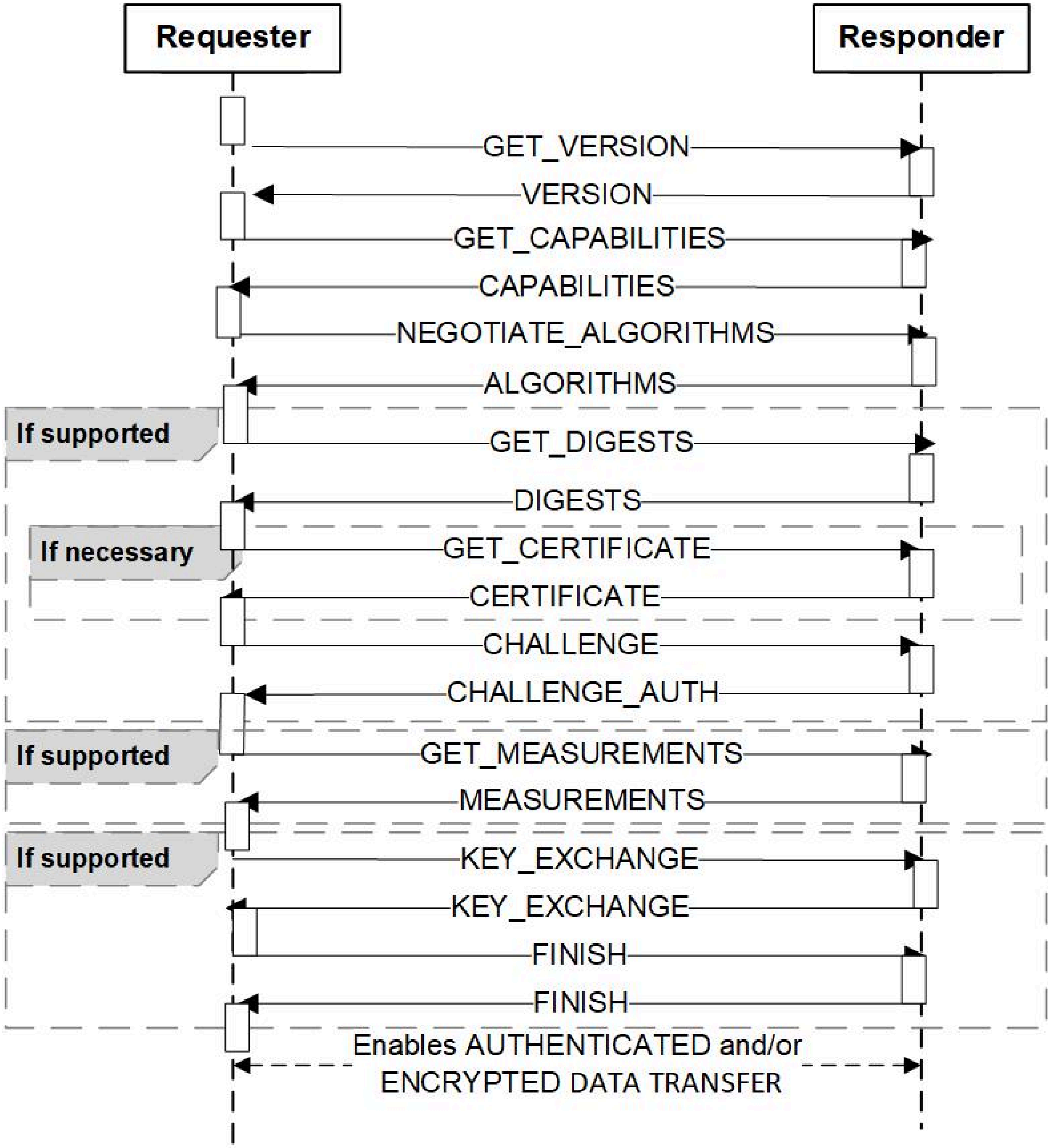
4.3 Firmware and configuration measurement

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.

5 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 a SPDM response message as defined in this specification unless otherwise stated in this specification.

The SPDM messaging protocol flow depicts the high-level request-response flow diagram for SPDM. An endpoint that acts as the *Requester* sends a SPDM request message to another endpoint that acts as the *Responder*, and the Responder returns a SPDM response message to the Requester.



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

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

5.1 Generic SPDM message format

The following table defines the fields that constitute a generic SPDM message, including the message header and payload.

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 the SPDM request codes table and the SPDM response codes table. `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.

5.2 SPDM request codes

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

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

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

SPDM request codes

Request	Code value	Implementation requirement	Message format
`GET_DIGESTS`	`0x81`	Optional	See the GET_DIGESTS request message table.
`GET_CERTIFICATE`	`0x82`	Optional	See the GET_CERTIFICATE request message table.
`CHALLENGE`	`0x83`	Optional	See the CHALLENGE request message table.
`GET_VERSION`	`0x84`	Required	See the GET_VERSION request message table.
`GET_MEASUREMENTS`	`0xE0`	Optional	See the GET_MEASUREMENTS request message table.
`GET_CAPABILITIES`	`0xE1`	Required	See the GET_CAPABILITIES request message table.
`NEGOTIATE_ALGORITHMS`	`0xE3`	Required	See the NEGOTIATE_ALGORITHMS request message table.
`KEY_EXCHANGE`	`0xE4`	Optional	See the KEY_EXCHANGE request message table.
`FINISH`	`0xE5`	Optional	See the FINISH request message table.
`PSK_BASED_EXCHANGE`	`0xE6`	Optional	See the [PSK_BASED_EXCHANGE request message](#psk-based-exchange req) table.
`PSK_BASED_FINISH`	`0xE7`	Optional	See the [PSK_BASED_FINISH request message](#psk-based-finish req) table.
`HEARTBEAT`	`0xE8`	Optional	See the [HEARTBEAT request message](#heartbeat req) table.
`KEY_UPDATE`	`0xE9`	Optional	See the [KEY_UPDATE request message](#key-update req) table.
`GET_ENCAPSULATED_REQUEST`	`0xEA`	Optional	See the [GET_ENCAPSULATED_REQUEST request message](#get-encapsulated-request req) table.
`DELIVER_ENCAPSULATED_RESPONSE`	`0xEB`	Optional	See the [DELIVER_ENCAPSULATED_RESPONSE request message](#deliver-encapsulated-response req) table.
`END_SESSION`	`0xEC`	Optional	See the [END_SESSION request message](#end-session req) 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.

Request	Code value	Implementation requirement	Message format
Reserved	`0x80` , `0x85` - `0xDF` , `0xE2` , `0xED` - `0xFD`	SPDM implementations compatible with this version shall not use the reserved request codes.	

5.3 SPDM response codes

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

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

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

SPDM response codes

Response	Value	Implementation requirement	Message format
`DIGESTS`	`0x01`	Optional	See the GET_DIGESTS request message table.
`CERTIFICATE`	`0x02`	Optional	See the GET_CERTIFICATE request message table.
`CHALLENGE_AUTH`	`0x03`	Optional	See the CHALLENGE request message table.
`VERSION`	`0x04`	Required	See the Successful VERSION response message table.
`MEASUREMENTS`	`0x60`	optional	See the GET_MEASUREMENTS request message table.
`CAPABILITIES`	`0x61`	Required	See the Successful CAPABILITIES response message table.
`ALGORITHMS`	`0x63`	Required	See the Successful ALGORITHMS response message table.
`KEY_EXCHANGE`	`0x64`	Optional	See the KEY_EXCHANGE response message table.
`FINISH`	`0x65`	Optional	See the FINISH response message table.
`PSK_BASED_EXCHANGE`	`0x66`	Optional	See the [PSK_BASED_EXCHANGE response message](#psk-based-exchange resp) table.

Response	Value	Implementation requirement	Message format
`PSK_BASED_FINISH`	`0x67`	Optional	See the [PSK_BASED_FINISH response message](#psk-based-finish resp) table.
`HEARTBEAT`	`0x68`	Optional	See the [HEARTBEAT response message](#heartbeat resp) table.
`KEY_UPDATE`	`0x69`	Optional	See the [KEY_UPDATE response message](#key-update resp) table.
`ENCAPSULATED_REQUEST`	`0x6A`	Optional	See the [ENCAPSULATED_REQUEST response message](#encapsulated-request resp) table.
`ENCAPSULATED_RESPONSE_ACK`	`0x6B`	Optional	See the [ENCAPSULATED_RESPONSE_ACK response message](#encapsulated-response-ack resp) table.
`END_SESSION`	`0x6C`	Optional	See the [END_SESSION response message](#end-session resp) 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` , `0x6D` - `0x7D`	SPDM implementations compatible with this version shall not use the reserved response codes.	

5.4 SPDM Request and Response Code Issuance Allowance

The SPDM Request and Response Validity Table describes when a request can be issued and by who can issue them.

SPDM Request and Response Validity Table

Type	Session Validity	Allowed Issuer
`FINISH` Request	Session Handshake Only	Requester
`PSK_BASED_FINISH` Request	Session Handshake Only	Requester
`HEARTBEAT` Request	Application Phase	Requester or Responder
`KEY_UPDATE` Request	Application Phase	Requester or Responder
`ERROR` Response	Sessionless	Responder
`ERROR` Response	Session Handshake or Application Phase	Requester or Responder

Type	Session Validity	Allowed Issuer
`GET_ENCAPSULATED_REQUEST` Request	Sessionless or Session Handshake	Requester
`DELIVER_ENCAPSULATED_RESPONSE` Request	Sessionless or Session Handshake	Requester
`VENDOR_DEFINED_REQUEST` Request	Sessionless or Application Phase	Requester
All Others	Sessionless Only	Requester

For `ERROR` response in Session Handshake or Application Phase of a session, the Requester is only allowed in certain situations to send the ERROR response.

Session is defined as outside of a session. For details on Session, see Session clause and subclauses.

5.5 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.

5.6 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.

5.7 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.

6 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.

6.1 Timing measurements

A Requester shall measure timing parameters, applicable to it, from the end of a successful transmission of a 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.

6.2 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.	us	Worst case round-trip transport timing. The maximum value shall be the worst case total time for the complete transmission and delivery of a SPDM message round trip at the transport layer(s). The actual value for this parameter is transport- or media-specific. Both the actual value and how an endpoint obtains this value are outside the scope of this specification.
ST1	Responder	100,000	us	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	us	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	$2^{CTExponent}$	us	The `CAPABILITIES` message reports the cryptographic timeout, in microseconds. `CTExponent` is reported in `GET_CAPABILITIES` . 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	us	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	$2^{RDTExponent}$	us	Recommended delay, in microseconds that the Responder needs to complete the requested cryptographic operation. When the Responder is unable to complete cryptographic processing response within the `CT` time, it shall provide `RDTExponent` as part of the `ERROR` response. See the ResponseNotReady extended error data table for the `RDTExponent` value. For details, see `ErrorCode=ResponseNotReady` in the ResponseNotReady extended error data table.
WT	Requester	RDT	us	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	us	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 `WT` Max. 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.

Timing parameter	Ownership	Value	Units	Description
HeartbeatPeriod	Requester and Responder	Variable	s	See HEARTBEAT Request and HEARTBEAT Response for detail.

7 SPDM messages

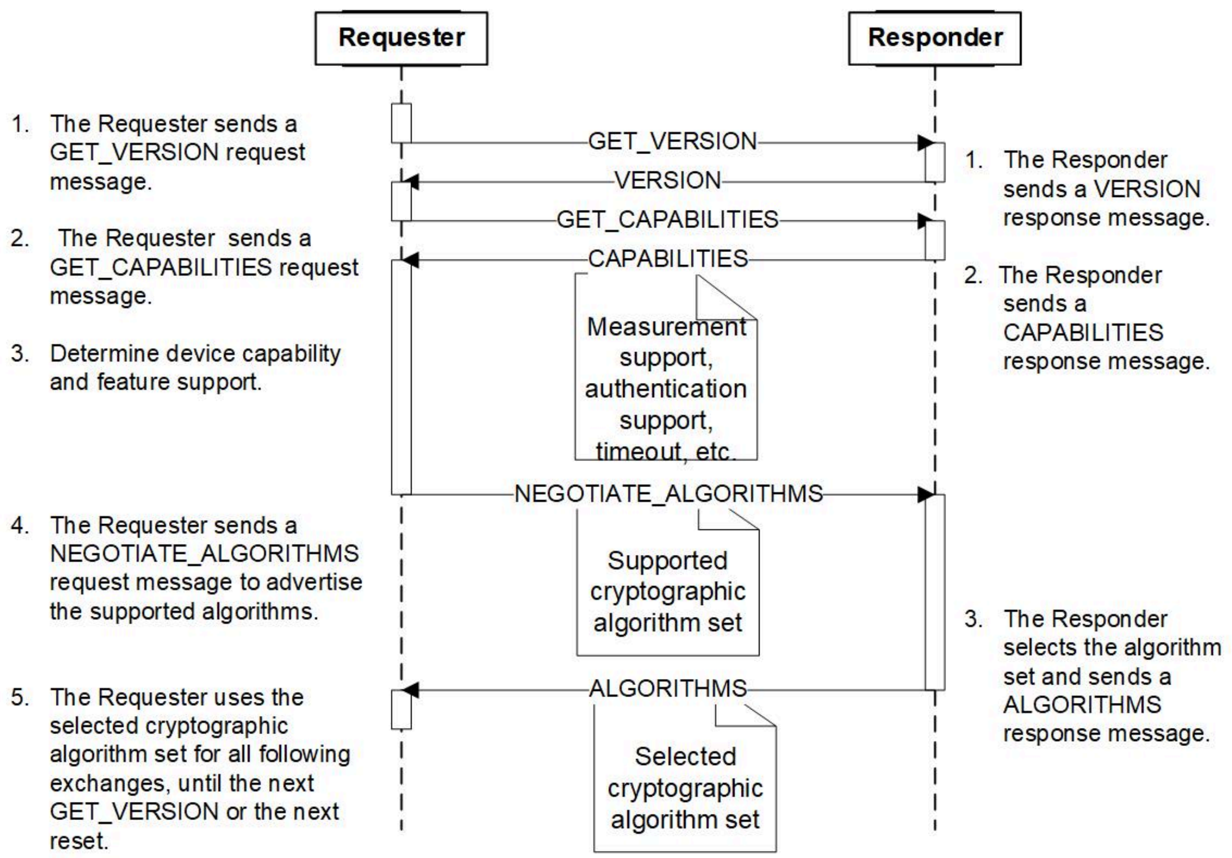
SPDM messages can be divided into the following categories, supporting different aspects of security exchanges between a Requester and Responder:

- Capability discovery and negotiation
- Responder identity authentication
- Firmware measurements

7.1 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:



7.2 GET_VERSION request message and VERSION response message

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

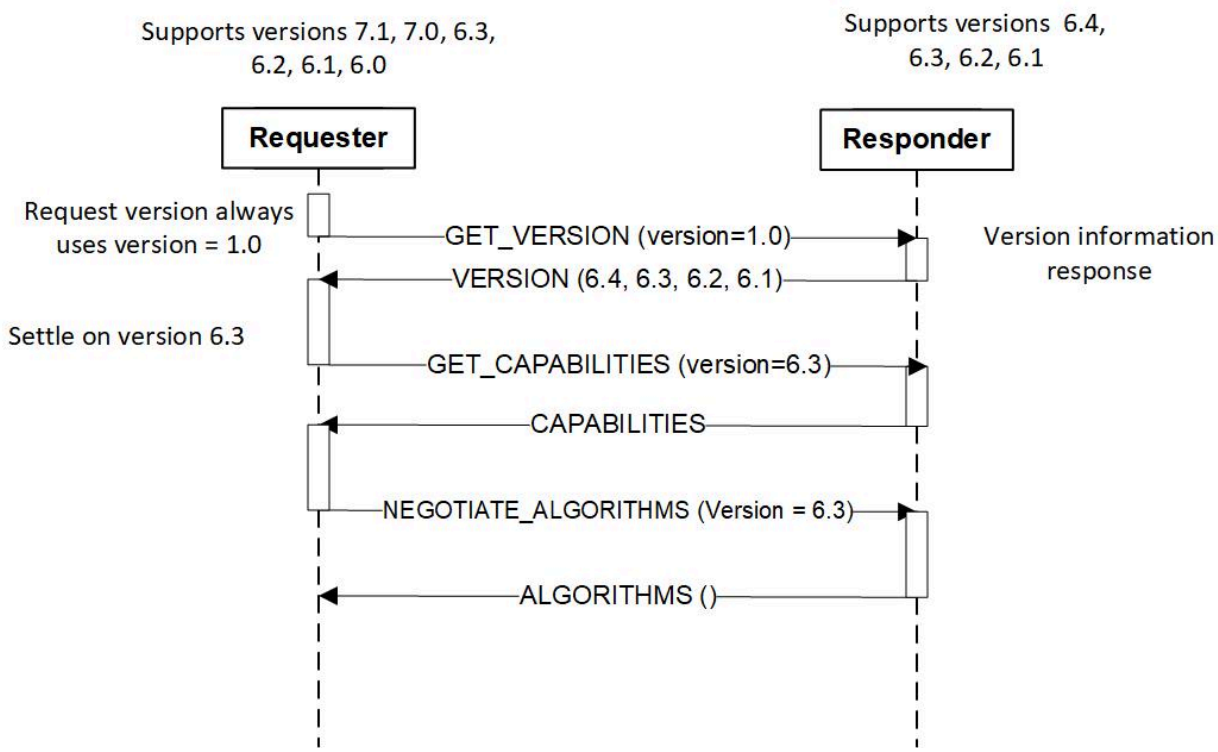
In all future SPDM versions, the `GET_VERSION` and `VERSION` response messages will be backward compatible with all previous versions.

