



Document Identifier: DSP0274

Date: 2023-04-05

Version: 1.3.0WIP99

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 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 This document is available for public review and comment until superseded.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: <https://www.dmtf.org/standards/feedback>

Supersedes: 1.2.1

Document Class: Normative

Document Status: Work in Progress

Document Language: en-US

Copyright Notice

Copyright © 2023 DMTF. All rights reserved.

- 11 DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. Members and non-members may reproduce DMTF specifications and documents, provided that correct attribution is given. As DMTF specifications may be revised from time to time, the particular version and release date should always be noted.
- 12 Implementation of certain elements of this standard or proposed standard may be subject to third-party patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations to users of the standard as to the existence of such rights and is not responsible to recognize, disclose, or identify any or all such third-party patent rights, owners, or claimants, nor for any incomplete or inaccurate identification or disclosure of such rights, owners, or claimants. DMTF shall have no liability to any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize, disclose, or identify any such third-party patent rights, or for such party's reliance on the standard or incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any party implementing such standard, whether such implementation is foreseeable or not, nor to any patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a patent owner for such implementations.
- 13 For information about patents held by third parties which have notified DMTF that, in their opinion, such patent may relate to or impact implementations of DMTF standards, visit <https://www.dmtf.org/about/policies/disclosures.php>.
- 14 This document's normative language is English. Translation into other languages is permitted.

1 Foreword	8
1.1 Acknowledgments	8
2 Introduction	10
2.1 Advice	10
2.2 Conventions	10
2.2.1 Document conventions	10
2.2.2 Reserved and unassigned values	10
2.2.3 Byte ordering	10
2.2.3.1 Hash byte order	11
2.2.3.2 Encoded ASN.1 byte order	11
2.2.3.3 Octet string byte order	11
2.2.3.4 Signature byte order	11
2.2.3.4.1 ECDSA signatures byte order	12
2.2.3.4.2 SM2 signatures byte order	12
2.2.4 SPDM data type conventions	12
2.2.4.1 SPDM data types	12
2.2.4.2 Integers	12
2.2.5 Version encoding	13
2.2.6 Notations	14
2.2.7 Text or string encoding	15
2.2.8 Deprecated material	15
3 Scope	16
4 Normative references	17
5 Terms and definitions	19
6 Symbols and abbreviated terms	24
7 SPDM message exchanges	25
7.1 Security capability discovery and negotiation	25
7.2 Identity authentication	25
7.2.1 Identity provisioning	26
7.2.1.1 Certificate models	26
7.2.1.1.1 Device certificate model	27
7.2.1.1.2 Alias certificate model	28
7.2.1.1.3 Generic certificate model	28
7.2.2 Raw public keys	29
7.2.3 Runtime authentication	29
7.3 Firmware and configuration measurement	30
7.4 Secure sessions	30
7.5 Mutual authentication overview	30
7.6 Multiple asymmetric key support	31
7.7 Custom environments	31
7.8 Notification overview	31

8	SPDM messaging protocol	32
8.1	SPDM connection model	34
8.2	SPDM bits-to-bytes mapping	34
8.3	Generic SPDM message format	35
8.3.1	SPDM version	36
8.4	SPDM request codes	36
8.5	SPDM response codes	38
8.6	SPDM request and response code issuance allowance	39
8.7	Concurrent SPDM message processing	41
8.8	Requirements for Requesters	42
8.9	Requirements for Responders	42
8.10	Transcript and transcript hash calculation rules	43
9	Timing requirements	44
9.1	Timing measurements	44
9.2	Timing parameters	44
9.3	Timing specification table	45
10	SPDM messages	48
10.1	Capability discovery and negotiation	48
10.1.1	Negotiated state preamble	49
10.2	GET_VERSION request and VERSION response messages	49
10.3	GET_CAPABILITIES request and CAPABILITIES response messages	52
10.3.1	Supported algorithms block	61
10.4	NEGOTIATE_ALGORITHMS request and ALGORITHMS response messages	62
10.4.1	Connection behavior after VCA	76
10.4.2	Multiple asymmetric key negotiation	77
10.4.3	Multiple asymmetric key use for Responder authentication	77
10.4.4	Multiple asymmetric key use for Requester authentication	77
10.4.5	Multiple asymmetric key connection	78
10.5	Responder identity authentication	79
10.6	Requester identity authentication	81
10.6.1	Certificates and certificate chains	81
10.7	GET_DIGESTS request and DIGESTS response messages	82
10.8	GET_CERTIFICATE request and CERTIFICATE response messages	86
10.8.1	Mutual authentication requirements for GET_CERTIFICATE and CERTIFICATE messages	89
10.8.2	SPDM certificate requirements and recommendations	90
10.8.2.1	Extended Key Usage authentication OIDs	92
10.8.2.2	SPDM Non-Critical Certificate Extension OID	92
10.8.2.2.1	Hardware identity OID	93
10.8.2.2.2	Mutable certificate OID	94
10.9	CHALLENGE request and CHALLENGE_AUTH response messages	94
10.9.1	CHALLENGE_AUTH signature generation	97
10.9.2	CHALLENGE_AUTH signature verification	98
10.9.2.1	Request ordering and message transcript computation rules for M1 and M2	99

10.9.3 Basic mutual authentication	101
10.9.3.1 Mutual authentication message transcript	102
10.10 Firmware and other measurements.	103
10.11 GET_MEASUREMENTS request and MEASUREMENTS response messages.	104
10.11.1 Measurement block	108
10.11.1.1 DMTF specification for the Measurement field of a measurement block	109
10.11.1.1.1 Measurement manifest	109
10.11.1.1.2 Hash-extend measurements.	109
10.11.1.2 Device mode field of a measurement block	112
10.11.1.3 Manifest format for a measurement block.	113
10.11.2 MEASUREMENTS signature generation	113
10.11.3 MEASUREMENTS signature verification	115
10.12 ERROR response message	116
10.12.1 Standards body or vendor-defined header	121
10.13 RESPOND_IF_READY request message format	122
10.14 VENDOR_DEFINED_REQUEST request message	123
10.15 VENDOR_DEFINED_RESPONSE response message	124
10.15.1 VendorDefinedReqPayload and VendorDefinedRespPayload defined by DMTF specifications	125
10.16 KEY_EXCHANGE request and KEY_EXCHANGE_RSP response messages	126
10.16.1 Session-based mutual authentication	133
10.16.1.1 Specify Requester certificate for session-based mutual authentication	133
10.17 FINISH request and FINISH_RSP response messages	134
10.17.1 Transcript and transcript hash calculation rules for KEY_EXCHANGE	136
10.18 PSK_EXCHANGE request and PSK_EXCHANGE_RSP response messages	139
10.19 PSK_FINISH request and PSK_FINISH_RSP response messages	145
10.20 HEARTBEAT request and HEARTBEAT_ACK response messages.	146
10.20.1 Heartbeat additional information	147
10.21 KEY_UPDATE request and KEY_UPDATE_ACK response messages	147
10.21.1 Session key update synchronization.	148
10.21.2 KEY_UPDATE transport allowances.	151
10.22 GET_ENCAPSULATED_REQUEST request and ENCAPSULATED_REQUEST response messages.	154
10.22.1 Encapsulated request flow	154
10.22.2 Optimized encapsulated request flow	154
10.22.3 Triggering GET_ENCAPSULATED_REQUEST	158
10.22.4 Additional constraints	159
10.23 DELIVER_ENCAPSULATED_RESPONSE request and ENCAPSULATED_RESPONSE_ACK response messages.	159
10.23.1 Additional information	161
10.23.2 Allowance for encapsulated requests	161
10.23.3 Certain error handling in encapsulated flows	162
10.23.3.1 Response not ready	162

10.23.3.2 Timeouts	162
10.24 END_SESSION request and END_SESSION_ACK response messages	163
10.25 Certificate provisioning	164
10.25.1 GET_CSR request and CSR response messages	164
10.25.2 SET_CERTIFICATE request and SET_CERTIFICATE_RSP response messages	167
10.26 Large SPDM message transfer mechanism	169
10.26.1 CHUNK_SEND request and CHUNK_SEND_ACK response message	169
10.26.2 CHUNK_GET request and CHUNK_RESPONSE response message	172
10.26.3 Additional chunk transfer requirements	174
10.27 Key configuration	175
10.27.1 GET_KEY_PAIR_INFO request and KEY_PAIR_INFO response	176
10.27.2 SET_KEY_PAIR_INFO request and SET_KEY_PAIR_INFO_ACK response	179
10.27.3 Key pair ID modification error handling	181
10.28 Event mechanism	181
10.28.1 GET_SUPPORTED_EVENT_TYPES request and SUPPORTED_EVENT_TYPES response message	184
10.28.1.1 Event group format additional information	186
10.28.2 SUBSCRIBE_EVENT_TYPES request and SUBSCRIBE_EVENT_TYPES_ACK response message	186
10.28.2.1 Additional subscription list information	188
10.28.3 SEND_EVENT request and EVENT_ACK response message	188
10.28.4 Event Instance ID	190
10.29 GET_ENDPOINT_INFO request and ENDPOINT_INFO response messages	190
10.29.1 ENDPOINT_INFO signature generation	194
10.29.2 ENDPOINT_INFO signature verification	194
10.30 Measurement extension log mechanism	195
10.30.1 GET_MEASUREMENT_EXTENSION_LOG request and MEASUREMENT_EXTENSION_LOG response messages	196
10.30.2 DMTF Measurement Extension Log Format	197
10.30.3 Example: Verifying Measurement Extension Log Against Hash-Extend Measurement	199
11 Session	201
11.1 Session handshake phase	201
11.2 Application phase	202
11.3 Session termination phase	202
11.4 Simultaneous active sessions	203
11.5 Records and session ID	203
12 Key schedule	204
12.1 DHE secret computation	207
12.2 Transcript hash in key derivation	207
12.3 TH1 definition	208
12.4 TH2 definition	208
12.5 Key schedule major secrets	209
12.5.1 Request-direction handshake secret	209

12.5.2 Response-direction handshake secret	209
12.5.3 Requester-direction data secret	209
12.5.4 Responder-direction data secret	209
12.6 Encryption key and IV derivation	210
12.7 finished_key derivation	211
12.8 Deriving additional keys from the Export Master Secret	211
12.9 Major secrets update	211
13 Application data	212
13.1 Nonce derivation	212
14 General opaque data format	213
15 Signature generation	215
15.1 Signing algorithms in extensions	216
15.2 RSA and ECDSA signing algorithms	217
15.3 EdDSA signing algorithms	217
15.3.1 Ed25519 sign	217
15.3.2 Ed448 sign	217
15.4 SM2 signing algorithm	217
15.5 Signature algorithm references	218
16 Signature verification	220
16.1 Signature verification algorithms in extensions	220
16.2 RSA and ECDSA signature verification algorithms	221
16.3 EdDSA signature verification algorithms	221
16.3.1 Ed25519 verify	221
16.3.2 Ed448 verify	221
16.4 SM2 signature verification algorithm	222
17 General ordering rules	223
18 DMTF event types	224
18.1 Event type details	224
18.1.1 Event Lost	224
18.1.2 Measurement changed event	225
18.1.3 Measurement pre-update event	225
18.1.4 Certificate changed event	226
19 ANNEX A (informative) TLS 1.3	228
20 ANNEX B (informative) Device certificate example	229
21 ANNEX C (informative) OID reference	231
22 ANNEX D (informative) variable name reference	232
23 ANNEX E (informative) change log	234
23.1 Version 1.0.0 (2019-10-16)	234
23.2 Version 1.1.0 (2020-07-15)	234
23.3 Version 1.2.0 (2021-11-01)	234
23.4 Version 1.3.0 (2023-04-05)	237
24 Bibliography	242

1 Foreword

The *Security Protocols and Data Models (SPDM)* 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 [DMTF](#).

This version supersedes version 1.2 and its errata versions. For a list of the changes, see [ANNEX E \(informative\) change log](#).

1.1 Acknowledgments

The DMTF acknowledges the following individuals for their contributions to this document:

Contributors:

- Richelle Ahlvers — Broadcom Inc.
- Jeff Andersen — Google
- Lee Ballard — Dell Technologies
- Steven Bellock — NVIDIA Corporation
- Heng Cai — Alibaba Group
- Patrick Caporale — Lenovo
- Yu-Yuan Chen — Intel Corporation
- Andrew Draper — Intel Corporation
- Nigel Edwards — Hewlett Packard Enterprise
- Daniil Egranov — Arm Limited
- Philip Hawkes — Qualcomm Inc.
- Brett Henning — Broadcom Inc.
- Jeff Hilland — Hewlett Packard Enterprise
- Yi Hou — Microchip
- Guerney Hunt — IBM
- Yuval Itkin — NVIDIA Corporation
- Theo Koulouris — Hewlett Packard Enterprise
- Raghupathy Krishnamurthy — NVIDIA Corporation
- Benjamin Lei — Lenovo
- Luis Luciani — Hewlett Packard Enterprise
- Masoud Manoo — Lenovo
- Donald Matthews — Advanced Micro Devices, Inc.
- Mahesh Natsu — Intel Corporation

- Chandra Nelogal — Dell Technologies
- Edward Newman — Hewlett Packard Enterprise
- Alexander Novitskiy — Intel Corporation
- Jim Panian — Qualcomm Inc.
- Scott Phuong — Cisco Systems Inc., Axiado Corporation
- Jeffrey Plank — Microchip
- Viswanath Ponnuru — Dell Technologies
- Lohith Rangappa — Marvell Technology, Inc.
- Xiaoyu Ruan — Intel Corporation
- Nitin Sarangdhar — Intel Corporation
- Vidya Satyamsetti — Google
- Hemal Shah — Broadcom Inc.
- Yoni Shternhell — Western Digital Technologies, Inc.
- Srikanth Varadarajan — Intel Corporation
- Peng Xiao — Alibaba Group
- Qing Yang — Alibaba Group
- Jiewen Yao — Intel Corporation

2 Introduction

The *Security Protocol and Data Model (SPDM) Specification* defines [messages](#), data objects, and sequences for performing message exchanges over a variety of transport and physical media. The description of message exchanges includes [authentication](#) and provisioning of hardware identities, measurement for firmware identities, session key exchange protocols to enable confidentiality with integrity-protected data communication, and other related capabilities. The SPDM enables efficient access to low-level security capabilities and operations. Other mechanisms, including non-DMTF-defined mechanisms, can use the SPDM.