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 `ErrorCode=ResponseNotReady`.

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.



GET_VERSION request message

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

Successful VERSION response message

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0x04=VERSION`
2	`Param1`	1	Reserved
3	`Param2`	1	Reserved

Offset	Field	Size (bytes)	Value
4	`Reserved`	1	Reserved
5	`VersionNumberEntryCount`	1	Number of version entries present in this table (=n).
6	`VersionNumberEntry1:n`	2 x n	16-bit version entry. See the GET_VERSION request message table.

VersionNumberEntry definition

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

7.3 GET_CAPABILITIES request message and CAPABILITIES response message

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

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

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

The Flag fields definitions table shows the flag fields definitions.

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

GET_CAPABILITIES request message

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

Offset	Field	Size (bytes)	Value
1	`RequestResponseCode`	1	`0xE1=GET_CAPABILITIES`
2	`Param1`	1	Reserved
3	`Param2`	1	Reserved
4	`Reserved`	1	Reserved
5	`CTExponent`	1	Shall be 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 (us). For example, if `CTExponent` is 10, `CT` is 2^{10}=1024 us.
6	`Reserved`	2	Reserved
8	`Flags`	4	See the Requester Flag fields definitions table.

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 used to calculate `CT`. See the Timing specification for SPDM messages table. The equation for `CT` shall be 2^{CT} microseconds (us). For example, if `CTExponent` is 10, `CT` is 2^{10}=1024 us.
6	`Reserved`	2	Reserved
8	`Flags`	4	See the Responder Flag fields definitions table.

Requester Flag fields definitions

Byte	Bit	Field	Value
0	`0`	`Reserved`	Reserved
0	`1`	`CERT_CAP`	If set, Requester supports `DIGESTS` and `CERTIFICATE` response messages.
0	`2`	`CHAL_CAP`	If set, Requester supports `CHALLENGE_AUTH` response message.
0	`4:3`	`MEAS_CAP`	The Requester's `MEASUREMENT` response capabilities. • `00b` . The Requester does not support `MEASUREMENTS` response capabilities. • `01b` . The Requester supports `MEASUREMENTS` response but cannot perform signature generation. • `10b` . The Requester supports `MEASUREMENTS` response and can generate signatures. • `11b` . Reserved
0	`5`	`MEAS_FRESH_CAP`	• `0` . As part of `MEASUREMENTS` response message, the Requester may return `MEASUREMENTS` that were computed during the last Requester's reset. • `1` . The Requester 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`	`PSK_CAP`	Requester's `PreSharedKey` capabilities. • `00b` . Requester does not support `PreSharedKey` capabilities. • `01b` . Requester supports `PreSharedKey` • `10b` and `11b` . Reserved
1	`0`	`MUT_AUTH_CAP`	If set, Requester supports mutual authentication
1	`1`	`ENCRPT_CAP`	If set, Requester supports message encryption
1	`2`	`MAC_CAP`	If set, Requester supports message authentication
1	`7:3`	`Reserved`	Reserved
2	`7:0`	`Reserved`	Reserved
3	`7:0`	`Reserved`	Reserved

Responder Flag fields definitions

Byte	Bit	Field	Value
0	`0`	`CACHE_CAP`	If set, the Responder supports the ability to cache the *Negotiated State* across a reset. This allows the Requester to skip reissuing the `GET_VERSION` , `GET_CAPABILITIES` and `NEGOTIATE_ALGORITHMS` requests after a reset. The Responder shall cache the selected cryptographic algorithms as one of the parameters of the Negotiated State. If the Requester chooses to skip issuing these requests after the reset, the Requester shall also cache the same selected cryptographic algorithms.
0	`1`	`CERT_CAP`	If set, Responder supports `DIGESTS` and `CERTIFICATE` response messages.

Byte	Bit	Field	Value
0	`2`	`CHAL_CAP`	If set, Responder supports `CHALLENGE_AUTH` response message.
0	`4:3`	`MEAS_CAP`	The Responder's `MEASUREMENT` response capabilities. • `00b` . The Responder does not support `MEASUREMENTS` response capabilities. • `01b` . The Responder supports `MEASUREMENTS` response but cannot perform signature generation. • `10b` . The Responder supports `MEASUREMENTS` response and can generate signatures. • `11b` . Reserved
0	`5`	`MEAS_FRESH_CAP`	• `0` . As part of `MEASUREMENTS` response message, the Responder may return `MEASUREMENTS` that were computed during the last Responder's reset. • `1` . The Responder can recompute all `MEASUREMENTS` in a manner that is transparent to the rest of the system and shall always return fresh `MEASUREMENTS` as part of `MEASUREMENTS` response message.
0	`7:6`	`PSK_CAP`	Responder's `PreSharedKey` capabilities. • `00b` . Responder does not support `PreSharedKey` capabilities. • `01b` . Responder supports `PreSharedKey` but does not provide responder_context for session key derivation. • `10b` . Responder supports `PreSharedKey` and provides responder_context for session key derivation. • `11b` . Reserved
1	`0`	`MUT_AUTH_CAP`	If set, Responder supports mutual authentication
1	`1`	`ENCRYPT_CAP`	If set, Responder supports message encryption
1	`2`	`MAC_CAP`	If set, Responder supports message authentication
1	`7:3`	`Reserved`	Reserved
2	`7:0`	`Reserved`	Reserved
3	`7:0`	`Reserved`	Reserved

7.4 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 with exactly one asymmetric algorithm and exactly one hashing algorithm.

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
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
8	*BaseAsymAlgo*	4	Bit mask listing Requester-supported SPDM-enumerated asymmetric key signature algorithms for the purposes of signature verification. • Byte 0 Bit 0. TPM_ALG_RSASSA_2048 • Byte 0 Bit 1. TPM_ALG_RSAPSS_2048 • Byte 0 Bit 2. TPM_ALG_RSASSA_3072 • Byte 0 Bit 3. TPM_ALG_RSAPSS_3072 • Byte 0 Bit 4. TPM_ALG_ECDSA_ECC_NIST_P256 • Byte 0 Bit 5. TPM_ALG_RSASSA_4096 • Byte 0 Bit 6. TPM_ALG_RSAPSS_4096 • Byte 0 Bit 7. TPM_ALG_ECDSA_ECC_NIST_P384 • Byte 1 Bit 0. TPM_ALG_ECDSA_ECC_NIST_P521 All other values reserved.

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	*DHENamedGroup*	4	Bit mask listing Requester-supported SPDM-enumerated cryptographic Diffie-Hellman algorithms. • Byte 0 Bit 0. ffdhe2048 • Byte 0 Bit 1. ffdhe3072 • Byte 0 Bit 2. ffdhe4096 • Byte 0 Bit 3. secp256r1 • Byte 0 Bit 4. secp384r1 • Byte 0 Bit 5. secp521r1 All other values reserved.
20	*AEADCipherSuite*	4	Bit mask listing Requester-supported SPDM-enumerated cryptographic Encryption Cipher Suite algorithms. • Byte 0 Bit 0. AES-128-GCM • Byte 0 Bit 1. AES-256-GCM • Byte 0 Bit 2. CHACHA20_POLY1305 All other values reserved.
24	`Reserved`	4	Reserved
28	*ExtAsymCount*	1	Number of Requester-supported extended asymmetric key signature algorithms (=A). A + E + R + S shall be less than or equal to 16.
29	*ExtHashCount*	1	Number of Requester-supported extended hashing algorithms (=E). A + E + R + S shall be less than or equal to 16.
30	*ExtSessionKeyAlgCount*	1	Number of Requester-supported session key exchange algorithms. (=S). A + E + R + S shall be less than or equal to 16.

Offset	Field	Size (bytes)	Value
31	*ExtAEADCipherCount*	1	Number of Requester-supported encryption and integrity protection algorithms. (=R). A + E + R + S shall be less than or equal to 16.
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.
32 + 4*A + 4*E	*ExtSessionKey*	4*S	List of the extended session key algorithms supported by Requester. The Extended algorithm field format table describes the format of this field.
32 + 4*A + 4*E + 4*S	*ExtAEADCipher*	4*R	List of the extended encryption algorithms supported by Requester. The [Extended algorithm field format](#tabl e-extended-algorithm-field-format) table describes the format of this field.
32 + 4*A + 4*E + 4*S + 4*R	*ReqBaseAsymAlg*	4	Bit mask listing Requester-supported SPDM-enumerated asymmetric key signature algorithms for the purposes of signature generation. • Byte 0 Bit 0. TPM_ALG_RSASSA_2048 • Byte 0 Bit 1. TPM_ALG_RSAPSS_2048 • Byte 0 Bit 2. TPM_ALG_RSASSA_3072 • Byte 0 Bit 3. TPM_ALG_RSAPSS_3072 • Byte 0 Bit 4. TPM_ALG_ECDSA_ECC_NIST_P256 • Byte 0 Bit 5. TPM_ALG_RSASSA_4096 • Byte 0 Bit 6. TPM_ALG_RSAPSS_4096 • Byte 0 Bit 7. TPM_ALG_ECDSA_ECC_NIST_P384 • Byte 1 Bit 0. TPM_ALG_ECDSA_ECC_NIST_P521 All other values reserved.

Successful ALGORITHMS response message

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0x63=ALGORITHMS`
2	`Param1`	1	Reserved
3	`Param2`	1	Reserved
4	`Length`	2	Length of the response message, in bytes.

Offset	Field	Size (bytes)	Value
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
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 one bit in this bit field shall be set. Otherwise, the Responder shall set this field to `0`. A Responder shall only select bit 0 if the Responder supports raw bit streams as the only form of measurement; otherwise, it shall select one of the other bits.
12	`BaseAsymSel`	4	Bit mask listing the SPDM-enumerated asymmetric key signature algorithm selected. A Responder that returns `CHAL_CAP=0` and `MEAS_CAP!=2` shall set this field 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	`DHESel`	4	Bit mask listing SPDM-enumerated cryptographic Diffie-Hellman algorithm selected. • Byte 0 Bit 0. ffdhe2048 • Byte 0 Bit 1. ffdhe3072 • Byte 0 Bit 2. ffdhe4096 • Byte 0 Bit 3. secp256r1 • Byte 0 Bit 4. secp384r1 • Byte 0 Bit 5. secp521r1 All other values reserved.

Offset	Field	Size (bytes)	Value
24	`AEADCipherSel`	4	Bit mask listing SPDM-enumerated cryptographic Encryption Cipher Suite algorithms selected. • Byte 0 Bit 0. AES-128-GCM • Byte 0 Bit 1. AES-256-GCM • Byte 0 Bit 2. CHACHA20_POLY1305 All other values reserved.
28	`Reserved`	4	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` .
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	`ExtSessionKeyAlgCount`	1	Number of Requester-supported session key exchange algorithms selected. Shall be either `0` or `1` . A Responder that returns `ENCRPT_CAP=0` and `MAC_CAP=0` shall set this field to `0` .
35	*ExtAEADCipherCount*	1	Number of Requester-supported encryption and integrity protection algorithms selected. Shall be either `0` or `1` . A responder that returns `ENCRPT_CAP=0` or `MAC_CAP=0` shall set this field to `0` .
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.
36 + 4*A' + 4*E'	`ExtSessionKeySel`	4*S'	Extended session key exchange algorithm selected. The Responder shall use this session key exchange algorithm during all subsequent response messages to the Requester. The Requester shall use this session key exchange algorithm during all subsequent applicable request messages to the Responder. The Extended algorithm field format table describes the format of this field.
36 + 4*A' +4*E' + 4*S'	`ExtAEADCipherSel`	4*R'	Extended encryption algorithm selected. The Responder shall use this encryption algorithm during all subsequent response messages to the Requester. The Requester shall use this encryption algorithm during all subsequent applicable request messages to the Responder. The Extended algorithm field format table describes the format of this field.

Offset	Field	Size (bytes)	Value
36 + 4*A' + 4*E' + 4*S' + 4*R'	`ReqBaseAsySel`	4	Bit mask listing the SPDM-enumerated asymmetric key signature verification algorithm selected. When a Requester indicates `CHAL_CAP=0` and `MEAS_CAP!=2`, the Responder shall set this field to `0`. Other Responders shall set no more than one bit.