2.1 Advice

The authors recommend readers visit tutorial and education materials under [Security Protocols and Data Models](#) and [Platform Management Communications Infrastructure \(PMCI\)](#) on the DMTF website prior to or during the reading of this specification to help understand this specification.

2.2 Conventions

The following conventions apply to all SPDM specifications.

2.2.1 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.

2.2.2 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, field values marked as Reserved shall be written as zero (0), ignored when read, not modified, and not interpreted as an error if not zero, and used in transcript hash calculations as is.

2.2.3 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).

34 2.2.3.1 Hash byte order

35 For fields or values containing a digest or hash, SPDM preserves the byte order of the digest as the specification of a given hash algorithm defines. SPDM views these digests, simply, as a string of octets where the first byte is the leftmost byte of the digest, the second byte is the second leftmost byte, the third byte is the third leftmost byte, and this pattern continues until the last byte of the digest. Thus, the byte order for SPDM digests or hashes is: the first byte is placed at the lowest offset in the field or value, the second byte is placed at the second lowest offset, the third byte is placed at the third lowest offset in the field or value and this pattern continues until the last byte.

36 For example, in [FIPS 180-4](#), a SHA 256 hash is the concatenation of eight 32-bit words where each word is in *big endian* order, but the order of words does not have any endianness associated with it. SPDM simply views this 256-bit digest as a string of octets that is 32 bytes in size where the first byte is the value at $H_0[31:24]$ of the final digest, the second byte is the value at $H_0[23:16]$, the third byte is the value at $H_0[15:8]$, the fourth byte is the value at $H_0[7:0]$, the fifth bytes is the value at $H_1[31:24]$, and this pattern continues until the last byte, which is the value at $H_7[7:0]$, where the FIPS 180-4 specification defines H_0 , H_1 , and H_7 .

37 2.2.3.2 Encoded ASN.1 byte order

38 For fields or values containing DER, CER, or BER encoded data, SPDM preserves the byte order as described in [X.690](#) specification. SPDM views a DER, CER, or BER encoded data as simply a string of octets where the first byte is the leftmost byte of Figure 1 or Figure 2 in the [X.690](#) specification, the second byte is the second leftmost byte, the third byte is the third leftmost byte, and this pattern continues until the last byte. The first byte is also called either the Identifier octet or the Leading identifier octet. The X.690 specification defines Figure 1, Figure 2, and identifier octets. When populating a DER, CER, or BER encoded data in SPDM fields, the first byte is placed in the lowest address, the second byte is placed in the second lowest offset, the third byte is placed in the third lowest offset in the field or value and this pattern continues until the last byte.

39 2.2.3.3 Octet string byte order

40 A string of octets is conventionally written from left to right. Also by convention, byte zero of the octet string shall be the leftmost byte of the octet, byte 1 of the octet string shall be the second leftmost byte of the octet, and this pattern shall continue until the very last byte. When placing an octet string into an SPDM field, the i^{th} byte of the octet string shall be placed in the i^{th} offset of that field.

41 For example, if placing an octet stream consisting of "0xAA 0xCB 0x9F 0xD8" into `DMTFSpecMeasurementValue` field, then offset 0 (the lowest offset) of `DMTFSpecMeasurementValue` will contain 0xAA, offset 1 of `DMTFSpecMeasurementValue` will contain 0xCB, offset 2 of `DMTFSpecMeasurementValue` will contain 0x9F, and offset 3 of `DMTFSpecMeasurementValue` will contain 0xD8.

42 2.2.3.4 Signature byte order

43 For fields or values containing a signature, SPDM attempts to preserve the byte order of the signature as the

specification of a given signature algorithm defines. Most signature specifications define a string of octets as the format of the signature, and others may explicitly state the endianness such as in the specification for [Edwards-Curve Digital Signature Algorithm](#). Unless otherwise specified, the byte order of a signature for a given signature algorithm shall be [octet string byte order](#).

44 2.2.3.4.1 ECDSA signatures byte order

45 [FIPS PUB 186-5](#) defines `r`, `s`, and ECDSA signature to be `(r, s)`, where `r` and `s` are just integers. For ECDSA signatures, excluding SM2, in SPDM, the signature shall be the concatenation of `r` and `s`. The size of `r` shall be the size of the selected curve. Likewise, the size of `s` shall be the size of the selected curve. See `BaseAsymAlgo` in `NEGOTIATE_ALGORITHMS` for the size of `r` and `s`. The byte order for `r` and `s` shall be big-endian order. When placing ECDSA signatures into an SPDM signature field, `r` shall come first, followed by `s`.

46 2.2.3.4.2 SM2 signatures byte order

47 [GB/T 32918.2-2016](#) defines `r` and `s` and SM2 signatures to be `(r, s)`, where `r` and `s` are just integers. The size of `r` and `s` shall each be 32 bytes. To form an SM2 signature, `r` and `s` shall be converted to an octet stream according to [GB/T 32918.2-2016](#) and [GB/T 32918.1-2016](#) with a target length of 32 bytes. Let the resulting octet string of `r` and `s` be called `SM2_R` and `SM2_S` respectively. The final SM2 signature shall be the concatenation of `SM2_R` and `SM2_S`. When placing SM2 signatures into an SPDM signature field, the SM2 signature byte order shall be [octet string byte order](#).

48 2.2.4 SPDM data type conventions

49 2.2.4.1 SPDM data types

50 [Table 1 — SPDM data types](#) lists the abbreviations and descriptions for common data types that SPDM message fields and data structure definitions use. These definitions follow [DSP0240](#).

51 **Table 1 — SPDM data types**

Data type	Interpretation
<code>ver8</code>	Eight-bit encoding of the SPDM version number. Version encoding defines the encoding of the version number.
<code>bitfield8</code>	Byte with 8-bit fields.
<code>bitfield16</code>	Two-byte word with 16-bit fields.

52 2.2.4.2 Integers

53 Unless noted otherwise, integers shall be unsigned.

2.2.5 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 <code>SPDMVersion</code> field in the SPDM message header.	Protocol modification breaks backward compatibility.
Minor	Minor version field in the <code>SPDMVersion</code> 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 or other previous minor versions. Whether an endpoint supports inter-operation with previous minor versions of the SPDM specification is an implementation-specific decision.

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`.

This specification considers two minor versions to be interoperable when it is possible for an implementation that is conformant to a higher minor version number to also communicate with an implementation that is conformant to a lower minor version number with minimal differences in operation. In such a case, the following rules apply:

- Both endpoints shall use the same lower version number in the `SPDMVersion` field for all messages.
- Functionality shall be limited to what the lower minor version of the SPDM specification defines.
- Computations and other operations between different minor versions of the Secured Messages using SPDM specification should remain the same, unless security issues of lower minor versions are fixed in higher minor versions and the fixes require change in computations or other operations. These differences are dependent on the value in the `SPDMVersion` field in the message.
- In a newer minor version of the SPDM specification, a given message can be longer, bit fields and enumerations can contain new values, and reserved fields can gain functionality. Existing numeric and bit fields retain their existing definitions.

For details on the version agreement process, see [GET_VERSION request and VERSION response messages](#). The

detailed version encoding that the `VERSION` response message returns contains an additional byte that indicates specification bug fixes or development versions. See [Table 9 — Successful `VERSION` response message format](#).

64 2.2.6 Notations

65 SPDM specifications use the following notations:

Notation	Description
<code>Concatenate()</code>	The concatenation function <code>Concatenate(a, b, ..., z)</code> , where the first entry occupies the least-significant bits and the last entry occupies the most-significant bits.
<code>M:N</code>	In field descriptions, this notation typically represents a range of byte offsets starting from byte <code>M</code> and continuing to and including byte <code>N</code> ($M \leq N$). The lowest offset is on the left. The highest offset is on the right.
<code>[4]</code>	Square brackets around a number typically indicate a bit offset. Bit offsets are zero-based values. That is, the least significant bit (<code>[LSb]</code>) offset = 0.
<code>[M:N]</code>	A range of bit offsets where <code>M</code> is greater than or equal to <code>N</code> . The most significant bit is on the left, and the least significant bit is on the right.
<code>1b</code>	A lowercase <code>b</code> after a number consisting of <code>0</code> s and <code>1</code> s indicates that the number is in binary format.
<code>0x12A</code>	Hexadecimal, as indicated by the leading <code>0x</code> .
<code>N+</code>	Variable-length byte range that starts at byte offset <code>N</code> .
<code>{ Payload }</code>	Used mostly in figures, this notation indicates that the payload specified in the enclosing curly brackets is encrypted and/or authenticated by the keys derived from one or more major secrets. The specific secret used is described throughout this specification. For example, <code>{ HEARTBEAT }</code> shows that the Heartbeat message is encrypted and/or authenticated by the keys derived from one or more major secrets.
<code>{ Payload }:::[Sx]</code>	Used mostly in figures, this notation indicates that the payload specified in the enclosing curly brackets is encrypted and/or authenticated by the keys derived from major Secret <code>X</code> . For example, <code>{ HEARTBEAT }:::[S2]</code> shows that the Heartbeat message is encrypted and/or authenticated by the keys derived from major secret <code>S2</code> .
<code>[\${message_name}] . \${field_name}</code>	Used to indicate a field in a message. <ul style="list-style-type: none"> <code>\${message_name}</code> is the name of the request or response message. <code>\${field_name}</code> is the name of the field in the request or response message. An asterisk (<code>*</code>) instead of a field name means all fields in that message except for any conditional fields that are empty (as for example <code>KEY_EXCHANGE . OpaqueData</code>).

2.2.7 Text or string encoding

When a value is indicated as a text or string data type, the encoding for the text or string shall be an array of contiguous *bytes* whose values are ordered. The first byte of the array resides at the lowest offset, and the last byte of the array is at the highest offset. The order of characters in the array shall be such that the leftmost character of the string is placed at the first byte in the array, the second leftmost character is placed in the second byte, and so forth until the last character is placed in the last byte.

Each byte in the array shall be the numeric value that represents that character, as [ASCII — ISO/IEC 646:1991](#) defines.

[Table 2 — "spdm" encoding example](#) shows an encoding example of the string "spdm":

Table 2 — "spdm" encoding example

Offset	Character	Value
0	s	0x73
1	p	0x70
2	d	0x64
3	m	0x6D

2.2.8 Deprecated material

Deprecated material is not recommended for use in new development efforts. Existing and new implementations can use this material, but they shall move to the favored approach as soon as possible. Implementations can implement any deprecated elements as required by this document to achieve backward compatibility. Although implementations can use deprecated elements, they are directed to use the favored elements instead.

The following typographical convention indicates deprecated material:

DEPRECATED

Deprecated material appears here.

DEPRECATED

In places where this typographical convention cannot be used (for example, in tables or figures), the "DEPRECATED" label is used alone.

78 **3 Scope**

- 79 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 measurements.
- 80 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.

81

4 Normative references

82

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 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 - 2021 (9th 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://dmtof.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
- IETF RFC 9147, *The Datagram Transport Layer Security (DTLS) Protocol Version 1.3*, April 2022
- IETF RFC 2986, *PKCS #10: Certification Request Syntax Specification*, November 2000
- IETF RFC 4716, *The Secure Shell (SSH) Public Key File Format*, November 2006
- IETF RFC 5234, *Augmented BNF for Syntax Specifications: ABNF*, January 2008
- IETF RFC 5280, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, May 2008
- IETF RFC 7250, *Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*, June 2014
- IETF RFC 7919, *Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)*, August 2016
- IETF RFC 8017, *PKCS #1: RSA Cryptography Specifications Version 2.2*, November, 2016
- IETF RFC 8032, *Edwards-Curve Digital Signature Algorithm (EdDSA)*, January 2017
- IETF RFC 8446, *The Transport Layer Security (TLS) Protocol Version 1.3*, August 2018
- *USB Authentication Specification Rev 1.0 with ECN and Errata through January 7, 2019*
- *TCG Algorithm Registry, Family "2.0", Level 00 Revision 01.32*, June 25, 2020

- NIST Special Publication 800-38D, *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, November 2007
- IETF RFC 8439, *ChaCha20 and Poly1305 for IETF Protocols*, June 2018
- IETF RFC 8998, *ShangMi (SM) Cipher Suites for TLS 1.3*, March 2021
- GB/T 32918.1-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 1: General*, August 2016
- GB/T 32918.2-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 2: Digital signature algorithm*, August 2016
- GB/T 32918.3-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 3: Key exchange protocol*, August 2016
- GB/T 32918.4-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 4: Public key encryption algorithm*, August 2016
- GB/T 32918.5-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 5: Parameter definition*, August 2016
- GB/T 32905-2016, *Information security technology—SM3 cryptographic hash algorithm*, August 2016
- GB/T 32907-2016, *Information security technology—SM4 block cipher algorithm*, August 2016
- **ASN.1 — ISO-822-1-4, DER — ISO-8825-1**
 - ITU-T X.680, X.681, X.682, X.683, X.690, 08/2015
- **ASCII — ISO/IEC 646:1991**, 09/1991
- **ECDSA**
 - Section 6, The Elliptic Curve Digital Signature Algorithm (ECDSA) in *FIPS PUB 186-5 Digital Signature Standard (DSS)*
 - NIST SP 800-186 Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters
 - IETF RFC 6979, *Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)*, August 2013
- **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*

5 Terms and definitions

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

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

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

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

The terms that [DSP0004](#), [DSP0223](#), [DSP0236](#), [DSP0239](#), [DSP0275](#), and [DSP1001](#) define also apply to this document.

This specification uses these terms:

Term	Definition
alias certificate	Certificate that is dynamically generated by the component or component firmware.
application data	Data that is specific to the application and whose definition and format is outside the scope of this specification. Application data usually exists at the application layer, which is, in general, the layer above SPDM and the transport layer. Examples of data that could be application data include: messages carried as DMTF MCTP payloads; Internet traffic; PCIe transaction layer packets (TLPs); camera images and video (MIPI CSI-2 packets); video display stream (MIPI DSI-2 packets); and touchscreen data (MIPI I3C Touch).
authentication initiator	Endpoint that initiates the authentication process by challenging another endpoint.
authentication	Process of determining whether an entity is who or what it claims to be.
byte	Eight-bit quantity. Also known as an <i>octet</i> .
certificate authority (CA)	Trusted entity that issues certificates.
certificate chain	Typically a series of two or more certificates. Each certificate is signed by the preceding certificate in the chain.

Term	Definition
certificate	Digital form of identification that provides information about an entity and certifies ownership of a particular asymmetric key-pair.
component	Physical device, contained in a single package. A "component" may also refer to a functional block implemented in hardware, firmware, and/or software.
device certificate	Certificate that contains information that identifies the component. Can be a leaf certificate or an intermediate certificate .
device	Physical entity such as a network controller or a fan.
DMTF	Formerly known as the Distributed Management Task Force, 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.
encapsulated request	A request embedded into an <code>ENCAPSULATED_REQUEST</code> or <code>ENCAPSULATED_RESPONSE_ACK</code> response message to allow the Responder to issue a request to a Requester. See GET_ENCAPSULATED_REQUEST request and ENCAPSULATED_REQUEST response messages .
generic certificate	A certificate, for use in certificate slots 1 or greater, that has minimal SPDM requirements to allow for numerous use cases that the vendor, standards body, or user defines.
endpoint	Logical entity that communicates with other endpoints over one or more transport protocols.
event notifier	An SPDM endpoint that is capable of sending asynchronous notifications using SPDM event mechanisms. See Event mechanism .
event recipient	An SPDM endpoint that is capable of receiving asynchronous notifications using SPDM event mechanisms. See Event mechanism .
intermediate certificate	Certificate that is neither a root certificate nor a leaf certificate.
invasive debug mode	A device mode that enables debug access that might expose or allow modification of firmware, hardware, or settings that can access (read or write) security keys, states, and contexts of the device. A device should not be trusted when it is operating in this mode.
large SPDM message	An SPDM message that is greater than the <code>DataTransferSize</code> of the receiving SPDM endpoint or greater than the transmit buffer size of the sending SPDM endpoint.
large SPDM request message	A large SPDM message that is an SPDM request.
large SPDM response message	A large SPDM message that is an SPDM response.
leaf certificate	Last certificate in a certificate chain. A leaf certificate is synonymous with an end entity certificate as RFC 5280 describes.
measurement	Representation of hardware/firmware/software or configuration data on an endpoint.

Term	Definition
message	See SPDM message .
message body	Portion of an SPDM message that carries additional data.
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. If an endpoint is communicating with multiple peer endpoints concurrently, the message transcripts for the peers are accumulated separately and independently.
monotonically increasing	This specification uses the term <i>monotonically increasing</i> to describe an integer field where the value of each instance of the field in a series increases from a lower starting point to a higher ending point without repeating values. For instance, a <i>monotonically increasing</i> field may contain the values 1, 3, 4, 7, and 9.
most significant byte (MSB)	Highest-order byte in a number consisting of multiple bytes.
Negotiated State	<p>Set of parameters that represents the state of the communication between a corresponding pair of Requester and Responder at the successful completion of the <code>NEGOTIATE_ALGORITHMS</code> messages.</p> <p>These parameters may include values provided in <code>VERSION</code>, <code>CAPABILITIES</code>, and <code>ALGORITHMS</code> messages.</p> <p>Additionally, they may include parameters associated with the transport layer.</p> <p>They may include other values deemed necessary by the Requester or Responder to continue or preserve communication with each other.</p>
nibble	Computer term for a four-bit aggregation, or half of a byte.
non-invasive debug mode	A device mode that enables debug access that does not expose or allow modification of security-critical firmware, hardware, or settings.
nonce	Number that is unpredictable to entities other than its generator. The probability of the same number occurring more than once is negligible. A nonce may be generated by combining a random number of at least 64 bits, optionally concatenated with a monotonically increasing counter of size suitable for the application.
opaque data	Opaque data fields transfer data that is outside the scope of this specification. The semantics and usage of this data are implementation specific and are also outside the scope of this specification.
payload	Information-bearing fields of a message. These fields are separate from the transport fields and elements, such as address fields, framing bits, and checksums, that transport the message from one point to another.
physical transport binding	Specifications that define how a base messaging protocol is implemented on a particular physical transport type and medium, such as SMBus/I ² C or PCI Express™ Vendor Defined Messaging.
Platform Management Component Intercommunication (PMCI)	Working group under the DMTF 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.
record	A unit or chunk of data that is either encrypted and/or authenticated.

Term	Definition
Requester	Original transmitter, or source, of an SPDM request message. It is also the ultimate receiver, or destination, of an SPDM response message. A Requester is the sender of the <code>GET_VERSION</code> request and remains the requester for the remainder of that connection.
Reset	This term is used to denote a Reset or restart of a device that runs the Requester or Responder code, which typically leads to the loss of all volatile state on the device.
Responder	Ultimate receiver, or destination, of an SPDM request message. It is also the original transmitter, or source of an SPDM response message.
root certificate	First certificate in a certificate chain, which acts as the trust anchor and is typically self-signed.
secure session	Provides either encryption or message authentication or both for communicating data over a transport.
Security Protocols and Data Models (SPDM) WG	Working group under the DMTF that defines standards to enable security for platforms, whether for the control plane, data plane, or other infrastructure.
sequentially decreasing	This specification uses the term <i>sequentially decreasing</i> to describe an integer field where the value of each instance of the field in a series decrements from a higher starting point to a lower ending point without skipping or repeating values. For instance, a <i>sequentially decreasing</i> field may contain the values 255, 254, 253, 252, and 251.
sequentially increasing	This specification uses the term <i>sequentially increasing</i> to describe an integer field where the value of each instance of the field in a series increments from a lower starting point to a higher ending point without skipping or repeating values. For instance, a <i>sequentially increasing</i> field may contain the values 1, 2, 3, 4, and 5.
session keys	Any secrets, derived cryptographic keys, or any cryptographic information bound to a session.
Session-Secrets-Exchange	Any SPDM request and their corresponding response that initiates a session and provides initial cryptographic exchange. Examples of such requests are <code>KEY_EXCHANGE</code> and <code>PSK_EXCHANGE</code> .
Session-Secrets-Finish	This term denotes any SPDM request and its corresponding response that finalizes a session setup and provides the final exchange of cryptographic or other information before application data can be securely transmitted. Examples of such requests are <code>FINISH</code> and <code>PSK_FINISH</code> .
SPDM message payload	Portion of the message body of an SPDM message. This portion of the message is separate from those fields and elements that identify the SPDM version, the SPDM request and response codes, and the two parameters.
SPDM message	Unit of communication in SPDM communications. See Generic SPDM message format .
SPDM request message	Message that is sent to an endpoint to request a specific SPDM operation. A corresponding SPDM response message acknowledges receipt of an SPDM request message.
SPDM response message	Message that is sent in response to a specific SPDM request message. This message includes a <code>Response Code</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 shall not be able to misbehave in a way that would leak any more privileges than are granted to them in accordance with the security policy. Reference: https://en.wikipedia.org/wiki/Trusted_computing_base

Term	Definition
trusted environment	An environment where the operator is assured of no unauthorized interference in operations.

6 Symbols and abbreviated terms

The abbreviations that [DSP0004](#), [DSP0223](#), and [DSP1001](#) define apply to this document.

The following additional abbreviations are used in this document.

Abbreviation	Definition
AEAD	Authenticated Encryption with Associated Data
CA	<i>certificate authority</i>
DMTF	Formerly the <i>Distributed Management Task Force</i>
ECC	Elliptic-curve cryptography
ECDSA	Elliptic-curve Digital Signature Algorithm
KDF	Key Derivation Function
MAC	Message Authentication Code
MSB	<i>most significant byte</i>
OID	Object identifier
PMCI	<i>Platform Management Component Intercommunication</i>
RMA	Return Merchandise Authorization
RSA	Rivest–Shamir–Adleman
SPDM	Security Protocol and Data Model
TCB	<i>trusted computing base</i>
VCA	Version-Capabilities-Algorithms

93 7 SPDM message exchanges

94 The message exchanges that this specification defines are between two endpoints and are performed and exchanged through sending and receiving of *SPDM* messages that *SPDM messages* defines. 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.

95 The specification-defined message exchanges enable Requesters to:

- Discover and negotiate the security capabilities of a Responder.
- Authenticate or provision an identity of a Responder.
- Retrieve the measurements of a Responder.
- Securely establish cryptographic *session keys* to construct a secure communication channel for the transmission or reception of *application data*.
- Receive notifications of selectable events when certain scenarios transpire.

96 These message exchange capabilities are built on top of well-known and established security practices across the computing industry. The following clauses provide a brief overview of each message exchange capability. Some message exchange capabilities are based on the security model that the *USB Authentication Specification Rev 1.0 with ECN and Errata through January 7, 2019* defines.

97 7.1 Security capability discovery and negotiation

98 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 this specification defines. 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.

99 7.2 Identity authentication

100 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.

101 At a high level, the authentication of the identity of a Responder involves these processes:

- [Identity provisioning](#)
 - [Runtime authentication](#)
-

102 7.2.1 Identity provisioning

103 Identity provisioning is the process that device vendors follow during or after hardware manufacturing to equip a device with a secure identifier. In the context of this specification, this secure identifier consists of an asymmetric key pair and, [optionally](#), a certificate to bind the key pair to a particular instance of a device and associate it with additional metadata. The specifics of key generation and provisioning are outside the scope of this specification. However, as the security of the SPDM protocol depends on device identities that cannot be easily modified, removed, or copied, it is strongly recommended that identity keys are unique per device and generated using cryptographically strong random seeds.

104 7.2.1.1 Certificate models

105 If trust in a device public key is established through a certificate, the certificate is typically part of a [certificate chain](#). The certificate chain has a [root certificate](#) (`RootCert`) as its root and a [leaf certificate](#) as the last certificate in it. The `RootCert` is generated by a trusted root [certificate authority \(CA\)](#) and certifies the certificate containing the device public key either directly or indirectly through a number of intermediate CAs. [Authentication initiators](#) use the `RootCert` to verify the validity of device certificate chains.

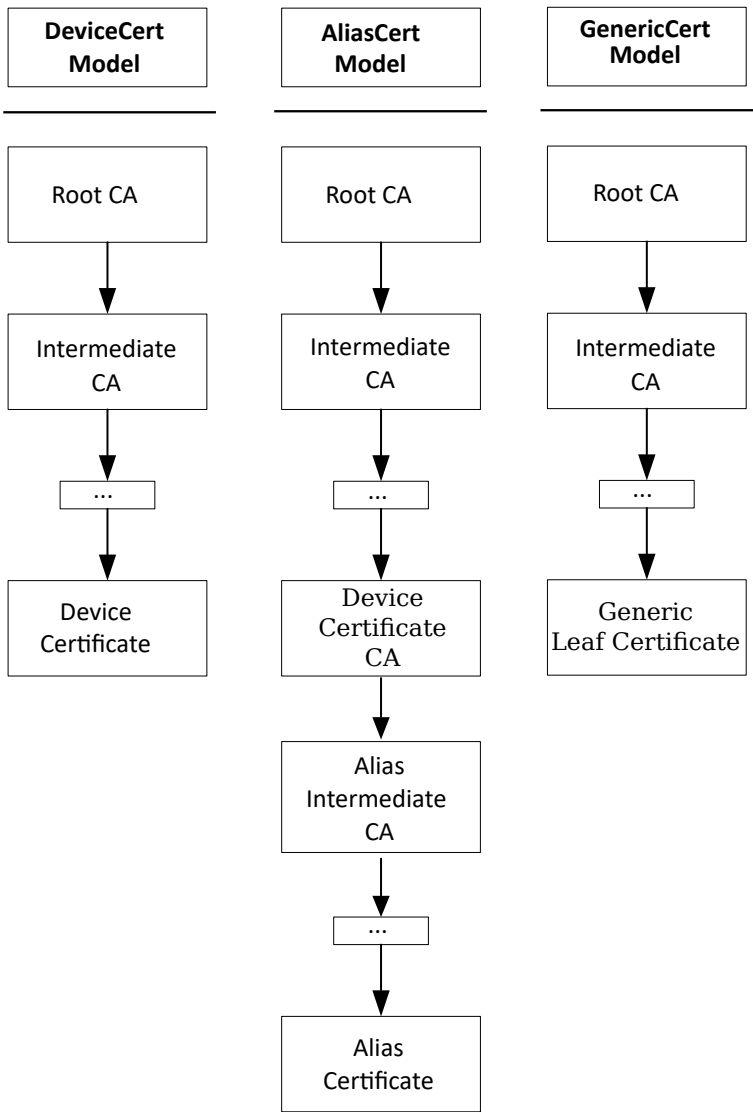
106 The certificate chain should contain at least one certificate that includes hardware identity information, regardless of the certificate model that is in use. The [Hardware identity OID](#) should be used to indicate which certificate conveys the hardware identity. Though existing deployments might not include the [Hardware identity OID](#) in a certificate, it is strongly recommended that new deployments include this information. The public/private key pair associated with a hardware identity certificate is constant on the instance of the device, regardless of the version of firmware running on the device.

107 SPDM defines multiple overarching formats for certificate chains, referred to as certificate chain models. While the details of each certificate chain model vary, all of them follow the general format of connecting from a [root certificate](#) (`RootCert`) to a [leaf certificate](#), possibly through one or more [intermediate certificates](#).

108 A Responder can use one or more of the certificate chain models. A Requester should be capable of performing [Runtime authentication](#) on a certificate chain that conforms to any of the models.

109 [Figure 1 — SPDM certificate chain models](#) shows the SPDM certificate chain models:

110 **Figure 1 — SPDM certificate chain models**



111 7.2.1.1.1 Device certificate model

112 When the device certificate (DeviceCert) model is in use, the leaf certificate is a Device Certificate, which contains the public key that corresponds to the device private key. Through the certificate chain, the root CA indirectly endorses the device public key in the Device Certificate. In this model, the Device Certificate should contain the [Hardware identity OID](#).