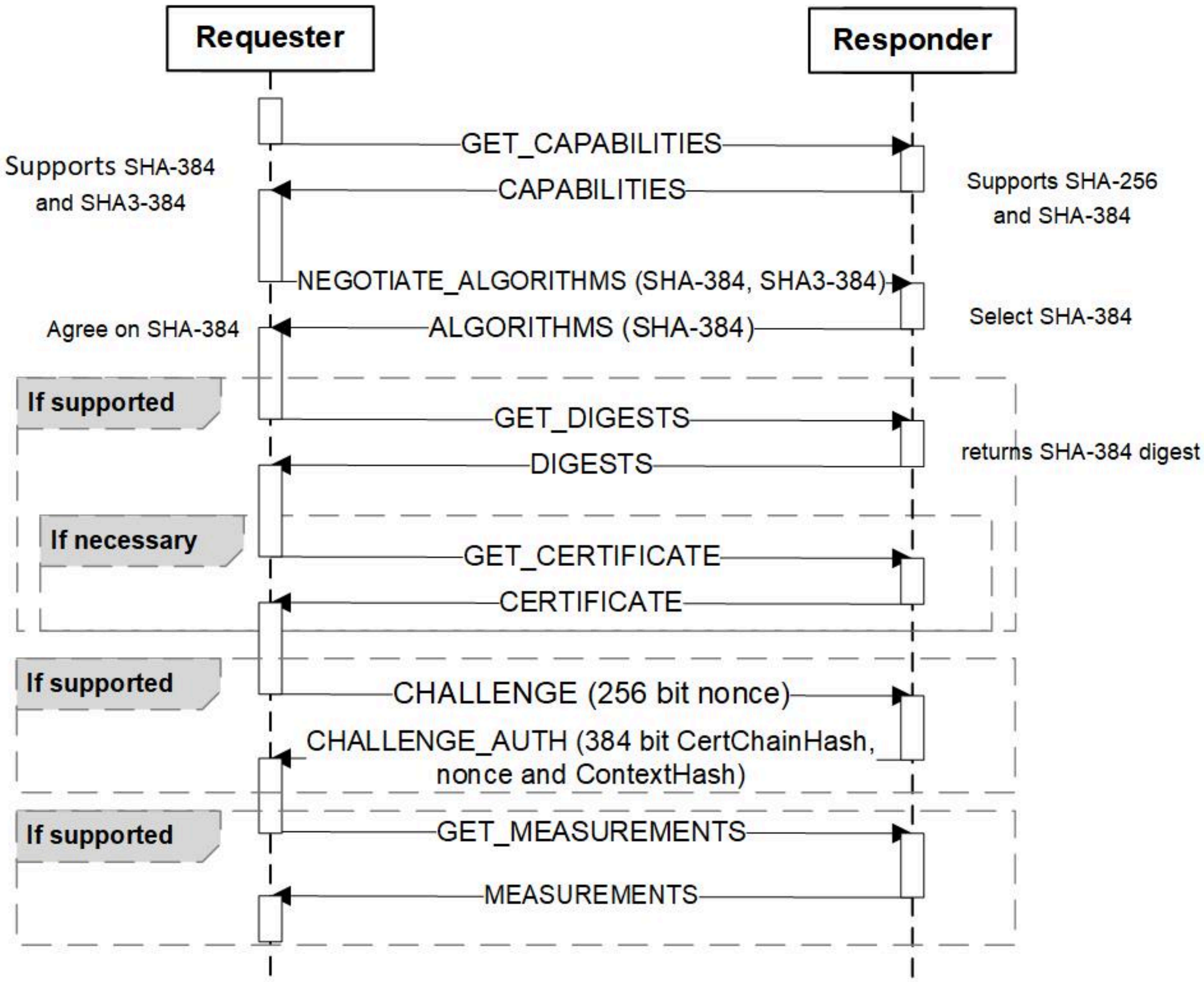
Extended algorithm field format

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 this field's value.
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.

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

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

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.

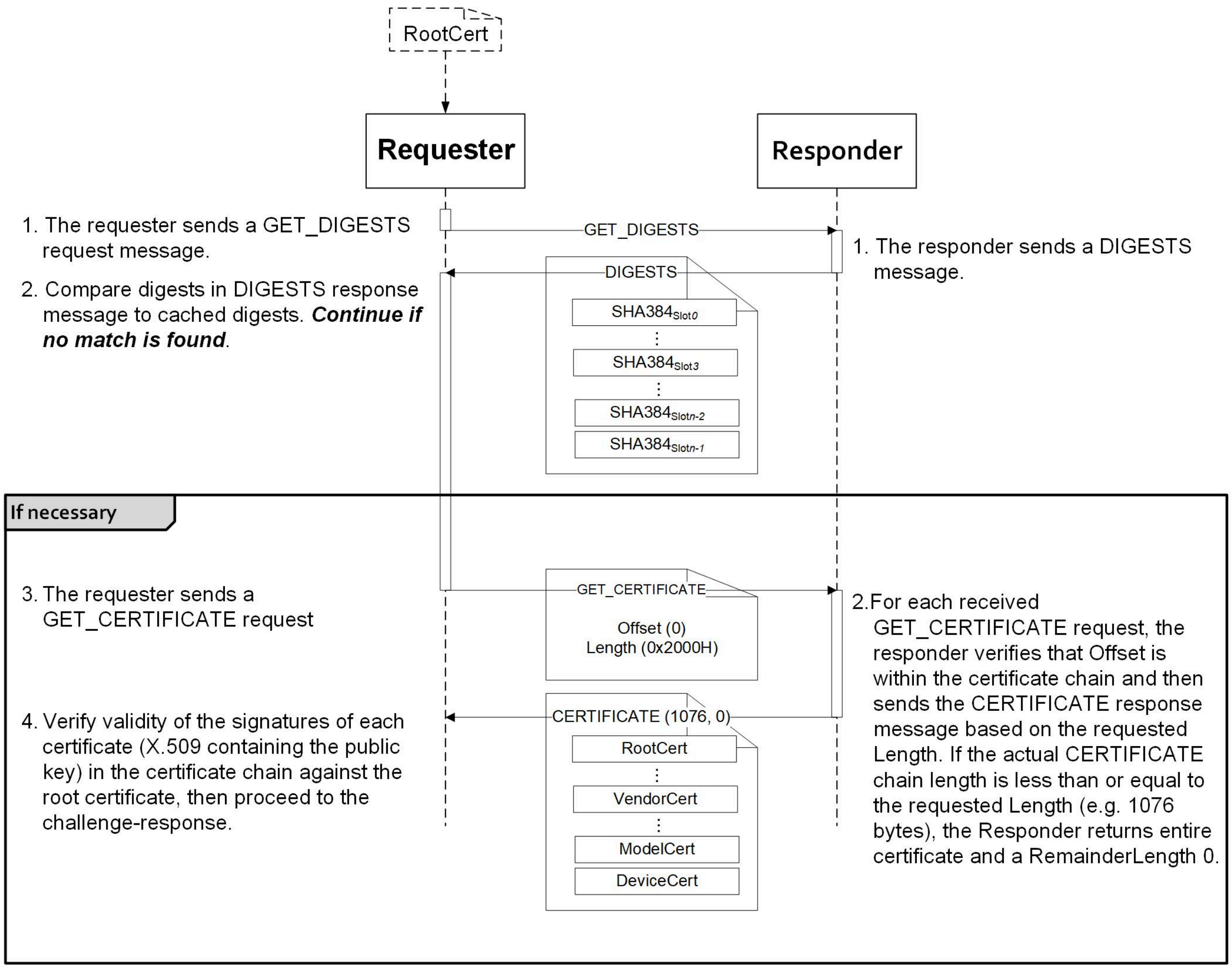


The SPDM protocol accounts for the possibility that both endpoints may issue `NEGOTIATE_ALGORITHMS` request messages independently of each other. In this case, the endpoint A Requester and endpoint B Responder communication pair may select a different algorithm compared to the endpoint B Requester and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

7.5 Responder identity authentication

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

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



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

For the runtime challenge-response flow, the signature field in the `CHALLENGE_AUTH` response message payload shall be signed 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.

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.

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.

7.5.1 Certificates and certificate chains

Each Responder that supports identity authentication shall carry at least one certificate chain. A certificate chain contains an ordered list of certificates, presented as the binary (byte) concatenation of the fields that the Certificate chain format shows.

Each certificate shall be in ASN.1 DER-encoded X.509 v3 format. The ASN.1 DER encoding of each individual certificate can be analyzed to determine its length. The minimum number of certificates within a chain shall be one, in which case the single certificate is the device-specific certificate. The Responder shall contain a single public-private key pair per supported algorithm for its hardware identity, regardless of how many certificate chains are stored on the device. The Responder selects a single asymmetric key signature algorithm per Requester.

Certificate chains are stored in locations called slots. Each slot shall either be empty or contain one complete certificate chain. A Product shall not contain more than 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.

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

Certificate chain format

Offset	Field	Size	Description
0	`Length`	2	Total length of the certificate chain, in bytes, including all fields in this table. This field is little endian.
2	`Reserved`	2	Reserved.
4	`RootHash`	H	Digest of the Root Certificate. Note that Root Certificate is ASN.1 DER-encoded for this digest. This field is big endian.
4 + H	`Certificates`	Length - (4 + H)	One or more ASN.1 DER-encoded 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.

7.6 GET_DIGESTS request message and DIGESTS response message

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

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

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

The digests in the Successful DIGESTS response message table are in big endian.

GET_DIGESTS request message

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

Successful DIGESTS response message

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0x01=DIGESTS`
2	`Param1`	1	Reserved
3	`Param2`	1	Slot mask. The bit in position K of this byte shall be set to 1b if and only if slot number K contains a certificate chain for the protocol version in the SPDMVersion field. (Bit 0 is the least significant bit of the byte.) 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.

7.7 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 Responder's certificate chain.

GET_CERTIFICATE request message

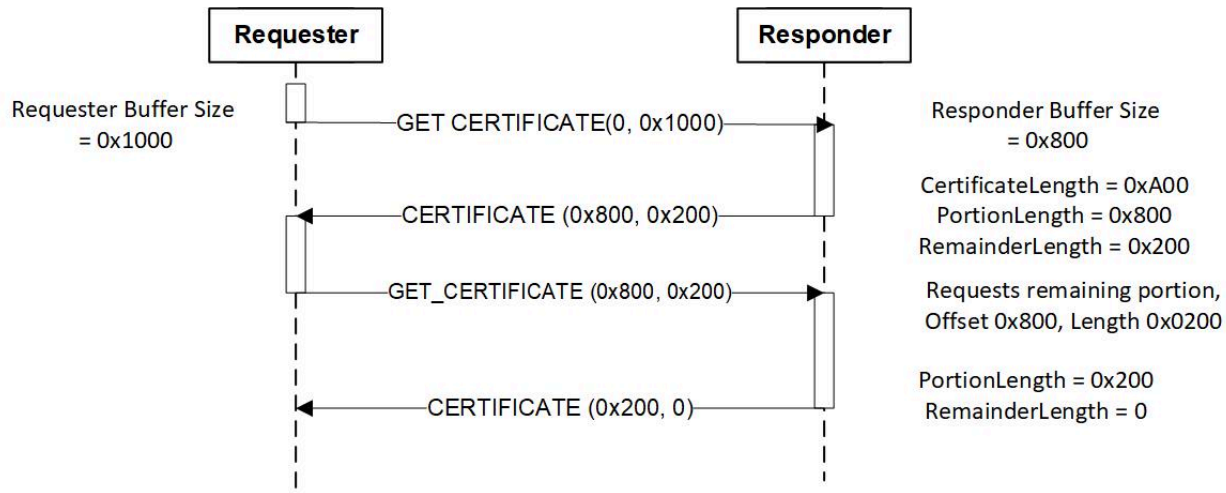
Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0x82=GET_CERTIFICATE`
2	`Param1`	1	Slot number of the target certificate chain to read from. The value in this field shall be between 0 and 7 inclusive.
3	`Param2`	1	Reserved
4	`Offset`	2	Offset in bytes from the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

Successful CERTIFICATE response message

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

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 Responder's internal buffer.
6	`RemainderLength`	2	Number of bytes of the certificate chain that have not been sent yet after the current response. For the last response, this field shall be 0 as an indication to the Requester that the entire certificate chain has been sent.
8	`CertChain`	`PortionLength`	Requested contents of target certificate chain, formatted in DER. This field is big endian.

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.



7.7.1 Leaf certificate

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

Required fields

Field	Description
`Version`	Version of the encoded certificate shall be present and shall be `3` or `2`.
`Serial Number`	CA-assigned serial number shall be present with a positive integer value.

Field	Description
`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`	Certificate may include this attribute. If the validity attribute is present, the value for `notBefore` field should be assigned the generalized `19700101000000Z` time value and `notAfter` field should be assigned the generalized `99991231235959Z` time value.
`Subject Public Key Info`	Device public key and the algorithm shall be present.
`Extended Key Usage`	Shall be present and key usage bit for digital signature shall be set.

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.

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 B - Leaf certificate example shows an example leaf certificate.

7.8 CHALLENGE request message and CHALLENGE_AUTH response message

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

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

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

CHALLENGE request message

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0x83=CHALLENGE`
2	`Param1`	1	Slot number of the Responder's certificate chain that shall be used for authentication.
3	`Param2`	1	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.

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.
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.
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.

7.8.1 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 Responder's leaf certificate 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.

7.8.2 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`

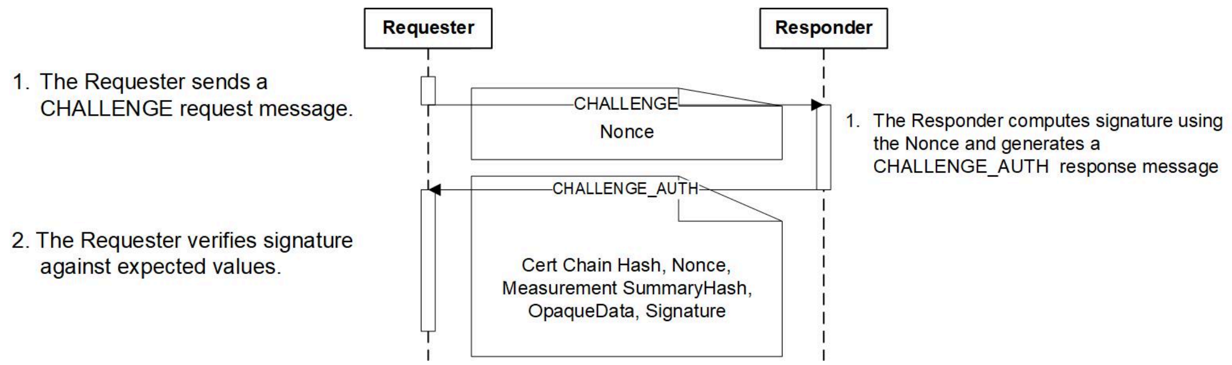
     Public key associated with the leaf certificate of the Responder with `slot=Param1` of the `CHALLENGE` request message.

   - `Hash`

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

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

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



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

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.

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`

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.

Request ordering and message transcript computation rules for M1/M2

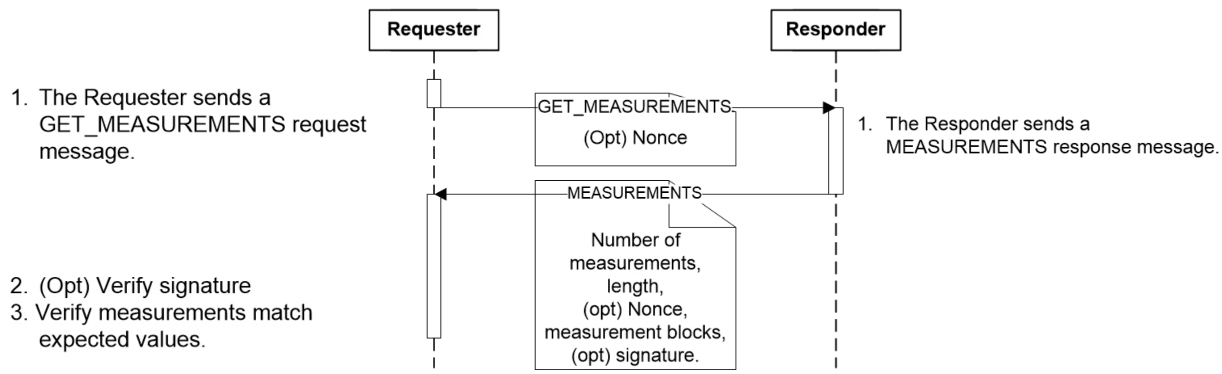
Requests	Implementation requirements	M1/M2=Concatenate (A, B, C)
Reset	NA	M1/M2=null

Requests	Implementation requirements	M1/M2=Concatenate (A, B, C)
`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_DIGEST, DIGEST, GET_CERTFICATE, CERTIFICATE)`
`GET_DIGESTS`, `GET_CERTFICATE`	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`
Other	If the Requester chooses to issue `GET_MEASUREMENTS` or `KEY_EXCHANGE` or `FINISH` or `PSK_BASED_EXCHANGE` or `PSK_BASED_FINISH` or `KEY_UPDATE` or `HEARTBEAT` or `GET_ENCAPSULATED_REQUEST` or `DELIVER_ENCAPSULATED_RESPONSE` or `END_SESSSION` request(s) and skips `CHALLENGE` completion, M1 and M2 are reset to `null`.	`M1/M2=null`

7.10 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.