113 7.2.1.1.2 Alias certificate model

- 114 When the alias certificate (`AliasCert`) model is in use, the leaf certificate is an Alias Certificate, in which case there may be one or more intermediate `AliasCert` certificates between the Device Certificate and the leaf Alias Certificate. In the `AliasCert` model, the device private key signs the next level Alias Certificate, and then the private key associated with the public key in each Alias Intermediate CA signs the Alias Certificate below it. When the `AliasCert` model is in use, the Device Certificate is referred to as a Device Certificate CA, indicating that the certificate both contains device hardware identity information and functions as a certificate authority to sign an additional certificate. In this model, the Device Certificate CA should contain the [Hardware identity OID](#).
- 115 A device that implements the `AliasCert` model might factor some mutable information, such as the measurement of a firmware image, into the derivation of the public/private key pairs for the intermediate and leaf alias certificates. Therefore, the asymmetric public/private key pairs for each Alias Certificate should be treated as mutable.
- 116 Through the certificate chain, the root CA indirectly endorses the device public key in the Device Certificate. When the `AliasCert` model is in use, the Alias Certificates are endorsed by the device private key, meaning that the Alias Certificates are also indirectly endorsed by the root CA.
- 117 When the `AliasCert` model is used, the device creates and endorses one or more certificates. The certificates from the root certificate to the Device Certificate are considered immutable because the Responder cannot change them, as they can only be changed through the `SET_CERTIFICATE` command or an equivalent capability. The certificates below the Device Certificate can be created on the device and are mutable certificates in that they can change when the device state changes, such as a device *reset*. The [Mutable certificate OID](#) should be used to indicate mutable certificates.
- 118 In addition, when the `AliasCert` model is used, one or more Alias Certificates can contain firmware identity information. Other standards bodies might define the format of the firmware identity information. Such definitions are outside the scope of this specification.
- 119 Note that a signature algorithm used with a mutable alias certificate can insert random data during signing, which would cause the digest of the certificate chain to change each time it is regenerated. An implementer can use a mechanism that is outside the scope of this specification to ensure that such a signature does not change between instances of `DIGESTS` and `CERTIFICATE` responses.

120 7.2.1.1.3 Generic certificate model

- 121 With the support of [multiple asymmetric keys](#), the need for another certificate model arises to accommodate varying use cases that `DeviceCert` and `AliasCert` models cannot fulfill. Thus, the generic certificate model offers the greatest flexibility to the device manufacturer, a manufacturer in the supply chain, and the users of the SPDM endpoint.
- 122 As [Figure 1 — SPDM certificate chain models](#) illustrates, much like the other certificate models, the generic certificate model, too, is composed of a chain of certificates starting with the root and ending with the leaf. The root CA, too,

either directly certifies the leaf certificate or indirectly certifies the leaf certificate (`GenericCert`) through one or more intermediate certificate authorities. In other words, this model is the most flexible (or least restrictive) of the certificate models in this specification. The main difference between this model and the other models is that SPDM shall not impose any requirements on the contents of each certificate in the chain in a generic certificate model other than the key pair and related information associated in the leaf certificate.

123 For example, in a device certificate model, the leaf certificate can contain elements that specifically identify the device and device manufacturer, whereas the generic certificate model has no such requirement nor any concept of a device certificate.

124 As such, the generic certificate model applies to certificates in slots greater than slot 0. A model in a certificate slot in this specification is either a `DeviceCert` , `AliasCert` , or `GenericCert` model.

125 The contents and use cases for the certificates of a generic certificate model, other than the associated key pair and related information in the leaf certificate, are outside the scope of this specification. Typically, the users of the SPDM endpoint, the device manufacturer, or standards define the contents and use cases of a generic certificate model.

126 7.2.2 Raw public keys

127 Instead of using certificate chains, the vendor can provision the raw public key of the Responder to the Requester in a trusted environment; for example, during the secure manufacturing process. In this case, trust of the public key of the Responder is established without the need for a certificate-based public key infrastructure.

128 The format of the provisioned public key is outside the scope of this specification. Vendors can use proprietary formats or public key formats that other standards define, such as [RFC 7250](#) and [RFC 4716](#).

129 7.2.3 Runtime authentication

130 Runtime authentication is the process by which an authentication initiator, or Requester, interacts with a Responder in a running system. The authentication initiator can retrieve the certificate chains from the Responder and send a unique challenge to the Responder. The Responder uses the private key associated with the leaf certificate to sign the challenge. The authentication initiator verifies the signature by using the public key associated with the leaf certificate of the Responder and any intermediate public keys within the certificate chain by using the root certificate as the trusted anchor.

131 If the public key of the Responder was provisioned to the Requester in a trusted environment, the authentication initiator sends a unique challenge to the Responder. The Responder signs the challenge with the private key. The authentication initiator verifies the signature by using the public key of the Responder. Device identification can be handled using the [GET_ENDPOINT_INFO request and ENDPOINT_INFO response messages](#) or the transport layer (which is outside the scope of this specification).

132 7.3 Firmware and configuration measurement

133 A measurement is a representation of firmware/software or configuration data on an endpoint. A measurement is typically either a cryptographic hash value of the data or the raw data itself. The endpoint optionally binds a measurement with the endpoint identity through the use of digital signatures. This binding enables an authentication initiator to establish the identity and measurement of the firmware/software or configuration running on the endpoint.

134 7.4 Secure sessions

135 Many devices exchange data that might require protection with other devices. In this specification, this data that is being exchanged is generically referred to as application data. The protocol of the application data usually exists at a higher layer, and as such it is outside the scope of this specification. The protocol of the application data usually allows for encrypted and/or authenticated data transfer.

136 This specification provides a method to perform a cryptographic key exchange such that the protocol of the application data can use the exchanged keys to provide a secure channel of communication by using encryption and message authentication. This cryptographic key exchange provides either Responder-only authentication or mutual authentication, both of which can be considered equivalent to [Runtime authentication](#). For more details, see the [Session](#) clause.

137 Finally, many SPDM requests and their corresponding responses can also be afforded the same protection. For more details, see [Table 6 — SPDM request and response messages validity](#) and the [SPDM request and response code issuance allowance](#) clause.

138 [Figure 2 — SPDM messaging protocol flow](#) gives a very high-level view of when the [secure session](#) starts.

139 7.5 Mutual authentication overview

140 The ability of a Responder to verify the authenticity of the Requester is called mutual authentication. Several mechanisms in this specification are detailed to provide mutual authentication capabilities. The cryptographic means to verify the identity of the Requester is the same as verifying the identity of the Responder. The [Identity provisioning](#) clause discusses identity in regards to the Responder but the details also apply to the Requester.

141 In general, when this specification states requirements or recommendations for Responders in the context of identity, those same rules also apply to Requesters in the context of mutual authentication. The various clauses in this specification provide more details.

142 7.6 Multiple asymmetric key support

143 An SPDM endpoint can use more than one asymmetric key pair for a negotiated asymmetric algorithm. This enables cryptographic isolation between different use cases which potentially increases the security posture of the SPDM endpoint and its corresponding SPDM connections. For example, an SPDM Responder can choose which key-pairs to use in a `CHALLENGE` request and which key pairs to use in a `GET_MEASUREMENTS` request. The SPDM Responder permits the `CHALLENGE` and `GET_MEASUREMENTS` requests to use the same key-pair for signing operations.

144 Additionally, a Responder can allow the Requester to select the use cases to associate with each asymmetric key pair. The Responder can, also, allow the Requester to request the generation of a new key pair.

145 To facilitate the use of multiple asymmetric keys, the ability to uniquely identify each key pair is essential. To achieve this, a unique key pair number, called `KeyPairID`, identifies each asymmetric key pair. Additionally, one or more leaf certificates can bind to the same asymmetric key pair.

146 7.7 Custom environments

147 A fixed or predetermined environment is an environment where certain characteristics of the environment are fixed or known before the SPDM endpoints communicate with each other. In many cases, these characteristics are determined even before the environment can operate such as during the design phase. An example of a such an environment is when two specific endpoints can only communicate with each other. These environments may forfeit certain SPDM features such as interoperability. However, the security posture and guarantees of these environments are outside the scope of this specification.

148 7.8 Notification overview

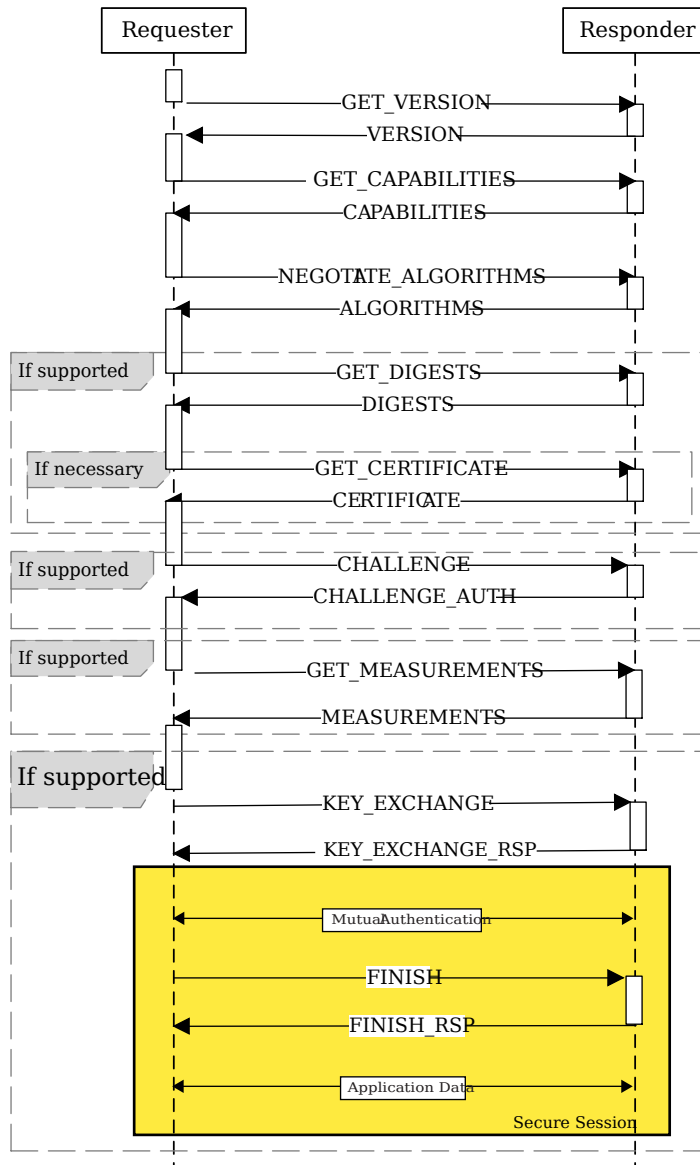
149 To aid an SPDM endpoint in enforcing its security policy requirements in an efficient, reliable, and timely manner, the [SPDM event mechanism](#) provides a method to asynchronously deliver a notification to or receive a notification from the interested SPDM endpoint. This mechanism allows an interested SPDM endpoint to choose only the event types it wants to receive. For more details, see [Event mechanism](#).

150 8 SPDM messaging protocol

151 The SPDM messaging protocol defines a request-response messaging model between two endpoints to perform the message exchanges outlined in [SPDM message exchanges](#). Each *SPDM request message* shall be responded to with an SPDM response message as this specification defines unless this specification states otherwise.

152 [Figure 2 — SPDM messaging protocol flow](#) depicts the high-level request-response flow diagram for SPDM. An endpoint that acts as the *Requester* sends an SPDM request message to another endpoint that acts as the *Responder*, and the Responder returns an SPDM response message to the Requester.

153 **Figure 2 — SPDM messaging protocol flow**



154 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 for each of the request and response messages.

155 The Requester shall issue `GET_VERSION` , `GET_CAPABILITIES` , and `NEGOTIATE_ALGORITHMS` request messages before issuing any other request messages. The responses to `GET_VERSION` , `GET_CAPABILITIES` , and `NEGOTIATE_ALGORITHMS` can be saved by the Requester so that after Reset the Requester can skip these requests.

8.1 SPDM connection model

In SPDM, communication between a pair of SPDM endpoints starts when one endpoint sends a `GET_VERSION` request to another SPDM endpoint. The SPDM endpoint that starts the communication is called the Requester. The endpoint receiving the `GET_VERSION` and providing the `VERSION` response is called a Responder. The communication between a pair of Requester and Responder is called a connection. One or more connections can exist between a Requester and Responder. Different connections might exist over the same transport or over different transports. When there are multiple connections over the same transport, the transport is responsible for providing mechanisms for SPDM endpoints to distinguish between one or more connections. When the transport does not provide such a mechanism, there shall be one connection between the Requester and Responder.

SPDM endpoints can be both a Requester and Responder. As a Requester, an SPDM endpoint can communicate with one or more Responders. Likewise, as a Responder, an SPDM endpoint can respond to multiple Requesters. The SPDM connection model considers each of these communications to be a separate connection. For example, a pair of SPDM endpoints can be both Requester and Responder to each other. Thus, the SPDM connection model considers this to be two separate connections.

Within a connection, the Requester remains the Requester for the remainder of the connection. Likewise, the Responder remains the Responder for the remainder of the connection. However, under certain scenarios allowed by SPDM, a Responder can send a request to a Requester and, likewise, a Requester might provide a response to a Responder. These cases are limited and this specification explicitly defines these cases. In such scenarios, when a Requester provides a response, the Requester shall abide by all requirements in this specification as if they are a Responder for that request. Similarly, when a Responder sends a request, the Responder shall abide by all requirements in this specification as if they are a Requester for that request.

Within a connection, the Requester can establish one or more secure sessions. These secure sessions are considered to be part of the same connection. Secure sessions can terminate and additional sessions can be established at any time. A `GET_VERSION` can reset the connection and all context associated with that connection including, but not limited to, information such as session keys and session IDs. However, this is not considered a termination of the connection. A connection can terminate due to external events such as a device reset or an error-handling strategy implemented on an SPDM endpoint, but such scenarios are outside the scope of this specification. Connections can be terminated using mechanisms outside the scope of this specification.

8.2 SPDM bits-to-bytes mapping

All SPDM fields, regardless of size or endianness, map the highest numeric bits to the highest numerically assigned byte in sequentially decreasing order down to and including the least numerically assigned byte of that field. The following two figures illustrate this mapping.

[Figure 3 — One-byte field bit map](#) shows the one-byte field bit map:

Figure 3 — One-byte field bit map

Example: A One-Byte Field

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

Figure 4 — Two-byte field bit map shows the two-byte field bit map:

Figure 4 — Two-byte field bit map

Example: A Two-Byte Field

Byte 3								Byte 2							
Bit	Bit	Bit	Bit	Bit	Bit	Bit	Bit	Bit	Bit	Bit	Bit	Bit	Bit	Bit	Bit
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

8.3 Generic SPDM message format

Table 3 — Generic SPDM message field definitions defines the fields that constitute a generic SPDM message, including the message header and payload:

Table 3 — Generic SPDM message field definitions

Byte offset	Bit offset	Size (bits)	Field	Description
0	[7:4]	4	SPDM Major Version	Shall be 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	Shall be 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	Shall be the request message code or response code, which Table 4 — SPDM request codes and Table 5 — SPDM response codes enumerate. 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	Shall be the first one-byte parameter. The contents of the parameter are specific to the Request Response Code.