7.11 GET_MEASUREMENTS request message and MEASUREMENTS response message

This request message shall retrieve firmware measurements. 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`

Offset	Field	Size (bytes)	Value
1	`RequestResponseCode`	1	`0xE0=GET_MEASUREMENTS`
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.

GET_MEASUREMENTS request attributes

Bits	Value	Description
`0`	`1`	If the Responder can generate a signature as shown in `CAPABILITIES` message, this bit's value shall indicate to the Responder to generate a signature. The Responder shall generate a signature in the corresponding response. The `Nonce` field shall be present in the request.
`0`	`0`	Responders that cannot generate a signature as shown in the `CAPABILITIES` message shall use this bit's value. For Responders that can generate signatures, this bit's value 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.
`[7:1]`	Reserved	Reserved

Successful MEASUREMENTS response message

Offset	Field	Size (bytes)	Value
0	`SPDMVersion`	1	`V1.0=0x10`
1	`RequestResponseCode`	1	`0x60=MEASUREMENTS`
2	`Param1`	1	When `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

Offset	Field	Size (bytes)	Value
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.
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.

7.11.1 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.

7.11.1.1 DMTF specification for the Measurement field of a measurement block

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.

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.

Offset	Field	Size (bytes)	Value
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 `DMTFSpecMeasurementType[7]`.

7.11.2 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 Responder's slot 0 leaf certificate.

   - `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.

7.11.3 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.

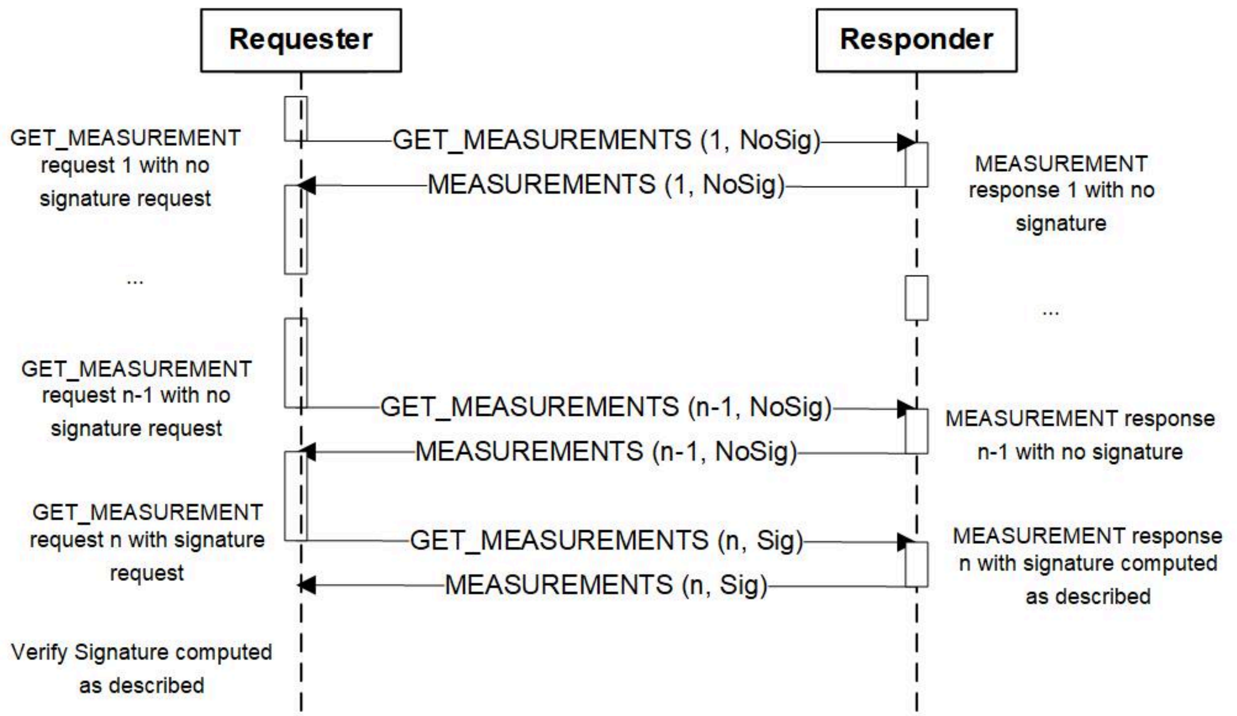
     The Successful ALGORITHMS response message table describes the `BaseAsymSel` and `ExtAsymSel` fields.

   - `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.

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.



7.12 ERROR response message

For a 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`

Offset	Field	Size (bytes)	Value
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.
`InvalidSession`	`0x02`	The record layer used an invalid session ID.	This shall be the invalid session ID.	Reserved
`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.
`DecryptError`	`0x06`	The receiver of the record cannot decrypt the record or verify data during the session handshake.	Reserved	Reserved
`UnsupportedRequest`	`0x07`	The `RequestResponseCode` in the request message is unsupported.	`RequestResponseCode` in the request message.	No extended error data is provided
`RequestInFlight`	`0x08`	The Responder has an delivered a request to which it is still waiting for the response.	Reserved	Reserved
`InvalidResponseCode`	`0x09`	The Requester delivered an invalid response for an encapsulated response.	Reserved	Reserved
`SessionLimitExceeded`	`0x0A`	Reserved	Reserved	Reserved
Reserved	`0x0b` - `0x40`	Reserved	Reserved	Reserved
`MajorVersionMismatch`	`0x41`	Requested SPDM Major Version is not supported.	`0x00`	No extended error data provided.

Error code	Value	Description	Error data	ExtendedErrorData
`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.

ResponseNotReady extended error data

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

Registry or standards body ID

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

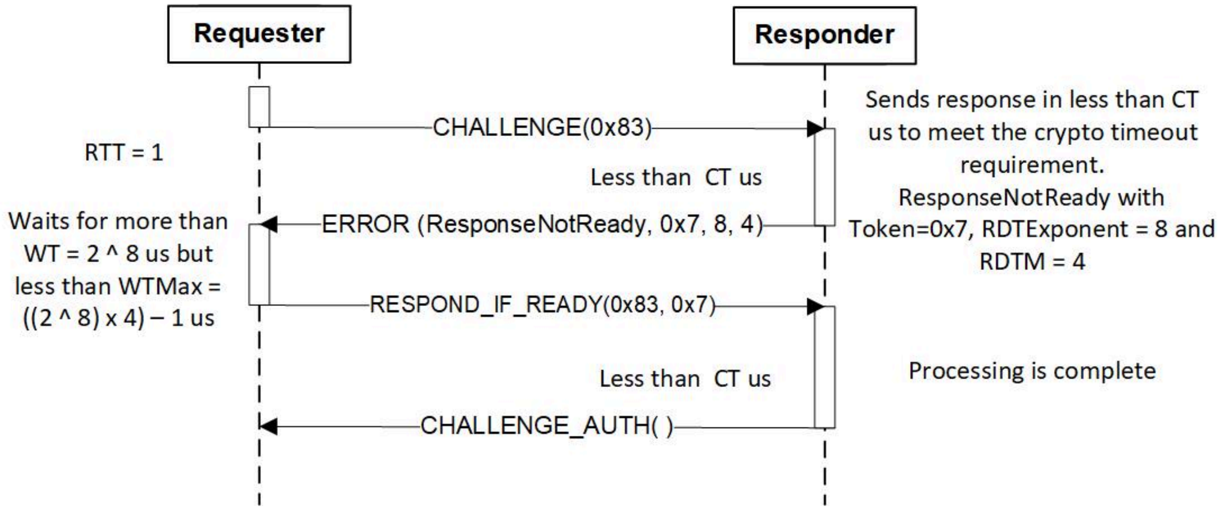
ID	Vendor ID length (bytes)	Registry or standards body name	Description
`0x0`	0	DMTF	DMTF does not have a Vendor ID registry. At present, DMTF does not have any algorithms defined for use in extended algorithms fields.
`0x1`	2	TCG	Vendor is identified by using TCG Vendor ID Registry. For extended algorithms,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

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.
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 `Error Data` 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.

7.13 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`
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.

7.14 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.
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.

7.14.1 VENDOR_DEFINED_RESPONSE response message

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

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

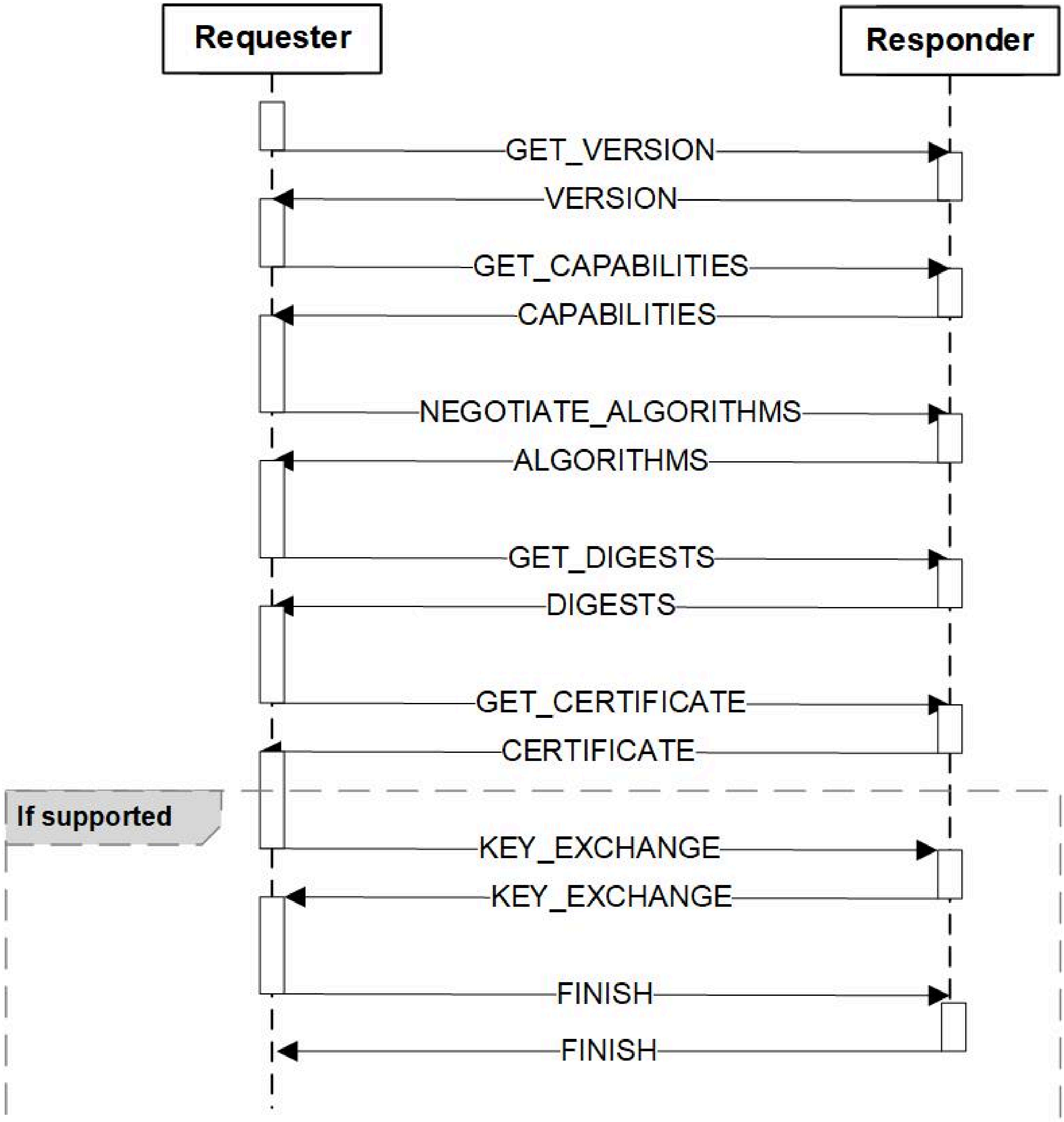
VENDOR_DEFINED_RESPONSE response message

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`	ReqLength	Standard or vendor shall use this value to send the response payload.

7.15 KEY_EXCHANGE Request and KEY_EXCHANGE Response

This request message shall initiate a handshake between Requester and Responder intended to authenticate the Responder (or optionally both parties), negotiate cryptographic parameters (in addition to those negotiated in the last `NEGOTIATE_ALGORITHMS` / `ALGORITHMS` exchange), and establish shared keying material. The KEY_EXCHANGE request message table shows the `KEY_EXCHANGE request` request message format and the KEY_EXCHANGE response message table shows the `KEY_EXCHANGE response` response message format. The handshake is completed by the successful exchange of the `FINISH request` and `FINISH response` messages, presented in the next section, and depends on the tight coupling between the two request/response message pairs.



KEY_EXCHANGE request message

Offset	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0xE4 = KEY_EXCHANGE Request`
2	Param1	1	Reserved
3	Param2	1	The slot number of the target certificate chain that the Responder will use for authentication. The value in this field shall be between 0 and 7 inclusive to identify a valid certificate slot.
4	DHE_Named_Group	4	• Byte 0 Bit 0 – Finite Field ffdhe2048 (D = 256) – RFC 7919 Appendix A.1 • Byte 0 Bit 1 – Finite Field ffdhe3072 (D = 384) – RFC 7919 Appendix A.2 • Byte 0 Bit 2 – Finite Field ffdhe4096 (D = 512) – RFC 7919 Appendix A.3 • Byte 0 Bit 3 – ECDHE secp256r1 (D = 64, C = 32) – RFC 8446 Section 4.2.8.2 • Byte 0 Bit 4 – ECDHE secp384r1 (D = 96, C = 48) – RFC 8446 Section 4.2.8.2 • Byte 0 Bit 5 – ECDHE secp521r1 (D = 132 C = 66) – RFC 8446 Section 4.2.8.2 All other values reserved.
8	RandomData	32	Requester-provided random data.
40	ExchangeData	D	If the selected DHE_Named_Group is finite field, then ExchangeData represents the computed public information. If the selected DHE_Named_Group is ECDHE, the exchange data represents the X and Y values in network byte order. Specifically, X is [0: C - 1] and Y is [C : D – 1]. In both cases the size of D (and C for ECDHE) is derived from the selected DHE_Named_Group.

Successful KEY_EXCHANGE response message

Offset	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0x64 = KEY_EXCHANGE response`
2	Param1	1	HeartbeatPeriod The value of this field shall be zero if Heartbeat is not supported. Otherwise, the value shall be in units of seconds.
3	Param2	1	Session ID. The Responder shall choose a session ID. It should be different from the 5 previous sessions or active sessions to the same endpoint.
4	Length	2	Length of the entire request in bytes.

Offset	Field	Size in bytes	Value
6	Mut_Auth_Requested	1	• Bit 0 – If set, Responder is requesting a Mutual Authentication flow. Requester shall initiate a GET_ENCAPSULATED_REQUEST request. • Bit 1 - If set, Responder is requesting a Mutual Authentication flow with implicit GET_DIGESTS request. Requester shall initiate a DELIVER_ENCAPSULATED_RESPONSE request which encapsulates DIGESTS response. Bit [7:2] reserved.
7	Reserved	1	reserved.
8	RandomData	32	Responder-provided random data.
40	ExchangeData	D	If the selected DHE_Named_Group is finite field, then ExchangeData represents the computed public information. If the selected DHE_Named_Group is ECDHE, the exchange data represents the X and Y values in network byte order. Specifically, X is [0: C - 1] and Y is [C : D – 1]. In both cases the size of D (and C for ECDHE) is derived from the selected DHE_Named_Group.
40+D	Signature	S	Signature over the transcript hash. S is the size of the asymmetric signing algorithm output the Responder selected via the last `ALGORITHMS` response message to the Requester. The construction of the transcript hash is defined in Transcript Hash for `KEY_EXCHANGE response` signature.
40+D+S	VerifyData	H	An HMAC of the transcript hash using a MAC key derived from the shared session keys generated by the Requester and Responder. The construction of the transcript hash is defined in Transcript Hash for `KEY_EXCHANGE response` HMAC.

7.16 FINISH Request and FINISH Response

This request message shall complete the handshake between Requester and Responder initiated by a `KEY_EXCHANGE request`. The purpose of the `FINISH request` and `FINISH response` messages is to provide key confirmation, bind each party's identity to the exchanged keys and protect the entire handshake against manipulation by an active attacker. The FINISH request message table shows the `FINISH request` request message format and the FINISH response message table shows the `FINISH response` response message format.

FINISH request message

Offset	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0xE5 = FINISH request`

Offset	Field	Size in bytes	Value
2	Param1	1	Bit 0 – If set, the Signature field is included. This bit shall be set when mutual authentication occurs. All other bits reserved.
3	Param2	1	Slot ID. Only valid if Param1= 0x01, otherwise reserved. Slot number of the target Certificate Chain being authenticated in signature field. The value in this field shall be between 0 and 7 inclusive.
4	Signature	S	Signature over the transcript hash. S is the size of the asymmetric signing algorithm output the Responder selected via the last `ALGORITHMS` response message to the Requester. S is zero and field not present if Param1 = 0x00. The construction of the transcript hash is defined in Transcript Hash for `FINISH request` signature, Responder-only authentication and Transcript Hash for `FINISH request` signature, mutual authentication.
4+S	VerifyData	H	An HMAC of the transcript hash using a MAC key derived from the shared session keys generated by the Requester and Responder. The construction of the transcript hash is defined in Transcript Hash for `FINISH request` HMAC, Responder-only authentication and Transcript Hash for `FINISH request` HMAC, mutual authentication.

Successful FINISH response message

Offset	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0x65 = FINISH response`
2	Param1	1	Reserved.
3	Param2	1	Reserved.

7.16.1 Transcript Hash calculation rules

The Transcript Hash is calculated by concatenating the prescribed full messages or message fields in order. In the following, the notation: `[${message_name}]`.`${field_name}` is used, where:

- `${message_name}` is the name of the request or response message.
- `${field_name}` is the name of the field in the request or response message. The asterisk (`*`) means all fields in that message.

Transcript Hash for `KEY_EXCHANGE` response signature:

```
1.  [GET_CAPABILITIES].*
2.  [CAPABILITIES].*
```

```
3.  [NEGOTIATE_ALGORITHMS].*
4.  [ALGORITHMS].*
5.  The specified certificate chain in DER format(i.e. KEY_EXCHANGE’s Slot Number)
6.  [KEY_EXCHANGE Request].*
7.  [KEY_EXCHANGE Response].SPDM Header Fields
8.  [KEY_EXCHANGE Response].Length
9.  [KEY_EXCHANGE Response].Mut_Auth_Requested
10. [KEY_EXCHANGE Response].Reserved
11. [KEY_EXCHANGE Response].RandomData
12. [KEY_EXCHANGE Response].ExchangeData
```

Transcript Hash for `KEY_EXCHANGE` response HMAC:

```
1.  [GET_CAPABILITIES].*
2.  [CAPABILITIES].*
3.  [NEGOTIATE_ALGORITHMS].*
4.  [ALGORITHMS].*
5.  The specified certificate chain in DER format (i.e. KEY_EXCHANGE’s request Param2)
6.  [KEY_EXCHANGE Request].*
7.  [KEY_EXCHANGE Response].SPDM Header Fields
8.  [KEY_EXCHANGE Response].Length
9.  [KEY_EXCHANGE Response].Mut_Auth_Requested
10. [KEY_EXCHANGE Response].Reserved
11. [KEY_EXCHANGE Response].RandomData
12. [KEY_EXCHANGE Response].ExchangeData
13. [KEY_EXCHANGE Response].Signature
```

Transcript Hash for `FINISH` request signature, Responder-only authentication:

```
1. [GET_CAPABILITIES].*
2. [CAPABILITIES].*
3. [NEGOTIATE_ALGORITHMS].*
4. [ALGORITHMS].*
5. The specified certificate chain in DER format (i.e. KEY_EXCHANGE’s request Param2)
6. [KEY_EXCHANGE Request].*
7. [KEY_EXCHANGE Response].*
8. [FINISH Request].SPDM Header Fields
```

Transcript Hash for `FINISH` request signature, mutual authentication:

```
1. [GET_CAPABILITIES].*
2. [CAPABILITIES].*
3. [NEGOTIATE_ALGORITHMS].*
```

```
4. [ALGORITHMS].*
5. The specified certificate chain in DER format (i.e. KEY_EXCHANGE’s request Param2)
6. [KEY_EXCHANGE Request].*
7. [KEY_EXCHANGE Response].*
8. The specified certificate chain in DER format (i.e. FINISH Request’s Param2).
9. [FINISH Request].SPDM Header Fields
```

Transcript Hash for `FINISH` request HMAC, Responder-only authentication:

```
1. [GET_CAPABILITIES].*
2. [CAPABILITIES].*
3. [NEGOTIATE_ALGORITHMS].*
4. [ALGORITHMS].*
5. The specified certificate chain in DER format (i.e. KEY_EXCHANGE’s request Param2)
6. [KEY_EXCHANGE Request].*
7. [KEY_EXCHANGE Response].*
8. [FINISH Request].SPDM Header Fields
```

Transcript Hash for `FINISH` request HMAC, mutual authentication:

```
1.  [GET_CAPABILITIES].*
2.  [CAPABILITIES].*
3.  [NEGOTIATE_ALGORITHMS].*
4.  [ALGORITHMS].*
5.  The specified certificate chain in DER format (i.e. KEY_EXCHANGE’s request Param2)
6.  [KEY_EXCHANGE Request].*
7.  [KEY_EXCHANGE Response].*
8.  The specified certificate chain in DER format (i.e. FINISH Request’s Param2).
9.  [FINISH Request].SPDM Header Fields
10. [FINISH Request].Signature
```

7.17 PSK_BASED_EXCHANGE Request and PSK_BASED_EXCHANGE Response

The Pre-Shared Key (PSK) key exchange scheme provides an option for a Requester and a Responder to perform mutual authentication and session key establishment with symmetric-key cryptography. This option is especially useful for endpoints that do not support asymmetric-key cryptography or certificate processing. This option can also be leveraged to expedite the session key establishment, even if asymmetric-key cryptography is supported.

This option requires the Requester and the Responder to have prior knowledge of a common PSK before the handshake. Essentially, the PSK serves as a mutual authentication credential and the base of the session key establishment. As such, only the two endpoints and potentially a trusted third party that provisions the PSK to the two endpoints may know the value of the PSK.

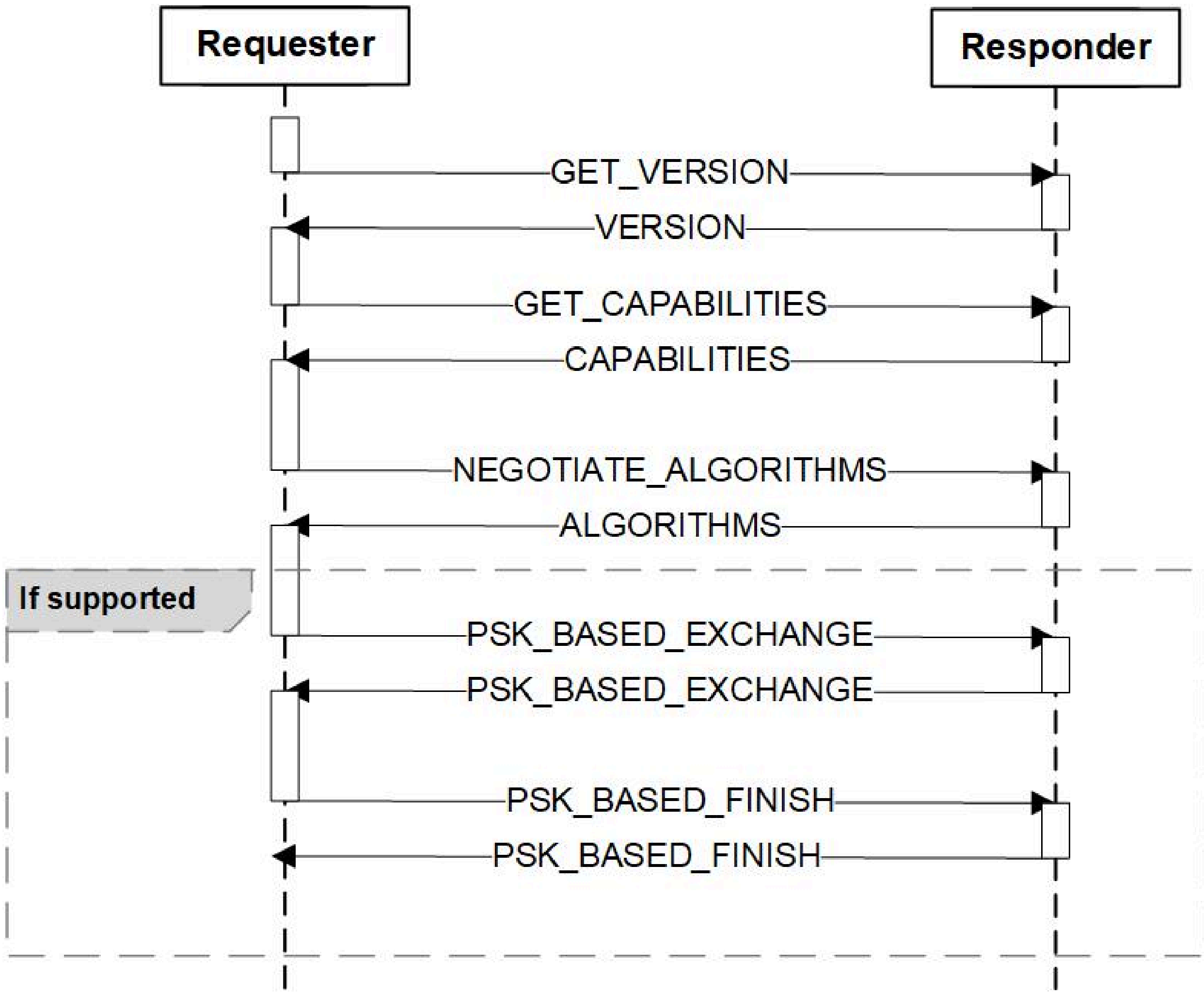
A Requester may be paired with multiple Responders. Likewise, a Responder may be paired with multiple Requesters. A pair of Requester and Responder may be provisioned with one or more PSKs. An endpoint may act as a Requester to one device and simultaneously a Responder to another device. It is the responsibility of the transport layer to identify the peer and establish communication between the two endpoints, before the PSK-based session key exchange starts.

The PSK may be provisioned in a trusted environment, for example, during the secure manufacturing process. In an untrusted environment, the PSK may be agreed upon between the two endpoints using a secure protocol. The mechanism for PSK provisioning is out of scope of this specification. The size of the provisioned PSK is determined by the requirement of security strength of the application, but should be at least 128 bits and recommended to be 256 bits or larger. During PSK provisioning, an endpoint's capabilities and supported algorithms may be communicated to the peer. Therefore, SPDM commands `GET_CAPABILITIES` and `NEGOTIATE_ALGORITHMS` are not required during session key establishment with the PSK option.

Two commands are defined for this option: PSK_BASED_EXCHANGE and PSK_BASED_FINISH.

The PSK_BASED_EXCHANGE command carries three responsibilities:

1. Prompts the Responder to acquire the specific PSK.
2. Exchanges contexts between the Requester and the Responder.
3. Proves to the Requester that the Responder knows the correct PSK and has derived the correct session keys.



PSK_BASED_EXCHANGE request message

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0xE6 = PSK_BASED_EXCHANGE Request`
2	Param1	1	Length of the opaque_PSK_data. Denoted as P onward.
3	Param2	1	Length of the requester_context. Denoted as R onward. R must be equal to or greater than H, where H is the size of the underlying MAC used in key derivation.
4:(4+R-1)	requester_context	R	Requester's context. Must include random nonce and optionally Requester's information.

Offsets	Field	Size in bytes	Value
(4+R):(4+R+P-1)	opaque_PSK_data	P	Opaque data required by the Responder to retrieve the PSK. Optional.
(4+R+P):(4+R+P+H-1)	requester_auth	H	Data to be verified by the Responder using requester_auth_key.

The field opaque_PSK_data is optional (absent if P is set to 0). It is introduced to address two scenarios:

- The Responder is provisioned with multiple PSKs and stores them in secure storage. The Requester uses opaque_PSK_data as an ID to specify which PSK will be used in this session.
- The Responder does not store the value of the PSK, but can derive the PSK using opaque_PSK_data. For example, if the Responder has an immutable UDS (Unique Device Secret) in fuses, then during provisioning, a PSK may be derived from the UDS or its derivative and a non-secret salt provided by the Requester. During session key establishment, the same salt is sent to the Responder in opaque_PSK_data of PSK_BASED_EXCHANGE request. This mechanism allows the Responder to support any number of PSKs, without consuming secure storage.

The requester_context is the Requester's contribution to session key derivation. It must contain a random nonce to make sure the derived session keys are ephemeral for this session only to mitigate against replay attacks. It may also contain other information from the Requester.

The requester_auth field is a MAC value. The MAC key, requester_auth_key, is calculated as described in Key Schedule. The data is the concatenation of all data sent so far between the Requester and the Responder:

1. `[GET_VERSION].*` (if issued)
2. `[VERSION].*` (if issued)
3. `[GET_CAPABILITIES].*` (if issued)
4. `[CAPABILITIES].*` (if issued)
5. `[NEGOTIATE_ALGORITHMS].*` (if issued)
6. `[ALGORITHMS].*` (if issued)
7. `[PSK_BASED_EXCHANGE Request].SPDMVersion`
8. `[PSK_BASED_EXCHANGE Request].RequestResponseCode`
9. `[PSK_BASED_EXCHANGE Request].Param1`
10. `[PSK_BASED_EXCHANGE Request].Param2`
11. `[PSK_BASED_EXCHANGE Request].requester_context`
12. `[PSK_BASED_EXCHANGE Request].opaque_PSK_data`

Upon receiving PSK_BASED_EXCHANGE request, the Responder:

1. Acquires PSK from opaque_PSK_data, if necessary.
2. Calculates requester_auth independently in the same manner and verifies the result matches

requester_auth in the request. If verification fails, the Responder aborts the session.

3. Generates responder_context, if supported.
4. Derives the Responder's finished_key by following Key Schedule.
5. Constructs PSK_BASED_EXCHANGE response message and sends to the Requester.

PSK_BASED_EXCHANGE response message

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0x66 = PSK_BASED_EXCHANGE Response`
2	Param1	1	Length of the responder_context. Denoted as Q onward.
3	Param2	1	Session ID. The Responder shall choose a session ID. It should be different from the 5 previous sessions or active sessions to the same endpoint.
4:(4+Q-1)	responder_context	Q	Responder's context. Optional. If present, must include a nonce and/or Responder's information.
(4+Q):(4+Q+H-1)	responder_verify_data	H	Data to be verified by the Requester using the Responder's finished_key.

The responder_context is the Responder's contribution to session key derivation. It should contain a nonce (random number or monotonic counter) and other information of the Responder. Because the Responder may be a constrained device that is not able to generate nonce, responder_context is optional. However, the Responder is required to use responder_context if it can generate a nonce.