Byte offset	Bit offset	Size (bits)	Field	Description
3	[7:0]	8	Param2	Shall be the second one-byte parameter. The contents of the parameter are specific to the Request Response Code .
4	See the description.	Variable	SPDM message payload	Shall be zero or more bytes that are specific to the Request Response Code .

8.3.1 SPDM version

The `SPDMVersion` field, present as the first field in all SPDM messages, indicates the version of the SPDM specification that the format of an SPDM message adheres to. The format of this field shall be the same as byte 0 in [Table 3 — Generic SPDM message field definitions](#). The value of this field shall be the same value as the version of this specification except for `GET_VERSION` and `VERSION` messages.

For example, if the version of this specification is 1.2, the value of `SPDMVersion` is `0x12`, which also corresponds to an `SPDM Major Version` of 1 and an `SPDM Minor Version` of 2. See [Version encoding](#) for more examples.

The version of this specification can be found on the title page and in the footer of the other pages in this document.

The `SPDMVersion` for the version of this specification shall be `0x13`.

The `SPDMVersionString` shall be a string formed by concatenating the major version, a period ("."), and the minor version. For example, if the version of this specification is 1.2.3, then `SPDMVersionString` is "1.2".

8.4 SPDM request codes

[Table 4 — SPDM request codes](#) defines the SPDM request codes. The **Implementation requirement** column indicates requirements on the Requester.

All SPDM-compatible implementations shall use [SPDM request codes](#).

If an `ERROR` response is sent for unsupported request codes, the `ErrorCode` shall be `UnsupportedRequest`.

Table 4 — SPDM request codes

Request	Code value	Implementation requirement	Message format
GET_DIGESTS	0x81	Optional	Table 34 — GET_DIGESTS request message format
GET_CERTIFICATE	0x82	Optional	Table 38 — GET_CERTIFICATE request message format

Request	Code value	Implementation requirement	Message format
CHALLENGE	0x83	Optional	Table 44 — CHALLENGE request message format
GET_VERSION	0x84	Required	Table 8 — GET_VERSION request message format
CHUNK_SEND	0x85	Optional	Table 96 — CHUNK_SEND request format
CHUNK_GET	0x86	Optional	Table 100 — CHUNK_GET request format
GET_ENDPOINT_INFO	0x87	Optional	Table 119 — GET_ENDPOINT_INFO request format
GET_MEASUREMENTS	0xE0	Optional	Table 49 — GET_MEASUREMENTS request message format
GET_CAPABILITIES	0xE1	Required	Table 11 — GET_CAPABILITIES request message format
GET_SUPPORTED_EVENT_TYPES	0xE2	Optional	Table 109 — GET_SUPPORTED_EVENT_TYPES request message format
NEGOTIATE_ALGORITHMS	0xE3	Required	Table 15 — NEGOTIATE_ALGORITHMS request message format
KEY_EXCHANGE	0xE4	Optional	Table 69 — KEY_EXCHANGE request message format
FINISH	0xE5	Optional	Table 72 — FINISH request message format
PSK_EXCHANGE	0xE6	Optional	Table 74 — PSK_EXCHANGE request message format
PSK_FINISH	0xE7	Optional	Table 76 — PSK_FINISH request message format
HEARTBEAT	0xE8	Optional	Table 78 — HEARTBEAT request message format
KEY_UPDATE	0xE9	Optional	Table 80 — KEY_UPDATE request message format
GET_ENCAPSULATED_REQUEST	0xEA	Optional	Table 83 — GET_ENCAPSULATED_REQUEST request message format
DELIVER_ENCAPSULATED_RESPONSE	0xEB	Optional	Table 85 — DELIVER_ENCAPSULATED_RESPONSE request message format
END_SESSION	0xEC	Optional	Table 87 — END_SESSION request message format
GET_CSR	0xED	Optional	Table 90 — GET_CSR request message format
SET_CERTIFICATE	0xEE	Optional	Table 93 — SET_CERTIFICATE request message format
GET_MEASUREMENT_EXTENSION_LOG	0xEF	Optional	Table 126 — GET_MEASUREMENT_EXTENSION_LOG message format
SUBSCRIBE_EVENT_TYPES	0xF0	Optional	Table 113 — SUBSCRIBE_EVENT_TYPES request message format
SEND_EVENT	0xF1	Optional	Table 116 — SEND_EVENT request message format
GET_KEY_PAIR_INFO	0xFC	Optional	Table 102 — GET_KEY_PAIR_INFO request message format

Request	Code value	Implementation requirement	Message format
SET_KEY_PAIR_INFO	0xFD	Optional	Table 106 — SET_KEY_PAIR_INFO request message format
VENDOR_DEFINED_REQUEST	0xFE	Optional	Table 57 — VENDOR_DEFINED_REQUEST request message format
RESPOND_IF_READY	0xFF	Required	Table 56 — RESPOND_IF_READY request message format
Reserved	All other values		SPDM implementations compatible with this version shall not use the reserved request codes.

8.5 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 [Table 5 — SPDM response codes](#).

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

[Table 5 — SPDM response codes](#) defines the response codes for SPDM. The **Implementation requirement** column indicates requirements on the Responder.

Table 5 — SPDM response codes

Response	Value	Implementation requirement	Message format
DIGESTS	0x01	Optional	Table 35 — Successful DIGESTS response message format
CERTIFICATE	0x02	Optional	Table 40 — Successful CERTIFICATE response message format
CHALLENGE_AUTH	0x03	Optional	Table 45 — Successful CHALLENGE_AUTH response message format
VERSION	0x04	Required	Table 9 — Successful VERSION response message format
CHUNK_SEND_ACK	0x05	Optional	Table 98 — CHUNK_SEND_ACK response message format
CHUNK_RESPONSE	0x06	Optional	Table 101 — CHUNK_RESPONSE response format
ENDPOINT_INFO	0x07	Optional	Table 122 — ENDPOINT_INFO response format
MEASUREMENTS	0x60	Optional	Table 52 — Successful MEASUREMENTS response message format
CAPABILITIES	0x61	Required	Table 12 — Successful CAPABILITIES response message format
SUPPORTED_EVENT_TYPES	0x62	Optional	Table 110 — SUPPORTED_EVENT_TYPES response message format
ALGORITHMS	0x63	Required	Table 21 — Successful ALGORITHMS response message format

Response	Value	Implementation requirement	Message format
KEY_EXCHANGE_RSP	0x64	Optional	Table 71 — Successful KEY_EXCHANGE_RSP response message format
FINISH_RSP	0x65	Optional	Table 73 — Successful FINISH_RSP response message format
PSK_EXCHANGE_RSP	0x66	Optional	Table 75 — PSK_EXCHANGE_RSP response message format
PSK_FINISH_RSP	0x67	Optional	Table 77 — Successful PSK_FINISH_RSP response message format
HEARTBEAT_ACK	0x68	Optional	Table 79 — HEARTBEAT_ACK response message format
KEY_UPDATE_ACK	0x69	Optional	Table 81 — KEY_UPDATE_ACK response message format
ENCAPSULATED_REQUEST	0x6A	Optional	Table 84 — ENCAPSULATED_REQUEST response message format
ENCAPSULATED_RESPONSE_ACK	0x6B	Optional	Table 86 — ENCAPSULATED_RESPONSE_ACK response message format
END_SESSION_ACK	0x6C	Optional	Table 89 — END_SESSION_ACK response message format
CSR	0x6D	Optional	Table 92 — CSR response message format
SET_CERTIFICATE_RSP	0x6E	Optional	Table 95 — Successful SET_CERTIFICATE_RSP response message format
MEASUREMENT_EXTENSION_LOG	0x6F	Optional	Table 127 — Successful MEASUREMENT_EXTENSION_LOG message format
SUBSCRIBE_EVENT_TYPES_ACK	0x70	Optional	Table 114 — SUBSCRIBE_EVENT_TYPES_ACK response message format
EVENT_ACK	0x71	Optional	Table 118 — EVENT_ACK response message format
KEY_PAIR_INFO	0x7C	Optional	Table 103 — KEY_PAIR_INFO response message format
SET_KEY_PAIR_INFO_ACK	0x7D	Optional	Table 108 — SET_KEY_PAIR_INFO_ACK response message format
VENDOR_DEFINED_RESPONSE	0x7E	Optional	Table 67 — VENDOR_DEFINED_RESPONSE response message format
ERROR	0x7F	Required	Table 57 — ERROR response message format
Reserved	All other values		SPDM implementations compatible with this version shall not use the reserved response codes.

188 8.6 SPDM request and response code issuance allowance

189 [Table 6 — SPDM request and response messages validity](#) describes the conditions under which a request and response can be issued.

- 190 The **Session** column describes whether the respective request and response can be sent in a session. If the value is "Allowed", the issuer of the request and response shall be able to send it in a secure session, thereby affording them the protection of a secure session. If the **Session** column value is "Prohibited", the issuer shall be prohibited from sending the respective request and response in a secure session.
- 191 The **Outside of a session** column indicates which requests and responses are allowed to be sent free and independent of a session, thereby lacking the protection of a secure session. An "Allowed" in this column indicates that the respective request and response shall be able to be sent outside the context of a secure session. Likewise, a "Prohibited" in this column shall prohibit the issuer from sending the respective request or response outside the context of a session.
- 192 A request and its corresponding response can have an "Allowed" value in both the **Session** and **Outside of a session** columns, in which case they can be sent and received in both scenarios but might have additional restrictions. For details, see the respective request and response clauses.
- 193 A request and its corresponding response that has an "Allowed" value in the **Session** and "Prohibited" in the **Outside of a session** columns are commands used by the session. These commands only operate on the session that they were sent under, which is outside the scope of this specification. The session ID is implicit from the session used to transmit the commands.
- 194 Finally, the **Session phases** column describes which phases of a session the respective request and response shall be issued when they are allowed to be issued in a session.
- 195 If, during the session handshake phase, an unexpected request is received using a valid session ID, the Responder shall either send an `ERROR` message in the session with `ErrorCode=UnexpectedRequest` or silently discard the request.
- 196 `Vendor-defined` shall indicate whether a `VENDOR_DEFINED_REQUEST` and `VENDOR_DEFINED_RESPONSE` is "Allowed" or "Prohibited" for use in the **Session**, **Outside of a session**, and the applicable **Session phases**.
- 197 For details, see the [Session](#) clause.

198 **Table 6 — SPDM request and response messages validity**

Request	Response	Outside of a session	Session	Session phases
GET_MEASUREMENTS	MEASUREMENTS	Allowed	Allowed	Application Phase
FINISH	FINISH_RSP	Prohibited	Allowed	Session Handshake
PSK_FINISH	PSK_FINISH_RSP	Prohibited	Allowed	Session Handshake
HEARTBEAT	HEARTBEAT_ACK	Prohibited	Allowed	Application Phase
KEY_UPDATE	KEY_UPDATE_ACK	Prohibited	Allowed	Application Phase
END_SESSION	END_SESSION_ACK	Prohibited	Allowed	Application Phase

Request	Response	Outside of a session	Session	Session phases
Not Applicable	ERROR	Allowed	Allowed	All Phases
GET_ENCAPSULATED_REQUEST	ENCAPSULATED_REQUEST	Allowed	Allowed	All Phases
DELIVER_ENCAPSULATED_RESPONSE	ENCAPSULATED_RESPONSE_ACK	Allowed	Allowed	All Phases
VENDOR_DEFINED_REQUEST	VENDOR_DEFINED_RESPONSE	Vendor-defined	Vendor-defined	Vendor-defined
CHUNK_SEND	CHUNK_SEND_ACK	Allowed	Allowed	All Phases
CHUNK_GET	CHUNK_RESPONSE	Allowed	Allowed	All Phases
GET_ENDPOINT_INFO	ENDPOINT_INFO	Allowed	Allowed	Application Phase
GET_CSR	CSR	Allowed	Allowed	Application Phase
SET_CERTIFICATE	SET_CERTIFICATE_RSP	Allowed	Allowed	Application Phase
GET_DIGESTS	DIGESTS	Allowed	Allowed	Application Phase
GET_CERTIFICATE	CERTIFICATE	Allowed	Allowed	Application Phase
GET_KEY_PAIR_INFO	KEY_PAIR_INFO	Allowed	Allowed	Application Phase
SET_KEY_PAIR_INFO	SET_KEY_PAIR_INFO_ACK	Allowed	Allowed	Application Phase
GET_MEASUREMENT_EXTENSION_LOG	MEASUREMENT_EXTENSION_LOG	Allowed	Allowed	Application Phase
GET_SUPPORTED_EVENT_TYPES	SUPPORTED_EVENT_TYPES	Prohibited	Allowed	Application Phase
SUBSCRIBE_EVENT_TYPES	SUBSCRIBE_EVENT_TYPES_ACK	Prohibited	Allowed	Application Phase
SEND_EVENT	EVENT_ACK	Prohibited	Allowed	Application Phase
RESPOND_IF_READY	Response to Original Request (*)	Allowed (*)	Allowed (*)	All Phases (*)
All others	All others	Allowed	Prohibited	Not Applicable

199 (*) [See RESPOND_IF_READY request description for details](#)

200 8.7 Concurrent SPDM message processing

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

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

203 8.8 Requirements for Requesters

- 204 A Requester shall not have multiple outstanding requests to the same Responder within a connection, with the following exceptions:
- As the [GET_VERSION request and VERSION response messages](#) clause describes, a Requester can issue a `GET_VERSION` to a Responder to reset the connection at any time, even if the Requester has existing outstanding requests to the same Responder.
 - In the [large SPDM message transfer mechanism](#), a single large SPDM request message and a single `CHUNK_SEND` request can be outstanding at the same time.
- 205 An outstanding request is a request where the request message has begun transmission, the corresponding response has not been fully received, and the request is not a retry as described in [Timing Requirements](#).
- 206 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.
 - The Requester encounters an internal error or Reset.
 - The Requester sends a `GET_VERSION` to reinitialize the session.
- 207 A Requester might send simultaneous request messages to different Responders.