It should be noted that the nonce in responder_context is critical for anti-replay. If a nonce is not present in responder_context, then the Responder is not challenging the Requester for real-time knowledge of PSK. Such a session is subject to replay attacks - a man-in-the-middle attacker could record and replay prior PSK_BASED_EXCHANGE and PSK_BASED_FINISH messages and set up a session with the Responder. But the bogus session would not leak secrets, so long as the PSK or session keys of the prior replayed session are not compromised.

Successful verification of requester_auth does not prove that the Requester has derived correct session keys for this session. If responder_context is present in the response (i.e., `PSK_CAP` in Responder's `CAPABILITIES` is `10b`), then the Requester must send PSK_BASED_FINISH with requester_verify_data to further prove that it has derived correct session keys. However, if responder_context is absent, then the Requester is not required to send PSK_BASED_FINISH, as the session keys are solely determined by the Requester. In other words, if the Responder demands session key verification, then it must use responder_context, even if a nonce is not included, to signal the Requester to send PSK_BASED_FINISH request.

To calculate responder_verify_data, the Responder calculates a MAC. The MAC key is the Responder's finished_key. The data is the concatenation of all data sent so far between the Requester and the Responder:

1. `[GET_VERSION].*` (if issued)
2. `[VERSION].*` (if issued)
3. `[GET_CAPABILITIES].*` (if issued)
4. `[CAPABILITIES].*` (if issued)
5. `[NEGOTIATE_ALGORITHMS].*` (if issued)
6. `[ALGORITHMS].*` (if issued)
7. `[PSK_BASED_EXCHANGE request].*`
8. `[PSK_BASED_EXCHANGE response].SPDMVersion`
9. `[PSK_BASED_EXCHANGE response].RequestResponseCode`
10. `[PSK_BASED_EXCHANGE response].Param1`
11. `[PSK_BASED_EXCHANGE response].Param2`
12. `[PSK_BASED_EXCHANGE response].responder_context`

Upon receiving PSK_BASED_EXCHANGE response, the Requester:

1. Derives the Responder's finish key by following Key Schedule.
2. Verify responder_verify_data by calculating the MAC in the same manner as the Responder. If verification fails, the Requester aborts the session.
3. If the Responder contributes to session key derivation (`PSK_CAP` in Responder's `CAPABILITIES` is `10b`), construct PSK_BASED_FINISH request and send to the Responder.

7.18 PSK_BASED_FINISH Request and PSK_BASED_FINISH Response

The PSK_BASED_FINISH request proves to the Responder that the Requester knows the PSK and has derived the correct session keys. This is achieved by a MAC value calculated with the Requester's finished_key and messages of this session. The Requester is required to send the PSK_BASED_FINISH only if responder_context is present in PSK_BASED_EXCHANGE response. Otherwise, PSK_BASED_FINISH is optional.

PSK_BASED_FINISH request message

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0xE7 = PSK_BASED_FINISH Request`
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4:(4+H-1)	requester_verify_data	H	Data to be verified by the Responder using the Requester's finished_key.

To calculate requester_verify_data, the Requester calculates a MAC. The key is the Requester's finished_key, as described in Key Schedule. The data is the concatenation of all data sent so far between the Requester and the Responder:

1. `[GET_VERSION].*` (if issued)
2. `[VERSION].*` (if issued)
3. `[GET_CAPABILITIES].*` (if issued)
4. `[CAPABILITIES].*` (if issued)
5. `[NEGOTIATE_ALGORITHMS].*` (if issued)
6. `[ALGORITHMS].*` (if issued)
7. `[PSK_BASED_EXCHANGE request].*`
8. `[PSK_BASED_EXCHANGE response].*`
9. `[PSK_BASED_FINISH request].SPDMVersion`
10. `[PSK_BASED_FINISH request].RequestResponseCode`
11. `[PSK_BASED_FINISH request].Param1`
12. `[PSK_BASED_FINISH request].Param2`

Upon receiving PSK_BASED_FINISH request, the Responder derives the Requester's finished_key and calculates the MAC independently in the same manner and verifies the result matches requester_verify_data. If verified, then the Responder constructs PSK_BASED_FINISH response and sends to the Requester. Otherwise, the Responder sends ERROR response message to the Requester.

PSK_BASED_FINISH response message

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0x67 = PSK_BASED_FINISH Response`
2	Param1	1	Reserved.
3	Param2	1	Reserved.

7.19 HEARTBEAT Request and HEARTBEAT Response

This request shall keep a session alive if `HEARTBEAT` is supported by both the Requester and Responder. The `HEARTBEAT` request shall be sent periodically as indicated in `HeartbeatPeriod` in either `KEY_EXCHANGE` or `PSK_BASED_EXCHANGE` response messages. The Responder shall terminate the session if a `HEARTBEAT` request is not received in twice `HeartbeatPeriod`. Likewise, the Requester shall terminate the session if a `HEARTBEAT` response or `ERROR` response is not received in twice `HeartbeatPeriod`. If an `Error` with `ErrorCode=InvalidSessionID` Response is received, the Requester shall terminate the session. The Requester may retry `HEARTBEAT` requests. The Requester shall wait `ST1` time for the response before retrying.

The timer for the Heartbeat period shall start at the transmission, for Responders, or reception, for Requester, of either the `PSK_BASED_FINISH` or `FINISH` response messages. When determining the value of HeartbeatPeriod, the Responder should ensure this value is sufficiently greater than `RTT`.

For further details of session termination, see Session Termination Handling.

The HEARTBEAT Request Message Format Table describes the format for the Heartbeat Request.

HEARTBEAT Request Message Format

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0xE8 = HEARTBEAT` Request
2	Param1	1	See Heartbeat Request Attributes Table.
3	Param2	1	Reserved.

HEARTBEAT Request Attributes Table

Bit Offset(s)	Value	Field Name	Description
0	0	Key Update Request	The Responder does not want to perform key update.
0	1	Key Update Request	The Responder requests a key update. The Requester shall perform a `KEY_UPDATE` when requested by the Responder and send the `KEY_UPDATE` request within `ST1` time. The Responder may set this bit. A Requester shall not set this bit.
[7:1]	Reserved	Reserved	Reserved

The HEARTBEAT Response Message Format Table describes the format for the Heartbeat Response.

HEARTBEAT Response Message Format

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0x68 = HEARTBEAT` Response

Offsets	Field	Size in bytes	Value
2	Param1	1	Reserved.
3	Param2	1	Reserved.

7.19.1 Heartbeat Additional Information

The `HEARTBEAT` request is one of two requests that a Responder may send without notice especially to ensure an active session or as part of the key update process.

7.20 KEY_UPDATE Request and KEY_UPDATE Response

To update session keys, this request shall be used. There are many reasons for doing this but an important one is when the per-record nonce will soon reach its maximum value and rollover. The KEY_UPDATE request is one of two requests that can be sent by the Responder as well. A KEY_UPDATE request shall update session keys in the direction of the request only. Because the Responder can also send this request, it is possible that two simultaneous key updates, one for each direction, can occur. However, only one KEY_UPDATE request for a single direction shall occur. Until the session key update synchronization successfully completes, subsequent KEY_UPDATE request for the same direction shall be considered a retry of the original KEY_UPDATE request.

KEY_UPDATE Request Message Format

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0xE9 = KEY_UPDATE` Request
2	Param1	1	Reserved.
3	Param2	1	Reserved.

KEY_UPDATE Response Message Format

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0xE9 = KEY_UPDATE` Request
2	Param1	1	Reserved.
3	Param2	1	Reserved.

7.20.1 Session Key Update Synchronization

For clarity, in the key update process, the term, sender, means the SPDM endpoint that sent the KEY_UPDATE request and the term, receiver, means the SPDM endpoint that received the KEY_UPDATE request, acted upon and responded to it accordingly. Furthermore, the sender only updates session keys in the sending direction and similarly, the receiver updates keys in the receiving direction.

To ensure the key update process is seamless while still allowing the transmission and reception of records, both sender and receiver shall follow this prescribed method. When the sender sends the `KEY_UPDATE` request, the sender should, at the same time, derive the new session keys for the sending direction. However, the sender shall not use the new session keys yet. Only upon the reception of the `KEY_UPDATE` response, the sender shall immediately use the new session keys as detailed in Major Secrets Update. At this time, best practices recommends the sender discards the old session key. Even though the receiver has transmitted the `KEY_UPDATE` response, the receiver shall use both the the current session keys and the new session keys. Assuming the transport layer delivers records in order, best practices recommend the receiver discard the old session keys upon successful decryption and authentication of a record using the new session keys.

After the sender switches to the new session keys, the sender shall send a `HEARTBEAT` request within `ST1` time and should retry until the `HEARTBEAT` response is received. This is to ensure that records are flowing in the direction of the receiver without reliance on the application layer. If no records are sent during this time, the receiver may have to maintain the old sessions keys for a longer than necessary period of time.

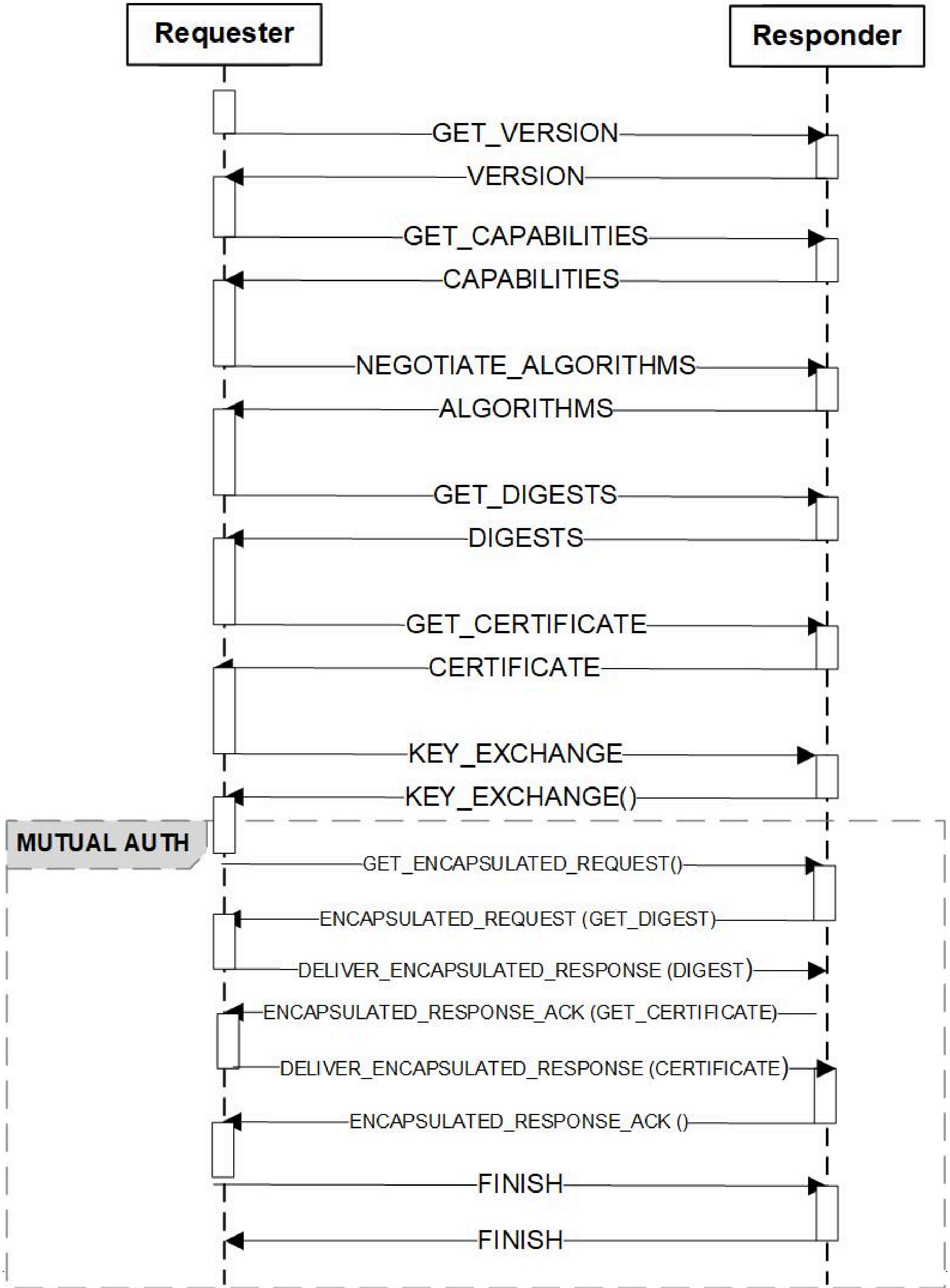
Finally, it bears repeating that a key update in one direction can happen simultaneously with a key update in the opposite direction. Still, the aforementioned synchronization process still works and occurs independently but simultaneously for each direction.

7.21 GET_ENCAPSULATED_REQUEST Request and ENCAPSULATED_REQUEST Response

This request retrieves an SPDM request message from the Responder. This request is only allowed in certain scenarios. See Session clauses for details.

The response for this message encapsulates an SPDM request message as if the Responder was a Requester. The request message format is described in `GET_ENCAPSULATED` Request Format Table. The Responder shall use the same SPDM version the Requester used.

Except for this request and `DELIVER_ENCAPSULATED_RESPONSE`, the Requester shall not send any other SPDM request message until successfully fulfilling the Responder's request. If a Responder receives a request other than `DELIVER_ENCAPSULATED_RESPONSE` or `GET_ENCAPSULATED_REQUEST` after the Responder already has provided a request to the Requester to which it has not received a response, the Responder shall respond with `ErrorCode=RequestInFlight`.


Requester
Responder
GET_VERSION
VERSION
GET_CAPABILITIES
CAPABILITIES
NEGOTIATE_ALGORITHMS
ALGORITHMS
GET_DIGESTS
DIGESTS
GET_CERTIFICATE
CERTIFICATE
KEY_EXCHANGE
KEY_EXCHANGE()
MUTUAL AUTH
GET_ENCAPSULATED_REQUEST()
ENCAPSULATED_REQUEST (GET_DIGEST)
DELIVER_ENCAPSULATED_RESPONSE (DIGEST)
ENCAPSULATED_RESPONSE_ACK (GET_CERTIFICATE)
DELIVER_ENCAPSULATED_RESPONSE (CERTIFICATE)
ENCAPSULATED_RESPONSE_ACK ()
FINISH
FINISH

GET_ENCAPSULATED_REQUEST Request Message Format

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0xEA = GET_ENCAPSULATED_REQUEST`
2	Param1	1	Reserved.
3	Param2	1	Reserved.

The ENCAPSULATED_REQUEST Response Format Table describes the format this response.

ENCAPSULATED_REQUEST Response Format Table

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0x6A = ENCAPSULATED_REQUEST` Response
2	Param1	1	Request ID. This field should be unique to help the Responder match response to request.
3	Param2	1	Reserved.
4+	Encapsulated Request	Variable	SPDM Request Message. The value of this field shall represent a valid SPDM request message. The length of this field is dependent on the SPDM Request message. The field shall start with the `RequestResponseCode` field. Both `GET_ENCAPSULATED_REQUEST` and `DELIVER_ENCAPSULATED_RESPONSE` shall be invalid requests and the Requester shall respond with `ErrorCode=UnexpectedRequest` if these requests are encapsulated.

7.22 DELIVER_ENCAPSULATED_RESPONSE Request and ENCAPSULATED_RESPONSE_ACK Received Message

In order to provide a response to a Responder's request, this request shall be used. This request delivers the response to the Responder's request which was encapsulated in the previous `ENCAPSULATED_REQUEST` response message.

Furthermore, if there are additional requests from the Responder, the Responder shall provide the next request in the `ENCAPSULATED_RESPONSE_ACK` response message.