208 8.9 Requirements for Responders

- 209 A Responder is not required to process more than one request message at a time, even across connections, with the following exceptions:
- As the [GET_VERSION request and VERSION response messages](#) clause describes, a Requester can issue a `GET_VERSION` to a Responder to reset a connection at any time, even if the Requester has existing outstanding requests to the same Responder.
 - In the [large SPDM message transfer mechanism](#), a single large SPDM request message and a single `CHUNK_SEND` request can be outstanding at the same time.
 - Retries can be issued multiple times to the same Responder, as [Timing requirements](#) defines.
- 210 A Responder that is not ready to accept a new request message or process more than one outstanding request at a time from the same Requester shall either respond with an `ERROR` message of `ErrorCode=Busy` or silently discard the request message.
-

211 If a Responder is working on a request message from a Requester, the Responder can respond with an `ERROR` message of `ErrorCode=Busy`.

212 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.

213 8.10 Transcript and transcript hash calculation rules

214 The transcript is a concatenation of the prescribed full messages or message fields in order. Consequently, the transcript hash is the hash of the transcript using the negotiated hash algorithm (`BaseHashSel` or `ExtHashSel` of `ALGORITHMS`). For messages that are encrypted, the plaintext messages are used in the transcript. Where a transcript indicates that the hash of the specified certificate chain is used, the hash of the certificate chain is calculated over the specified certificate chain, as [Table 33 — Certificate chain format](#) describes. Messages that contribute to a transcript may be optional and/or conditional and will only contribute to a transcript if issued. Such messages are identified by the text "if issued" in the transcript definition. For a given message, if it does not have the "if issued" text in the transcript definition, then it is required to be present in the transcript. When an endpoint calculates the transcript hash over a series of messages, the endpoint shall ensure both the existence and the order of the messages as specified by each transcript hash calculation rule.

9 Timing requirements

[Table 7 — Timing specification for SPDM messages](#) shows the timing specifications for Requesters and Responders.

If the Requester does not receive a response within **T1** or **T2** time accordingly, the Requester can retry a request message. A retry of a request message shall be a complete retransmission of the original SPDM request message. From the perspective of a Requester, a retry of a request message is the retransmission of the original SPDM request one or more times in succession directly following the transmission of the original SPDM request. From the perspective of a Responder, a retry of a request message is the reception of the same SPDM request one or more times in succession, assuming that the transport receives messages in order. Successive SPDM requests are different if the values of any bits differ between them, in which case the Responder will process them differently.

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

9.1 Timing measurements

Unless otherwise stated, a Requester shall measure timing parameters applicable to it from the end of a successful transmission of an SPDM request to the beginning of the reception of the corresponding SPDM response. A Responder shall measure timing parameters applicable to it from the end of the reception of the SPDM request to the beginning of transmission of the response. The requirement assumes that the Responder has immediate access to the transport.

9.2 Timing parameters

In [Table 7 — Timing specification for SPDM messages](#), timing parameters are differentiated into two categories: the timing parameters for non-cryptographic operations (**T1**) and the timing parameters for cryptographic operations (**T2**). The timing parameters are differentiated in this manner to allow a Responder to request additional time for cryptographic operations. The timing parameters apply to normal conditions, and some operations may take additional time in some situations. For instance, a Responder may need additional time to process a non-cryptographic operation because of another operation in progress or some other condition. In this case, the Responder shall respond with an **ERROR** message of **ErrorCode=ResponseNotReady** to indicate that it needs more time.

The Responder can request time beyond **ST1** for any non-cryptographic operation other than **GET_VERSION**. Since **GET_VERSION** serves as a reset to the connection, a Requester might send **GET_VERSION** requests as quickly as allowed by **T1** until it receives a response. The Responder shall not respond to **GET_VERSION** with an **ERROR** message of **ErrorCode=ResponseNotReady**.

9.3 Timing specification table

The **Ownership** column of [Table 7 — Timing specification for SPDM messages](#) specifies whether the timing parameter applies to the Responder or Requester. For *encapsulated requests*, the Requester shall comply with the timing parameters where the **Ownership** indicates a Responder.

Table 7 — Timing specification for SPDM messages

Timing parameter	Ownership	Value	Units	Description
RTT	Requester	See the description.	μs	<p>This value shall be the worst-case round-trip transport timing.</p> <p>The maximum value shall be the worst-case total time for the complete transmission and delivery of an SPDM message round trip at the transport layer(s). The actual value for this parameter is transport- or media-specific. Both the actual value and how an endpoint obtains this value are outside the scope of this specification. A Requester shall measure this timing parameter from the end of a successful transmission of an SPDM request to the beginning of the reception of the corresponding SPDM response less <code>ST1</code> or <code>CT</code>, depending on the Request.</p>
ST1	Responder	100,000	μs	<p>This value shall be the maximum amount of time the Responder has to provide a response under normal conditions to requests that do not require cryptographic processing, such as the <code>GET_CAPABILITIES</code>, <code>GET_VERSION</code>, or <code>NEGOTIATE_ALGORITHMS</code> request messages.</p> <p>See Table 11 — GET_CAPABILITIES request message format, Table 8 — GET_VERSION request message format, and Table 15 — NEGOTIATE_ALGORITHMS request message format.</p>
T1	Requester	<code>RTT</code> + <code>ST1</code>	μs	<p>This value shall be the minimum amount of time the Requester shall wait before issuing a retry for requests that do not require cryptographic processing.</p> <p>For details, see the <code>ST1</code> timing parameter.</p>

Timing parameter	Ownership	Value	Units	Description
CT	Requester and Responder	$2^{CTExponent}$	μs	<p>CTExponent is reported in the <code>GET_CAPABILITIES</code> request message and <code>CAPABILITIES</code> response message.</p> <p>This parameter is applicable to both a Responder and Requester as the Ownership column shows. Specifically for a Requester, this field is applicable when the Requester provides a response that requires cryptographic processing such as in the mutual authentication portion of a <code>KEY_EXCHANGE</code> flow. When the Requester provides a response that requires cryptographic processing, the Requester shall measure timing just as a Responder would.</p> <p>This timing parameter shall be the maximum amount of time the endpoint has to provide any response requiring cryptographic processing under normal conditions, such as the <code>GET_MEASUREMENTS</code> or <code>CHALLENGE</code> request messages. If the Responder cannot respond within <code>CT</code>, the Responder shall respond with an <code>ERROR</code> message of <code>ErrorCode=ResponseNotReady</code> to indicate that it needs more time.</p> <p>See Table 11 — GET_CAPABILITIES request message format, Table 12 — Successful CAPABILITIES response message format, Table 49 — GET_MEASUREMENTS request message format, and Table 44 — CHALLENGE request message format.</p>
T2	Requester	<code>RTT</code> + <code>CT</code>	μs	<p>This value shall be the minimum amount of time the Requester shall wait before issuing a retry for requests that require cryptographic processing.</p> <p>For details, see the <code>CT</code> timing parameter.</p>
RDT	Responder	$2^{RDTEExponent}$	μs	<p>This value shall be the recommended delay in microseconds that the Responder needs to complete the requested cryptographic operation. When the Responder cannot complete cryptographic processing response within the <code>CT</code> time, it shall provide <code>RDTEExponent</code> as part of the <code>ERROR</code> response as Table 57 — ERROR response message format shows. For details, see <code>ErrorCode=ResponseNotReady</code> in Table 59 — ResponseNotReady extended error data for the <code>RDTEExponent</code> value.</p>
WT	Requester	<code>RDT</code>	μs	<p>This value shall be the amount of time that the Requester should wait before issuing the <code>RESPOND_IF_READY</code> request message as Table 65 — RESPOND_IF_READY request message format shows.</p> <p>The Requester shall measure this time parameter from the reception of the <code>ERROR</code> response to the transmission of the <code>RESPOND_IF_READY</code> request. The Requester can include the transmission time of the <code>ERROR</code> from the Responder to Requester as time spent waiting for <code>WT</code> to expire. For example, if a Responder indicates <code>WT</code> is two seconds and the <code>ERROR</code> response takes one second to transport to the Requester, the Requester only needs to wait an additional one second upon reception of the <code>ERROR</code> response.</p> <p>For details, see the <code>RDT</code> timing parameter.</p>

Timing parameter	Ownership	Value	Units	Description
WT _{Max}	Requester	$(RDT * RDTM) - RTT$	μs	<p>This value shall be the maximum wait time the Requester has to issue the RESPOND_IF_READY request message, as Table 65 — RESPOND_IF_READY request message format shows, unless the Requester issued a successful RESPOND_IF_READY request message, as Table 65 — RESPOND_IF_READY request message format shows, earlier. The Requester shall start measuring time from the reception of the first ERROR message of ErrorCode=ResponseNotReady with the same Token until WT_{Max} μs elapses or the corresponding Response is successfully received.</p> <p>After this time has passed, the Responder is allowed to drop the response. The Requester shall take into account the transmission time of the ERROR response, as Table 57 — ERROR response message format shows, from the Responder to Requester when calculating WT_{Max}.</p> <p>The RDTM value appears in Table 59 — ResponseNotReady extended error data.</p> <p>The Responder should ensure that WT_{Max} does not result in less than WT in determination of RDTM.</p> <p>See ErrorCode=ResponseNotReady in Table 59 — ResponseNotReady extended error data.</p>
HeartbeatPeriod	Requester and Responder	Variable	s	See the HEARTBEAT request and HEARTBEAT_ACK response clause.

10 SPDM messages

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

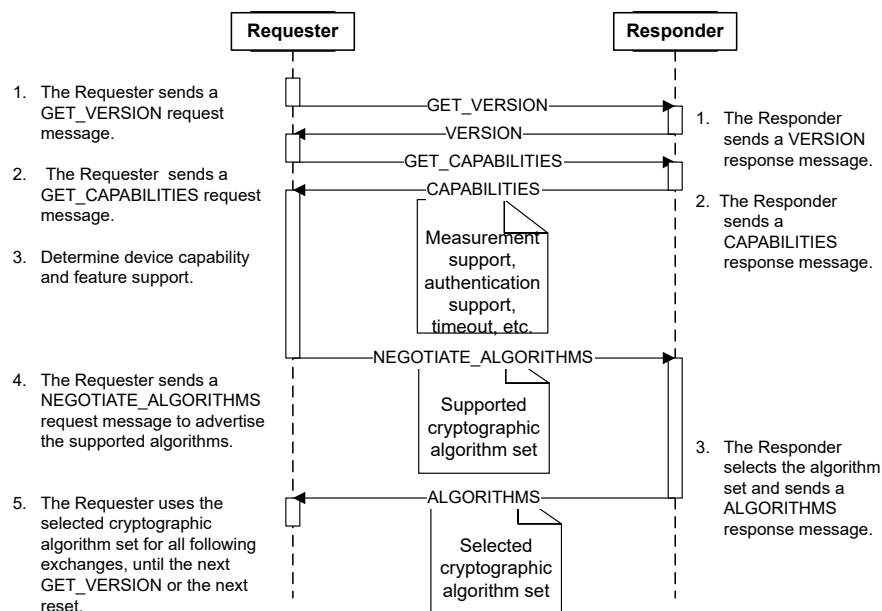
- [Capability discovery and negotiation](#)
- [Responder identity authentication](#)
- [Measurement](#)
- [Key agreement for secure-channel establishment](#)

10.1 Capability discovery and negotiation

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

[Figure 5 — Capability discovery and negotiation flow](#) shows the high-level request-response flow and sequence for the capability discovery and negotiation:

Figure 5 — Capability discovery and negotiation flow



234 10.1.1 Negotiated state preamble

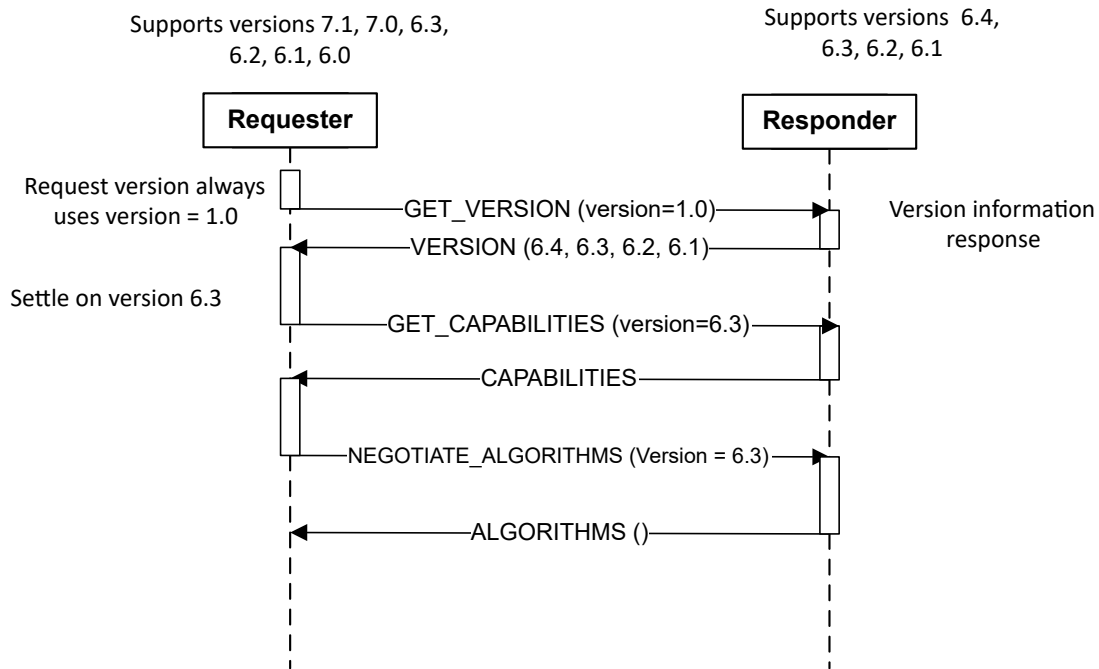
- 235 The **VCA** (*Version-Capabilities-Algorithms*) shall be the concatenation of messages **GET_VERSION** , **VERSION** , **GET_CAPABILITIES** , **CAPABILITIES** , **NEGOTIATE_ALGORITHMS** , and **ALGORITHMS** last exchanged between the Requester and the Responder.
- 236 If the two endpoints do not support session key establishment with the PSK (Pre-Shared Key) option, or if the two endpoints support PSK but the negotiated capabilities and algorithms are not provisioned to both endpoints alongside the PSK, then the Requester shall issue **GET_VERSION** , **GET_CAPABILITIES** , and **NEGOTIATE_ALGORITHMS** to construct **VCA** .
- 237 If the Responder supports caching the negotiated state (**CACHE_CAP=1**), the Requester might not issue **GET_VERSION** , **GET_CAPABILITIES** , and **NEGOTIATE_ALGORITHMS** . In this case, the Requester and the Responder shall store the most recent **VCA** as part of the Negotiated State.
- 238 If the two endpoints support session key establishment with the PSK and if the negotiated capabilities and algorithms (the **C** and **A** of **VCA**) are provisioned to both endpoints alongside the PSK, then the Requester shall not issue **GET_CAPABILITIES** and **NEGOTIATE_ALGORITHMS** .

239 10.2 GET_VERSION request and VERSION response messages

- 240 This request message shall retrieve the SPDM version of an endpoint. [Table 8 — GET_VERSION request message format](#) shows the **GET_VERSION** request message format and [Table 9 — Successful VERSION response message format](#) shows the **VERSION** response message format.
- 241 In all future SPDM versions, the **GET_VERSION** and **VERSION** response messages will be backward compatible with all earlier versions.
- 242 The Requester shall begin the discovery process by sending a **GET_VERSION** request message with the value of the **SPDMVersion** field set to **0x10** . All Responders shall always support the **GET_VERSION** request message with major version **0x1** and provide a **VERSION** response containing all supported versions, as [Table 8 — GET_VERSION request message format](#) describes.
- 243 The Requester shall consult the **VERSION** response to select a common supported version, which should be the latest supported common version. The Requester shall use the selected version in all future communication of other requests. A Requester shall not issue other requests until it receives a successful **VERSION** response and identifies a common version that both sides support. A Responder shall not respond to the **GET_VERSION** request message with an **ERROR** message of **ErrorCode=ResponseNotReady** . The selected version shall be the version in the **SPDMVersion** field of the Request (other than **GET_VERSION**) immediately following the **GET_VERSION** request. If the Requester uses a version other than the selected version in a Request, the Responder should either return an **ERROR** message of **ErrorCode=VersionMismatch** or silently discard the Request.

- 244 A Requester can issue a `GET_VERSION` request message to a Responder at any time, which serves as an exception to [Requirements for Requesters](#) to allow for scenarios where a Requester is required to restart the protocol due to an internal error or Reset.
- 245 After receiving a valid `GET_VERSION` request, the Responder shall invalidate state and data associated with all previous requests from the same Requester. All active sessions between the Requester and the Responder are terminated, and information (such as session keys and session IDs) for those sessions should not be used anymore. Additionally, this message shall clear the previously *Negotiated State*, if any, in both the Requester and its corresponding Responder. An invalid `GET_VERSION` request that results in the Responder returning an error to the Requester shall not affect the session state. The `ERROR` message resulting from an invalid `GET_VERSION` request shall have the value of the `SPDMVersion` field set to `0x10`.
- 246 After sending the `VERSION` response for a `GET_VERSION` request, if the Responder completes a runtime code or configuration change for its hardware or firmware measurement and the change has taken effect, then the Responder shall either silently discard any request received outside of a session or respond with an `ERROR` message of `ErrorCode=RequestResynch` to any request received outside of a session, until a `GET_VERSION` request is received. For requests received within a session, the Responder shall follow the selected session policy that the Requester selects in [Table 70 — Session policy](#) at the time of session establishment.
- 247 [Figure 6 — Discovering the common major version](#) shows the process:
- 248 **Figure 6 — Discovering the common major version**

249



250 [Table 8 — GET_VERSION request message format](#) shows the `GET_VERSION` request message format:

251 **Table 8 — GET_VERSION request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be 0x10 (V1.0).
1	RequestResponseCode	1	Shall be 0x84 = GET_VERSION . See Table 4 — SPDM request codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.

252 [Table 9 — Successful VERSION response message format](#) shows the successful `VERSION` response message format:

253 **Table 9 — Successful VERSION response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be 0x10 (V1.0).

Byte offset	Field	Size (bytes)	Description
1	RequestResponseCode	1	Shall be 0x04 = VERSION . See Table 5 — SPDM response codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	Reserved	1	Reserved.
5	VersionNumberEntryCount	1	Number of version entries present in this table (=n).
6	VersionNumberEntry1:n	2 * n	16-bit version entry. See Table 10 — VersionNumberEntry definition . Each entry should be unique.

254 [Table 10 — VersionNumberEntry definition](#) shows the VersionNumberEntry definition:

255 **Table 10 — VersionNumberEntry definition**

Bit offset	Field	Description
[15:12]	MajorVersion	Shall be the version of the specification having changes that are incompatible with one or more functions in earlier major versions of the specification.
[11:8]	MinorVersion	Shall be the version of the specification having changes that are compatible with functions in earlier minor versions of this major version specification.
[7:4]	UpdateVersionNumber	Shall be the 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	Shall be the pre-release work-in-progress version of the specification. 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 might not be fully interoperable. Released versions shall have an Alpha value of zero (0).

256 10.3 GET_CAPABILITIES request and CAPABILITIES response messages

257 This request message shall retrieve the SPDM capabilities of an endpoint.

258 [Table 11 — GET_CAPABILITIES request message format](#) shows the GET_CAPABILITIES request message format.

259 [Table 12 — Successful CAPABILITIES response message format](#) shows the CAPABILITIES response message format.

260 [Table 13 — Flag fields definitions for the Requester](#) shows the flag fields definitions for the Requester.

261 Likewise, [Table 14 — Flag fields definitions for the Responder](#) shows the flag fields definitions for the Responder.

- 262 A Responder shall not respond to `GET_CAPABILITIES` request message with an `ERROR` message of `ErrorCode=ResponseNotReady` .
- 263 To properly support transferring of SPDM messages, the Requester and Responder shall indicate two buffer sizes:
- One for receiving a single SPDM transfer called `DataTransferSize`
 - One for indicating their maximum internal buffer size for processing a single assembled received SPDM message called `MaxSPDMmsgSize`
- 264 Additionally, the Requester and Responder can have a transmit buffer. The transmit buffer size is not communicated to the other SPDM endpoint, but it can be less than the `DataTransferSize` of the receiving SPDM endpoint.
- 265 Both the Requester and Responder shall support a minimum size for both the transmit and receive buffer to successfully transfer SPDM messages. The minimum size is referred to as `MinDataTransferSize`. For this version of the specification, the `MinDataTransferSize` shall be 42. This value is the size, in bytes, of the SPDM message with the largest size from this list, assuming all fields are present:
- `GET_VERSION`
 - `VERSION` assuming no versions returned contain `Alpha` versions in `VersionNumberEntry` and version entries are not duplicated.
 - `GET_CAPABILITIES`
 - `CAPABILITIES` with `Param1` in the `GET_CAPABILITIES` request set to 0.
 - `CHUNK_SEND` using the size of the SPDM Header for the size of the `SPDMchunk` field.
 - `CHUNK_SEND_ACK` using the maximum size of `ERROR` message for the size of the `ResponseToLargeRequest` field.
 - `CHUNK_GET`
 - `CHUNK_RESPONSE` using the size of SPDM Header for the size of the `SPDMchunk` field.
 - `ERROR` using the maximum size for the `ExtendedErrorData`
- 266 The `GET_CAPABILITIES` request with Extended capabilities (Bit 0 of `Param1` set to a value of 1) is only allowed if both the Requester and Responder support the [Large SPDM message transfer mechanism](#) (`CHUNK_CAP=1`). If the `GET_CAPABILITIES` request sets Bit 0 of `Param1` to a value of 1, then the Responder shall use the value for `DataTransferSize` and `MaxSPDMmsgSize` from the request for the transmission of the `CAPABILITIES` response. A Responder can report that it needs to transmit the response in smaller transfers by sending an `ERROR` message of `ErrorCode=LargeResponse` . If the `GET_CAPABILITIES` request sets Bit 0 of `Param1` to a value of 1 and the Responder does not support the [Large SPDM message transfer mechanism](#) (`CHUNK_CAP=0`), the Responder shall send an `ERROR` message of `ErrorCode=InvalidRequest` .

267 **Table 11 — `GET_CAPABILITIES` request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0xE1</code> = <code>GET_CAPABILITIES</code> . See Table 4 — SPDM request codes .
2	Param1	1	Shall be the extended capabilities to include in the response. <ul style="list-style-type: none"> Bit 0. If set in the requests, the Responder shall include the Supported Algorithms Block in its <code>CAPABILITIES</code> response if it supports this extended capability. If the Requester does not support the Large SPDM message transfer mechanism (<code>CHUNK_CAP=0</code>), this bit shall be 0. All other values reserved.
3	Param2	1	Reserved.
4	Reserved	1	Reserved.
5	CTExponent	1	Shall be exponent of base 2, which is used to calculate <code>CT</code> . See Table 7 — Timing specification for SPDM messages . The equation for <code>CT</code> shall be $2^{\text{CTExponent}}$ microseconds (μs). For example, if <code>CTExponent</code> is 10, <code>CT</code> is $2^{10} = 1024 \mu\text{s}$.
6	Reserved	2	Reserved.
8	Flags	4	See Table 13 — Flag fields definitions for the Requester .
12	DataTransferSize	4	This field shall indicate the maximum buffer size, in bytes, of the Requester for receiving a single and complete SPDM message whose message size is less than or equal to the value in this field. The value of this field shall be equal to or greater than <code>MinDataTransferSize</code> . The <code>DataTransferSize</code> shall exclude transport headers, encryption headers, and <code>MAC</code> . This field helps the sender of the SPDM message know whether or not it needs to utilize the Large SPDM message transfer mechanism .
16	MaxSPDMmsgSize	4	If the Requester supports the Large SPDM message transfer mechanism , this field shall indicate the maximum size, in bytes, of the internal buffer of a Requester used to reassemble a single and complete Large SPDM message . This field shall be greater than or equal to <code>DataTransferSize</code> . This buffer size is most helpful when transferring a Large SPDM message in multiple chunks because it tells the sender whether or not there is enough memory for the fully reassembled SPDM message. If the Requester does not support the Large SPDM message transfer mechanism, this field shall be equal to the <code>DataTransferSize</code> of the Requester.

268 **Table 12 — Successful CAPABILITIES response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x61</code> = CAPABILITIES . See Table 5 — SPDM response codes .
2	Param1	1	<p>Shall be the extended capabilities included in the response.</p> <ul style="list-style-type: none"> Bit 0. If the request message sets the Supported Algorithms extended capability bit and the Responder supports this extended capability, then the Responder shall set this bit in the response and shall include the Supported Algorithms Block in its <code>CAPABILITIES</code> response. If the Responder does not support this extended capability or does not support the Large SPDM message transfer mechanism (<code>CHUNK_CAP=0</code>), this bit shall be 0. All other values reserved.
3	Param2	1	Reserved.
4	Reserved	1	Reserved.
5	CTExponent	1	<p>Shall be the exponent of base 2, which used to calculate <code>CT</code> .</p> <p>See Table 7 — Timing specification for SPDM messages.</p> <p>The equation for <code>CT</code> shall be $2^{\text{CTExponent}}$ microseconds (μs).</p> <p>For example, if <code>CTExponent</code> is 10, <code>CT</code> is $2^{10} = 1024 \mu\text{s}$.</p>
6	Reserved	2	Reserved.
8	Flags	4	See Table 14 — Flag fields definitions for the Responder .
12	DataTransferSize	4	This field shall indicate the maximum buffer size, in bytes, of the Responder for receiving a single and complete SPDM message whose message size is less than or equal to the value in this field. The value of this field shall be equal to or greater than <code>MinDataTransferSize</code> . The <code>DataTransferSize</code> shall exclude transport headers, encryption headers, and <code>MAC</code> . This field helps the sender of the SPDM message know whether or not it needs to utilize the Large SPDM message transfer mechanism .
16	MaxSPDMmsgSize	4	<p>If the Responder supports the Large SPDM message transfer mechanism, this field shall indicate the maximum size, in bytes, of the internal buffer of a Responder used to reassemble a single and complete Large SPDM message. This field shall be greater than or equal to <code>DataTransferSize</code> . This buffer size is most helpful when transferring a Large SPDM message in multiple chunks because it tells the sender whether or not there is enough memory for the fully reassembled SPDM message.</p> <p>If the Responder does not support the Large SPDM message transfer mechanism, this field shall be equal to the <code>DataTransferSize</code> of the Responder.</p>
20	SupportedAlgorithms	AlgSize or 0	If present, this field shall be <code>AlgSize</code> in size and the format of the field shall be as described in Supported algorithms block . If Bit 0 of <code>Param1</code> does not indicate that the Supported Algorithm extended capability is included in this response, then this field shall be absent.

269 As described in other parts of this specification, a Requester or Responder can reverse roles or take on both roles for certain SPDM messages and flows. Thus, an SPDM endpoint cannot send a Large SPDM message that exceeds the `MaxSPDMmsgSize` of the receiving SPDM endpoint. Specifically, a requesting SPDM endpoint shall not send a request that exceeds the size of `MaxSPDMmsgSize` of the responding SPDM endpoint. Likewise, a responding SPDM endpoint shall not send a response that exceeds the size of `MaxSPDMmsgSize` of the requesting SPDM endpoint. If the size of a response message exceeds the size of the `MaxSPDMmsgSize` of the requesting SPDM endpoint, the responding SPDM endpoint shall respond with an `ERROR` message of `ErrorCode=ResponseTooLarge`. If the size of a request message exceeds the size of the `MaxSPDMmsgSize` of the responding SPDM endpoint, the responding SPDM endpoint shall either respond with an `ERROR` message of `ErrorCode=RequestTooLarge` or silently discard the request. Additionally, an SPDM endpoint should provide graceful error handling (for example, buffer overflow/underflow protection) in the event that it receives an SPDM message that exceeds its `MaxSPDMmsgSize`.

270 [Table 13 — Flag fields definitions for the Requester](#) shows the flag fields definitions for the Requester.

271 Unless otherwise stated, if a Requester indicates support for a capability associated with an SPDM request or response message, it means the Requester can receive the corresponding request and produce a successful response. In other words, the Requester is acting as a Responder to that SPDM request associated with that capability. For example, if a Requester sets the `CERT_CAP` bit to 1, the Requester can receive a `GET_CERTIFICATE` request and send back a successful `CERTIFICATE` response message.