As with the `GET_ENCAPSULATED_REQUEST` message, the Requester shall not send any other requests with the exception of `DELIVER_ENCAPSULATED_RESPONSE` until successfully delivering the response to the current request from the Responder. If a Responder receives a request other than `DELIVER_ENCAPSULATED_RESPONSE` after the Responder already has provided a request to the Requester to which it has not received a response, the Responder shall respond with `ErrorCode=RequestInFlight`.

The timing parameters for the response shall depend on the encapsulated request. This allows the Responder to process the response before delivering the next request. See Additional Information for more details.

The request message format is described in `DELIVER_ENCAPSULATED_RESPONSE` Request Message Format Table.

DELIVER_ENCAPSULATED_RESPONSE Request Message Format

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0xEB = DELIVER_ENCAPSULATED_RESPONSE` Request
2	Param1	1	Request ID. The Requester shall use the same Request ID as provided by the Responder.
3	Param2	1	Reserved.
4+	Encapsulated Response	Variable	SPDM Response Message. The value of this field shall represent a valid SPDM response message. The length of this field is dependent on the SPDM Response message. The field shall start with the `RequestResponseCode` field. Both `ENCAPSULATED_REQUEST` and `ENCAPSULATED_RESPONSE_ACK` shall be invalid responses and the Responder shall respond with `ErrorCode=InvalidResponseCode` if these responses are encapsulated.

The response message format is described in `ENCAPSULATED_RESPONSE_ACK` Response Format Table.

ENCAPSULATED_RESPONSE_ACK Response Format

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0x6B = ENCAPSULATED_RESPONSE_ACK`

Offsets	Field	Size in bytes	Value
2	Param1	1	Request ID. This field should be unique to help the Responder match response to request. This field shall be non-zero to indicate the presence of the next request in this message.
3	Param2	1	Reserved.
4+	Encapsulated Request	Variable	SPDM Request Message. The value of this field shall represent a valid SPDM request message. The length of this field is dependent on the SPDM Request message. The field shall start with the `RequestResponseCode` field. Both `GET_ENCAPSULATED_REQUEST` and `DELIVER_ENCAPSULATED_RESPONSE` shall be invalid requests and the Requester shall respond with `ErrorCode=UnexpectedRequest` if these requests are encapsulated.

7.22.1 Additional Information

Using a unique request ID is highly recommended to avoid confusion between a retry and a new request of the `DELIVER_ENCAPSULATED_RESPONSE` request. For example, if the Responder sent the `ENCAPSULATED_RESPONSE_ACK` and that failed in transmission over the wire, the Requester could send a retry. The responder may think the `DELIVER_ENCAPSULATED_RESPONSE` was a new request especially if the request encapsulated an `ERROR` message for the original request when in fact it was a retry of the original message.

In general, if a Responder has a new request, the response timing for `ENCAPSULATED_RESP_ACK` shall be subject to the same timing constraints as the original request. For example, if the encapsulated request was `CHALLENGE_AUTH`, the Responder, too, shall adhere to `CT` timing rules when it has a subsequent request. The Responder may return `ErrorCode=ResponseNotReady`.

7.23 END_SESSION Request and END_SESSION_ACK Response

This request shall terminate a session. Further communication between the Requester and Responder using the same session ID shall be prohibited. The Responder shall return `ErrorCode=InvalidSession` after session termination. See Session Termination Handling clause for details.

The END_SESSION Request Format table describes this request's format.

END_SESSION Request Message Format

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`

Offsets	Field	Size in bytes	Value
1	`RequestResponseCode`	1	`0xEC = END_SESSION`
2	Param1	1	See End Session Request Attributes.
3	Param2	1	Reserved.

End Session Request Attributes

Bit Offset(s)	Value	Field Name	Description
0	0	Negotiated State Preservation Indicator	If the Responder supports Negotiated State caching (`CAP_CACHE==1`), the Responder shall preserve the Negotiated State.
0	1	Negotiated State Preservation Indicator	If the Responder supports Negotiated State caching, the Responder shall also clear the Negotiated State as part of session termination.
[7:1]	Reserved	Reserved	Reserved

The response message for this request is described in END_SESSION_ACK Response Format Table.

END_SESSION_ACK Response Message Format

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	`V1.1 = 0x11`
1	`RequestResponseCode`	1	`0x6C = END_SESSION_ACK`
2	Param1	1	Reserved.
3	Param2	1	Reserved.

8 Session

Sessions allows a Requester and Responder to have multiple channels of communication. More importantly, it allows a Requester and Responder to build a secure communication channel with cryptographic information that is bound ephemerally.

There are 3 phases in a session: the handshake, the application and termination.

8.1 Session Handshake

The session handshake begins with either `PSK_BASED_EXCHANGE` or `KEY_EXCHANGE`. This phase also allows for authentication of the Requester if the Responder indicated that earlier in `ALGORITHMS` response. Furthermore, this phase of the session uses the handshake secrets to secure the communication as described in the Key Schedule.

The purpose of this phase is to build trust between the Responder and Requester, first, before either side can send application data. Additionally, it also ensures the integrity of the handshake and to a certain degree, synchronicity with the derived handshake secrets.

In this phase of the session, `GET_ENCAPSULATED_REQUEST` and `DELIVER_ENCAPSULATED_RESPONES` shall be used to obtain requests from the Responder to complete the authentication of the Requester, if the Responder indicated this in `ALGORITHMS` message. The only requests allowed to be encapsulated shall be `GET_DIGEST` and `GET_CERTIFICATE`. The Requester shall provide a signature in the `FINISH` request as described in the Finish clause.

If an error occurs in this phase with `ErrorCode = DecryptError`, the session shall immediately terminate and proceed to session termination.

A successful handshake ends with either `FINISH` or `PSK_BASED_FINISH` and the application phase begins.

8.2 Application Phase

Once the handshake completes and all validation passes, the session reaches the next phase where either the Responder and Requester may send application data. This phase is secured by the Record Layer.

The application phase ends when either an `HEARTBEAT` fails, `END_SESSION`, `HEARTBEAT` failure or an `ERROR` message with `ErrorCode = DecryptError`. The next phase is session termination.

8.3 Session Termination

Session terminations is simply an internal phase; there are no explicit SPDM messages sent or received. Requesters and Responders may have other reasons to terminate a session but that is outside the scope of this specification.

When a session terminates, both Requester and Responder shall destroy or clean up all session keys such as derived session secrets, DHE secrets and encryption keys. Requester and Responder may have other internal data tied to this session that they may want to also clean up.

8.4 Maximum Simultaneous Active Session

If a Responder supports key exchanges, the maximum number of simultaneous active sessions shall be a minimum of one. If the `KEY_EXCHANGE` or `PSK_BASED_EXCHANGE` request will exceed the Responder's maximum number of simultaneous active session, the Responder shall respond with an `Errorcode = SessionLimitExceeded`.

9 Key Schedule

A key schedule describes how to derive the various keys such as encryption keys used by a session as well as indicate when each key is used. Key derivation makes heavy use of `HMAC` as defined by RFC2104 and `HKDF-Expand` as described in RFC5869. SPDM defines the following additional functions.

```
BinConcat(Label, Context, Length)
```

where `BinConcat` shall be the concatenation of binary data, in the order shown in BinConcat Details Table:

BinConcat Details Table

Order	Data	Form	Endianness	Size
1	Length	Binary	Little	16 bits
2	"spdm1.1 "	Text	Text	8 bytes
3	Label	Text	Text	Variable
4	Context	Binary	Little	Hash.Length

The `HKDF-Expand` function prototype is as follows:

```
    HKDF-Expand(secret, context, Hash.Length)
```

The `HMAC-Hash` function prototype is described as follows:

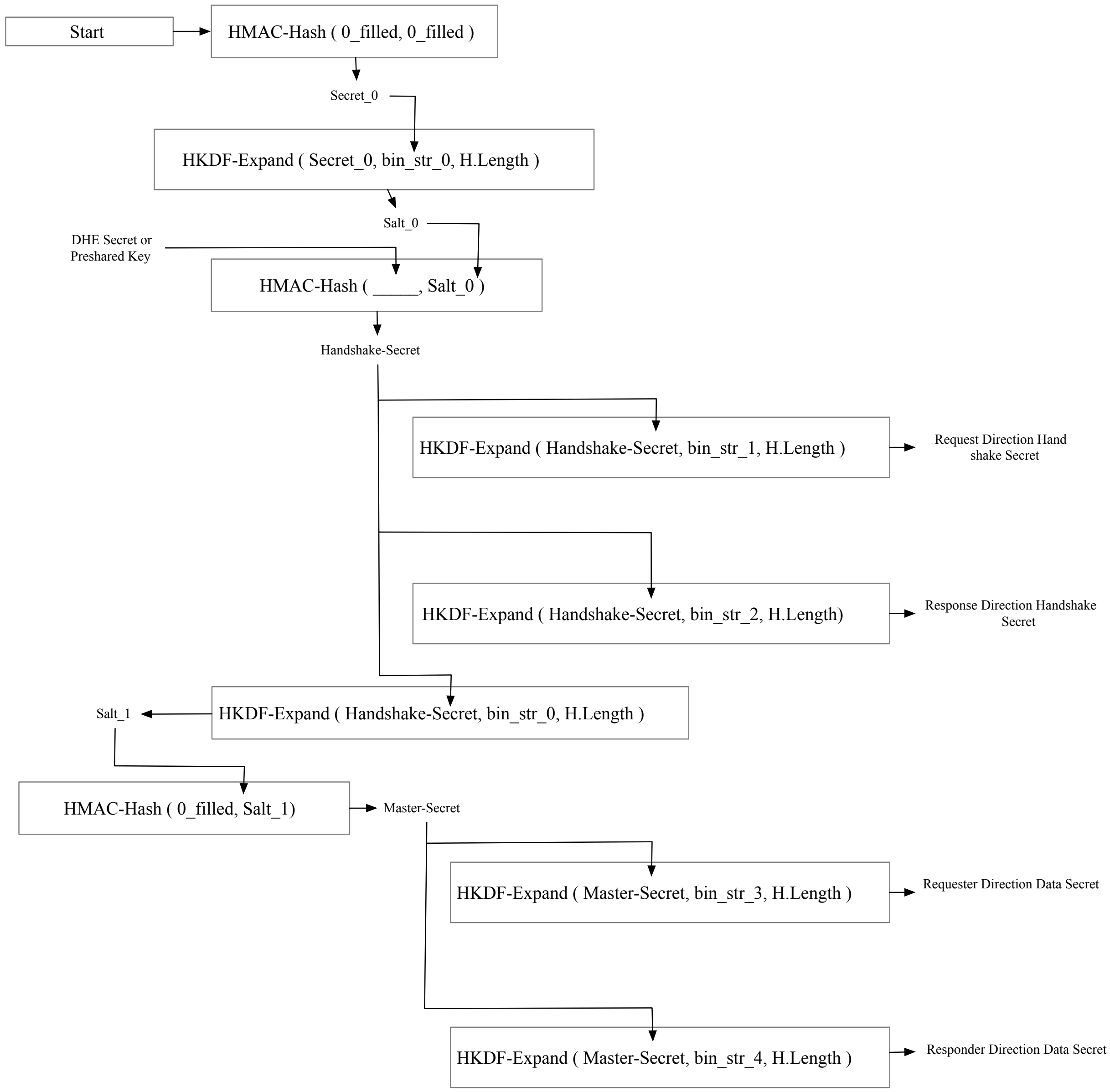
```
HMAC-Hash(salt, IKM);
```

where IKM is the Input Keying Material and HMAC-Hash uses `HMAC` as defined in RFC2104.

For `HKDF-Expand` and `HMAC-Hash`, the hash function shall be the selected hash function in `ALGORITHMS` response. Also, `Hash.Length` notation shall be the length of the hash function in `ALGORITHMS` response.

Both Responder and Requester shall use the key schedule shown in the Key Schedule Figure.

Key Schedule Figure



In the figure, arrows going out of the box are outputs of that box. Arrows going into the box are inputs into the box and point to the specific input parameter they are used in. All boxes represent a single function producing a single output and are given a name for clarity.

The Key Schedule Table accompanies the figure to complete the Key Schedule. The Responder and Requester shall also adhere to the definition of this table.

Key Schedule Table

Variable Name	Variable Definition
0_filled	A zero filled array of Hash.Length length.
bin_str0	BinConcat("derived", NULL, Hash.Length).
bin_str1	BinConcat("requester traffic", TH1, Hash.Length).
bin_str2	BinConcat("responder traffic", TH1, Hash.Length).
bin_str3	BinConcat("requester app traffic", TH2, Hash.Length)
bin_str4	BinConcat("responder app traffic", TH2, Hash.Length)
DHE Secret	This shall be the secret derived from `KEY_EXCHANGE`
Pre-shared Key	PSK

9.1 Transcript Hash in Key Derivation

There are two transcript hashes used in the Key Schedule, namely, **TH1** and **TH2**.

9.2 TH1 Definition

For `KEY_EXCHANGE`, the transcript hash for **TH1** shall be the following:

1. `[GET_VERSION].*` (if issued)
2. `[VERSION].*` (if issued)
3. `[GET_CAPABILITIES].*` (if issued)
4. `[CAPABILITIES].*` (if issued)
5. `[NEGOTIATE_ALGORITHMS].*` (if issued)
6. `[ALGORITHMS].*` (if issued)
7. The specified certificate chain in DER format (i.e. KEY_EXCHANGE's request Param2)
8. `[KEY_EXCHANGE Request].*`
9. `[KEY_EXCHANGE Response].*`

The PSK-based key exchange scheme derives three keys from Handshake-Secret: requester_auth_key, Requester's finished_key, and Responder's finished_key.

To calculate bin_str1 that is used in deriving the Requester's requester_auth_key for requester_auth in `PSK_BASED_EXCHANGE` request, the transcript hash for **TH1** shall be the following:

1. `[GET_VERSION].*` (if issued)

2. `[VERSION].*` (if issued)
3. `[GET_CAPABILITIES].*` (if issued)
4. `[CAPABILITIES].*` (if issued)
5. `[NEGOTIATE_ALGORITHMS].*` (if issued)
6. `[ALGORITHMS].*` (if issued)
7. `[PSK_BASED_EXCHANGE Request].SPDMVersion`
8. `[PSK_BASED_EXCHANGE Request].RequestResponseCode`
9. `[PSK_BASED_EXCHANGE Request].Param1`
10. `[PSK_BASED_EXCHANGE Request].Param2`
11. `[PSK_BASED_EXCHANGE Request].requester_context`
12. `[PSK_BASED_EXCHANGE Request].opaque_PSK_data`

To calculate bin_str2 that is used in deriving the Responder's finished_key for `PSK_BASED_EXCHANGE` response, the transcript hash for **TH1** shall be the following:

1. `[GET_VERSION].*` (if issued)
2. `[VERSION].*` (if issued)
3. `[GET_CAPABILITIES].*` (if issued)
4. `[CAPABILITIES].*` (if issued)
5. `[NEGOTIATE_ALGORITHMS].*` (if issued)
6. `[ALGORITHMS].*` (if issued)
7. `[PSK_BASED_EXCHANGE Request].*`
8. `[PSK_BASED_EXCHANGE response].SPDMVersion`
9. `[PSK_BASED_EXCHANGE response].RequestResponseCode`
10. `[PSK_BASED_EXCHANGE response].Param1`
11. `[PSK_BASED_EXCHANGE response].Param2`
12. `[PSK_BASED_EXCHANGE response].responder_context`

To calculate bin_str1 that is used in deriving the Requester's finished_key for `PSK_BASED_FINISH` request, the transcript hash for **TH1** shall be the following:

1. `[GET_VERSION].*` (if issued)
2. `[VERSION].*` (if issued)
3. `[GET_CAPABILITIES].*` (if issued)
4. `[CAPABILITIES].*` (if issued)
5. `[NEGOTIATE_ALGORITHMS].*` (if issued)
6. `[ALGORITHMS].*` (if issued)
7. `[PSK_BASED_EXCHANGE Request].*`

8. `[PSK_BASED_EXCHANGE Response].*`
9. `[PSK_BASED_FINISH request].SPDMVersion`
10. `[PSK_BASED_FINISH request].RequestResponseCode`
11. `[PSK_BASED_FINISH request].Param1`
12. `[PSK_BASED_FINISH request].Param2`

9.3 TH2 Definition

If the Requester and Responder used `KEY_EXCHANGE` to exchange initial keying information, then **TH2** shall be the following:

1. `[GET_CAPABILITIES].*`
2. `[CAPABILITIES].*`
3. `[NEGOTIATE_ALGORITHMS].*`
4. `[ALGORITHMS].*`
5. The specified certificate chain in DER format (i.e. KEY_EXCHANGE's request Param2)
6. `[KEY_EXCHANGE Request].*`
7. `[KEY_EXCHANGE Response].*`
8. The specified certificate chain in DER format (i.e. FINISH Request's Param2). (Valid only in Mutual Authentication)
9. `[FINISH Request].*` (Valid only in Mutual Authentication)
10. `[FINISH Response].*`

If the Requester and Responder used `PSK_BASED_EXCHANGE` to exchange initial keying information, then **TH2** shall be the following:

1. `[GET_VERSION].*` (if issued)
2. `[VERSION].*` (if issued)
3. `[GET_CAPABILITIES].*` (if issued)
4. `[CAPABILITIES].*` (if issued)
5. `[NEGOTIATE_ALGORITHMS].*` (if issued)
6. `[ALGORITHMS].*` (if issued)
7. `[PSK_BASED_EXCHANGE Request].*`
8. `[PSK_BASED_EXCHANGE Response].*`
9. `[PSK_BASED_FINISH request].*`
10. `[PSK_BASED_FINISH response].*`

9.4 Key Schedule Major Secrets

The key schedule produces 4 major secrets:

- Request-Direction Handshake Secret
- Response-Direction Handshake Secret
- Request-Direction Data Secret
- Response-Direction Data Secret.

Each secret applies in a certain direction of transmission and only valid during a certain time frame. These four major secrets, each, will be used to derive their respective encryption key and salt to be used in the AEAD function as selected in the `ALGORITHMS` response.

9.4.1 Request-Direction Handshake Secret

This secret shall only be used during the session handshake phase and shall be applied to all requests after `KEY_EXCHANGE` up to and including `FINISH`.

9.4.2 Response-Direction Handshake Secret

This secret shall only be used during the session handshake phase and shall be applied to all responses after `KEY_EXCHANGE` up to and including `FINISH`.

9.4.3 Requester-Direction Data Secret

This secret shall be used for any data transmitted in the session, including but not limited to SPDM requests that are allowed to be issued post handshake. This secret shall only be applied for all data traveling from the Requester to the Responder.

9.4.4 Responder-Direction Data Secret

This secret shall be used for any data transmitted in the session, including but not limited to SPDM responses that are allowed to be issued post handshake. This secret shall only be applied for all data traveling from the Responder to the Requester.

9.5 Encryption Key and Salt Derivation

For each Key Schedule Major Secret, the following function shall be applied to obtain the encryption key and salt value.

```
EncryptionKey = HDKF-Expand(major-secret, bin_str_5, key_length);
Salt = HKDF-Expand(major-secret, bin_str_6, iv_length);

bin_str5 = BinConcat("key", NULL, key_length);
bin_str6 = BinConcat("iv", NULL, iv_length);
```

Both `key_length` and `iv_length` shall be the lengths associated with the selected AEAD algorithm in `ALGORITHMS` message.

9.6 Finish Key Derivation

This key shall be used to compute the verify data used in various SPDM messages. The key, `finished_key` is defined as follows:

```
finished_key = HKDF-Expand(handshake-secret, bin_str7, Hash.Length);
bin_str7 = BinConcat("finished", NULL, Hash.Length);
```

The handshake-secret shall either be Request-Direction Handshake Secret or Response-Direction Handshake secret.

9.7 Major Secret Update

The major secrets can be updated during an active session to avoid the overhead of closing down a session and recreating the session. This is achieved by issuing the `KEY_UPDATE` request.

The major secrets are rekeyed as a result of this. To compute the new secret for each new major data secret, the following algorithm shall be applied.

```
new_secret = HKDF-Expand(current_secret, bin_str8, Hash.Length);
bin_str8 = BinConcat("traffic upd", NULL, Hash.Length);
```

In computing the new secret, `current_secret` shall either be the current Requester-Direction Data Secret or Responder-Direction Data Secret. As a consequence of updating these secrets, new encryption keys and salts shall be derived from the new secrets and used immediately.

10 Record Layer

The record layer describes or defines the necessary SPDM data and data encoding to transmit application data over a secure session once the session handshake completes. Records form the basis and foundation for the transmission of any data over a secured session. This clause and subclauses describes in a generic way how the Responder and Requester can communicate with each other securely using records.

At a high level, a record is comprised of data sent in the clear, called associated data, and data sent encrypted. The record layer should be comprised of the following:

Field	Type	Description
Version	Associated Data	Identifies the version of the record layer. This can be used if the record layer format changes in the future.
Session ID	Associated Data	Identifies the session. Both Responder and Requester uses this information as binding to the respective set of secrets and derived session keys (i.e. session keys).
Length	Associated Data	Identifies the size of the entire record.
True Length	Encrypted	The true length of the payload.
Padding	Encrypted	Padding to obfuscate the the True Length.
Application Data	Encrypted	The application specific data.
MAC	Message Authentication Code	This provides authentication and integrity of the record

The need for the padding field is to obfuscate the true size of the application data to prevent side channel attacks or attacks derived from knowledge of the length of the application data. Each record should randomize the amount of padding needed.

10.1 Record Protection

SPDM utilizes Authenticated Encryption with Associated Data (AEAD) cipher algorithms in much the same way that TLS 1.3 does to protect the record layer. AEAD algorithms provide both encryption and message authentication. Each algorithm specifies the details such as the size of the nonce, the position and length of the MAC and many other factors to ensure a strong cryptographic algorithm.

AEAD functions shall provide the following functions and comply with the requirements defined in RFC5116:

```
AEAD_Encrypt(encryption_key, nonce, associated_data, clear_text);
AEAD_Decrypt(encryption_key, nonce, associated_data, cipher_text);
```

where:

- `encryption_key` is the derived encryption key for the respective direction. See Key Schedule for details.
- `nonce` is the nonce. See blah for details on nonce computation.
- `associated_data` is the associated data.
- `clear_text` is the data to encrypt.
- `cipher_text` is the data to decrypt.

The function, `AEAD_Encrypt`, fully encrypts the `clear_text`, computes the MAC across both the `associated_data` and `clear_text` and produces the `cipher_text` which includes the MAC as well. The `AEAD_Decrypt` function fully decrypts the `cipher_text`, verifies the MAC and if validation is successful, produces the original `clear_text`.

10.1.1 Per-Record Nonce Derivation

The nonce used at the record layer shall be bound to a single record. The transport protocol is responsible for retrying records that failed at that layer and is outside the scope of this specification.

The nonce shall never be transmitted in the record. This means that both Responder and Requester must internally track the nonce. In order to ensure proper tracking, the Responder shall follow the nonce derivation schedule laid henceforth.

Internally, before the creation of the first record in the session, both Responder and Requester shall start with a 64-bit sequence number with a value of zero. For each record, both SPDM endpoint shall follow the recipe as prescribed:

1. Zero Extend the Sequence Number to `iv_length` according to the selected AEAD cipher suite in `ALGORITHMS` messages.
2. Perform a bitwise XOR of the zero-extended Sequence Number with the respective salt derived in the Key Schedule clauses.
3. The output of the above step is the per-record nonce.
4. Increment the sequence number by a value of one for the next record.

Because different secrets are used for different directions of data transmission, each endpoint would have to track two sequence numbers: one for the reception and the other for the transmission in order to properly process the record.

Lastly, when a `KEY_UPDATE` occurs, the sequence number shall reset to 0 before sending the first record using the new session keys.

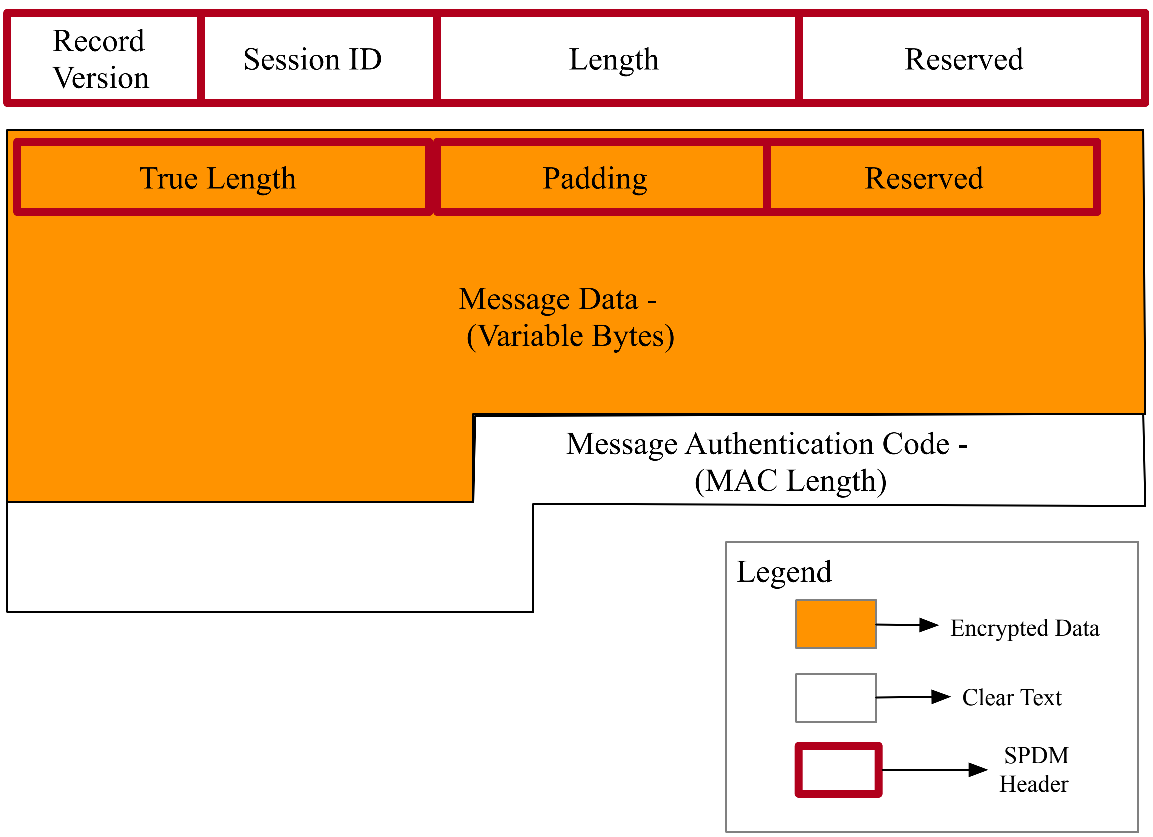
10.2 Record Retries

A retry of a record shall be defined as the reuse of a sequence number. While SPDM requests retry are permitted, record retries shall be expressly prohibited. SPDM recognizes that, in general, transport layers may retransmit the record but that is outside the scope of this specification.

10.3 Record Format Finalization

In generally, an AEAD algorithm determines the format of both the encrypted `clear_text` and the MAC in `cipher_text`. Some AEAD algorithms allow flexibility to where the MAC resides. For such algorithms, the Generic Record Format Figure illustrates a possible solution.

Generic Record Format



A record shall contain the complete cipher text as produced by a single invocation of `AEAD_Encrypt` using the appropriate encryption key for the given direction of transmission, the appropriate per-record nonce and the selected AEAD Cipher Suite in `ALGORITHMS`.

The actual contents and format of the `associated_data` and `clear_text` should be defined by the transport protocol but that is outside the scope of this specification.

11 ANNEX A (informative)

This specification heavily models TLS 1.3. TLS 1.3 and consequently this specification assumes the transport layer(s) provides these capabilities or attributes:

- Reliability in transmission and reception of data
- Transmission of data is either in order or the order of data can be reconstructed at reception.

While not all transports are created equal, if a transport cannot meet the above capabilities, adoption of SPDM is still possible. In these transports, this specification recommends DTLS 1.3 which at the time of this specification is still in draft form.

12 ANNEX B - Leaf certificate example

Certificate:

```
    Data:
        Version: 3 (0x2)
        Serial Number: 8 (0x8)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=CA, ST=NC, L=city, O=ACME, OU=ACME Devices, CN=CA
        Validity
            Not Before: Jan  1 00:00:00 1970 GMT
            Not After : Dec 31 23:59:59 9999 GMT
        Subject: C=US, ST=NC, O=ACME Widget Manufacturing, OU=ACME Widget Manufacturing Unit, CN=w0123456789
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:ba:67:47:72:78:da:28:81:d9:81:9b:db:88:03:
                    e1:10:a4:91:b8:48:ed:6b:70:3c:ec:a2:68:a9:3b:
                    5f:78:fc:ae:4a:d1:1c:63:76:54:a8:40:31:26:7f:
                    ff:3e:e0:bf:95:5c:4a:b4:6f:11:56:ca:c8:11:53:
                    23:e1:1d:a2:7a:a5:f0:22:d8:b2:fb:43:da:dd:bd:
                    52:6b:e6:a5:3f:0f:3b:60:b8:74:db:56:08:d9:ee:
                    a0:30:4a:03:21:1e:ee:60:ad:e4:00:7a:6e:6b:32:
                    1c:28:7e:9c:e8:c3:54:db:63:fd:1f:d1:46:20:9e:
                    ef:80:88:00:5f:25:db:cf:43:46:c6:1f:50:19:7f:
                    98:23:84:38:88:47:5d:51:8e:11:62:6f:0f:28:77:
                    a7:20:0e:f3:74:27:82:70:a7:96:5b:1b:bb:10:e7:
                    95:62:f5:37:4b:ba:20:4e:3c:c9:18:b2:cd:4b:58:
                    70:ab:a2:bc:f6:2f:ed:2f:48:92:be:5a:cc:5c:5e:
                    a8:ea:9d:60:e8:f8:85:7d:c0:0d:2f:6a:08:74:d1:
                    2f:e8:5e:3d:b7:35:a6:1d:d2:a6:04:99:d3:90:43:
                    66:35:e1:74:10:a8:97:3b:49:05:51:61:07:c6:08:
                    01:1c:dc:a8:5f:9e:30:97:a8:18:6c:f9:b1:2c:56:
                    e8:67
                Exponent: 65537 (0x10001)
                X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Subject Alternative Name:
                otherName:1.3.6.1.4.1.412.274.1;UTF8STRING:ACME:WIDGET:0123456789
            Signature Algorithm: ecdsa-with-SHA256
            Signature Value:
                30:45:02:21:00:fc:8f:b0:ad:6f:2d:c3:2a:7e:92:6d:29:1d:
                c7:fc:0d:48:b0:c6:39:5e:c8:76:d6:40:9a:12:46:c3:39:0e:
                36:02:20:1a:ea:3a:59:ca:1e:bc:6d:6e:61:79:af:a2:05:7c:
```

```
            7d:da:41:a9:45:6d:cb:04:49:43:e6:0b:a8:8d:cd:da:e
```

12.1 Change log

Version	Date	Description
1.1.0a	2019-10-30	

12.2 Bibliography

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