272 `AlgSize` is the size of the [Supported algorithms block](#). If the Supported Algorithms Block is not included in the response, then the `SupportedAlgorithms` field shall be absent.

273 **Table 13 — Flag fields definitions for the Requester**

Byte offset	Bit offset	Field	Description
0	0	Reserved	Reserved.
0	1	CERT_CAP	If set, Requester supports <code>DIGESTS</code> and <code>CERTIFICATE</code> response messages. Shall be 0b if the Requester does not support asymmetric algorithms.
0	2	CHAL_CAP	DEPRECATED: If set, Requester supports <code>CHALLENGE_AUTH</code> response message.
0	[5:3]	Reserved	Reserved.
0	6	ENCRYPT_CAP	If set, Requester supports message encryption in a secure session. If set, when the Requester chooses to start a secure session, the Requester shall send one of the Session-Secrets-Exchange request messages supported by the Responder. This capability applies to all phases of a secure session.
0	7	MAC_CAP	If set, Requester supports message authentication in a secure session. If set, when the Requester chooses to start a secure session, the Requester shall send one of the Session-Secrets-Exchange request messages supported by the Responder. This capability applies to all phases of a secure session. <code>MAC_CAP</code> is not the same as the <code>HMAC</code> in the <code>RequesterVerifyData</code> or <code>ResponderVerifyData</code> fields of Session-Secrets-Exchange and Session-Secrets-Finish messages.

Byte offset	Bit offset	Field	Description
1	0	MUT_AUTH_CAP	If set, Requester supports mutual authentication.
1	1	KEY_EX_CAP	If set, Requester supports <code>KEY_EXCHANGE</code> request message. If set, <code>ENCRYPT_CAP</code> or <code>MAC_CAP</code> shall be set.
1	[3:2]	PSK_CAP	<p>Pre-Shared Key capabilities of the Requester.</p> <ul style="list-style-type: none"> • <code>00b</code> . Requester does not support Pre-Shared Key capabilities. • <code>01b</code> . Requester supports Pre-Shared Key • <code>10b</code> and <code>11b</code> . Reserved. <p>If supported, <code>ENCRYPT_CAP</code> or <code>MAC_CAP</code> shall be set.</p>
1	4	ENCAP_CAP	<p>If set, Requester supports <code>GET_ENCAPSULATED_REQUEST</code> , <code>ENCAPSULATED_REQUEST</code> , <code>DELIVER_ENCAPSULATED_RESPONSE</code> , and <code>ENCAPSULATED_RESPONSE_ACK</code> messages. Additionally, the transport may require the Requester to support these messages.</p> <p><code>ENCAP_CAP</code> was previously deprecated because Basic mutual authentication is deprecated. Deprecation is removed since some messages, such as <code>KEY_UPDATE</code> , do not require mutual authentication but still require <code>ENCAP_CAP</code> .</p>
1	5	HBEAT_CAP	If set, Requester supports <code>HEARTBEAT</code> messages.
1	6	KEY_UPD_CAP	If set, Requester supports <code>KEY_UPDATE</code> messages.
1	7	HANDSHAKE_IN_THE_CLEAR_CAP	<p>If set, the Requester can support a Responder that can only send and receive all SPDM messages exchanged during the Session Handshake Phase in the clear (such as without encryption and message authentication). Application data is encrypted and/or authenticated using the negotiated cryptographic algorithms as normal. Setting this bit leads to changes in the contents of certain SPDM messages, as discussed in other parts of this specification.</p> <p>If this bit is cleared, the Requester signals that it requires encryption and/or message authentication of SPDM messages exchanged during the Session Handshake Phase.</p> <p>If the Requester supports Pre-Shared Keys (<code>PSK_CAP</code> is <code>01b</code>) and does not support asymmetric key exchange (<code>KEY_EX_CAP</code> is <code>0b</code>), then this bit shall be zero. If the Requester does not support encryption and message authentication, then this bit shall be zero.</p> <p>In other words, this bit indicates whether <code>MAC_CAP</code> and <code>ENCRYPT_CAP</code> is involved accordingly in the handshake phase of a secure session or both encryption and message authentication capabilities are disabled in the session handshake phase of a secure session.</p>
2	0	PUB_KEY_ID_CAP	If set, the public key of the Requester was provisioned to the Responder. The transport layer is responsible for identifying the Responder. In this case, the <code>CERT_CAP</code> of the Requester shall be <code>0</code> .
2	1	CHUNK_CAP	If set, Requester supports Large SPDM message transfer mechanism messages.

Byte offset	Bit offset	Field	Description
2	[5:2]	Reserved	Reserved.
2	[7:6]	EP_INFO_CAP	<p>The <code>ENDPOINT_INFO</code> response capabilities of the Requester.</p> <ul style="list-style-type: none"> <code>00b</code> . The Requester does not support <code>ENDPOINT_INFO</code> response capabilities. <code>01b</code> . The Requester supports the <code>ENDPOINT_INFO</code> response but cannot perform signature generation for this response. <code>10b</code> . The Requester supports the <code>ENDPOINT_INFO</code> response and can generate signatures for this response. <code>11b</code> . Reserved.
3	0	Reserved	Reserved.
3	1	EVENT_CAP	If set, the Requester is an Event Notifier . See Event mechanism for details.
3	[3:2]	MULTI_KEY_CAP	<p>Shall be the Multiple Asymmetric Key capabilities of the Requester.</p> <ul style="list-style-type: none"> <code>00b</code> . Requester does not support Multiple Asymmetric Key capabilities. <code>01b</code> . Requester only supports Multiple Asymmetric Key capabilities. <code>10b</code> . Requester supports Multiple Asymmetric Key capabilities, and Responder can use <code>RequesterMultiKeyConnSel</code> as Multiple Asymmetric Key Negotiation describes. <code>11b</code> . Reserved. <p>If set to <code>01b</code> or <code>10b</code> , the Requester supports more than one key pair for at least one asymmetric algorithm for use in Requester authentication such as in mutual authentication. In the case of mutual authentication, these are the key pairs belonging to the Requester.</p>
3	[7:4]	Reserved	Reserved.

274 [Table 14 — Flag fields definitions for the Responder](#) shows the flag fields definitions for the Responder.

275 Unless otherwise stated, if a Responder indicates support for a capability associated with an SPDM request or response message, it means the Responder can receive the corresponding request and produce a successful response. For example, if a Responder sets the `CERT_CAP` bit to `1` , the Responder can receive a `GET_CERTIFICATE` request and send back a successful `CERTIFICATE` response message.

276 **Table 14 — Flag fields definitions for the Responder**

Byte offset	Bit offset	Field	Description
0	0	CACHE_CAP	If set, the Responder supports the ability to cache the <i>Negotiated State</i> 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. Shall be 0b if the Responder does not support asymmetric algorithms.
0	2	CHAL_CAP	If set, Responder supports CHALLENGE_AUTH response message.
0	[4:3]	MEAS_CAP	<p>MEASUREMENTS response capabilities of the Responder.</p> <ul style="list-style-type: none"> • 00b . The Responder does not support MEASUREMENTS response capabilities. • 01b . The Responder supports MEASUREMENTS response but cannot perform signature generation for this response. • 10b . The Responder supports MEASUREMENTS response and can generate signatures for this response. • 11b . Reserved. <p>Note that, apart from affecting MEASUREMENTS , this capability also affects Param2 of CHALLENGE , Param1 of KEY_EXCHANGE , Param1 of PSK_EXCHANGE , and the MeasurementSummaryHash field of KEY_EXCHANGE_RSP , CHALLENGE_AUTH , and PSK_EXCHANGE_RSP . See the respective request and response clauses for further details.</p>
0	5	MEAS_FRESH_CAP	<ul style="list-style-type: none"> • 0 . As part of MEASUREMENTS response message, the Responder may return MEASUREMENTS that were computed during the last Responder's Reset. • 1 . The Responder supports recomputing all MEASUREMENTS without requiring a Reset and shall always return fresh MEASUREMENTS as part of MEASUREMENTS response message.
0	6	ENCRYPT_CAP	If set, Responder supports message encryption in a secure session. If set, PSK_CAP or KEY_EX_CAP shall be set accordingly to indicate support. This capability applies to all phases of a secure session.
0	7	MAC_CAP	If set, Responder supports message authentication in a secure session. If set, PSK_CAP or KEY_EX_CAP shall be set accordingly to indicate support. This capability applies to all phases of a secure session. MAC_CAP is not the same as the HMAC in the RequesterVerifyData or ResponderVerifyData fields of Session-Secrets-Exchange and Session-Secrets-Finish messages.
1	0	MUT_AUTH_CAP	If set, Responder supports mutual authentication.
1	1	KEY_EX_CAP	If set, Responder supports KEY_EXCHANGE_RSP response message. If set, ENCRYPT_CAP or MAC_CAP shall be set.

Byte offset	Bit offset	Field	Description
1	[3:2]	PSK_CAP	<p>Pre-Shared Key capabilities of the Responder.</p> <ul style="list-style-type: none"> 00b . Responder does not support Pre-Shared Key capabilities. 01b . Responder supports Pre-Shared Key but does not provide ResponderContext for session key derivation. 10b . Responder supports Pre-Shared Key and provides ResponderContext for session key derivation. 11b . Reserved. <p>If supported, ENCRYPT_CAP or MAC_CAP shall be set.</p>
1	4	ENCAP_CAP	<p>If set, Responder supports GET_ENCAPSULATED_REQUEST , ENCAPSULATED_REQUEST , DELIVER_ENCAPSULATED_RESPONSE , and ENCAPSULATED_RESPONSE_ACK messages. Additionally, the transport may require the Responder to support these messages.</p> <p>ENCAP_CAP was previously deprecated because Basic mutual authentication is deprecated. Deprecation is removed since some messages, such as KEY_UPDATE , do not require mutual authentication but still require ENCAP_CAP .</p>
1	5	HBEAT_CAP	If set, Responder supports HEARTBEAT messages.
1	6	KEY_UPD_CAP	If set, Responder supports KEY_UPDATE messages.
1	7	HANDSHAKE_IN_THE_CLEAR_CAP	<p>If set, the Responder can only send and receive messages without encryption and message authentication during the Session Handshake Phase. If set, KEY_EX_CAP shall also be set. Setting this bit leads to changes in the contents of certain SPDM messages, as discussed in other parts of this specification.</p> <p>If the Responder supports Pre-Shared Keys (PSK_CAP is 01b) and does not support asymmetric key exchange (KEY_EX_CAP is 0b), then this bit shall be zero. If the Responder does not support encryption and message authentication, then this bit shall be zero.</p> <p>In other words, this bit indicates whether message authentication and/or encryption (MAC_CAP and ENCRYPT_CAP) are used in the handshake phase of a secure session.</p>
2	0	PUB_KEY_ID_CAP	If set, the public key of the Responder was provisioned to the Requester. The transport layer is responsible for identifying the Requester. In this case, CERT_CAP and ALIAS_CERT_CAP of the Responder shall both be 0 .
2	1	CHUNK_CAP	If set, Responder supports Large SPDM message transfer mechanism messages.
2	2	ALIAS_CERT_CAP	If set, the Responder uses the AliasCert model. See Identity provisioning for details.
2	3	SET_CERT_CAP	If set, Responder supports SET_CERTIFICATE_RSP response messages.
2	4	CSR_CAP	If set, Responder supports CSR response messages. If this bit is set, SET_CERT_CAP shall be set.

Byte offset	Bit offset	Field	Description
2	5	CERT_INSTALL_RESET_CAP	If set, Responder may return an <code>ERROR</code> message of <code>ErrorCode=ResetRequired</code> to complete a certificate provisioning request. If this bit is set, <code>SET_CERT_CAP</code> shall be set and <code>CSR_CAP</code> can be set.
2	[7:6]	EP_INFO_CAP	The <code>ENDPOINT_INFO</code> response capabilities of the Responder. <ul style="list-style-type: none"> • <code>00b</code> . The Responder does not support <code>ENDPOINT_INFO</code> response capabilities. • <code>01b</code> . The Responder supports the <code>ENDPOINT_INFO</code> response but cannot perform signature generation for this response. • <code>10b</code> . The Responder supports the <code>ENDPOINT_INFO</code> response and can generate signatures for this response. • <code>11b</code> . Reserved.
3	0	MEL_CAP	If set, Responder supports <code>MEASUREMENT_EXTENSION_LOG</code> response message.
3	1	EVENT_CAP	If set, the Responder is an Event Notifier . See Event mechanism for details.
3	[3:2]	MULTI_KEY_CAP	Shall be the Multiple Asymmetric Key capabilities of the Responder. <ul style="list-style-type: none"> • <code>00b</code> . Responder does not support Multiple Asymmetric Key capabilities. • <code>01b</code> . Responder only supports Multiple Asymmetric Key capabilities. • <code>10b</code> . Responder supports Multiple Asymmetric Key capabilities, and Requester can use <code>ResponderMultiKeyConn</code> as Multiple Asymmetric Key Negotiation describes. • <code>11b</code> . Reserved. <p>If set to <code>01b</code> or <code>10b</code> , the Responder supports more than one key pair for at least one asymmetric algorithm for the SPDM connection to use in Responder authentication.</p>
3	4	GET_KEY_PAIR_INFO_CAP	If set, Responder supports <code>KEY_PAIR_INFO</code> response messages. If the Responder sets <code>MULTI_KEY_CAP</code> , this bit shall also be set.
3	5	SET_KEY_PAIR_INFO_CAP	If set, Responder supports <code>SET_KEY_PAIR_INFO_ACK</code> response message.
3	[7:6]	Reserved	Reserved.

277 In the case where an SPDM implementation incorrectly returns an illegal combination of capability flags as they are defined by this specification (for example, `ENCRYPT_CAP` is set but both `KEY_EX_CAP` and `PSK_CAP` are cleared), the following guidance is provided: If a Responder detects an illegal capability flag combination reported by the Requester, it shall issue an `ERROR` message of `ErrorCode=InvalidRequest` .

278 10.3.1 Supported algorithms block

279 The Supported Algorithms Block reports all options from the `ALGORITHMS` response that are supported by the Responder. The Supported Algorithms Block shall conform to the [Table 15 — NEGOTIATE_ALGORITHMS request message format](#), including all fields from `Param1` through the end of the message, inclusive. When constructing the Supported Algorithms Block, the Responder shall follow all requirements for the Requester, and shall set all bits and values that reflect algorithms that the Responder supports.

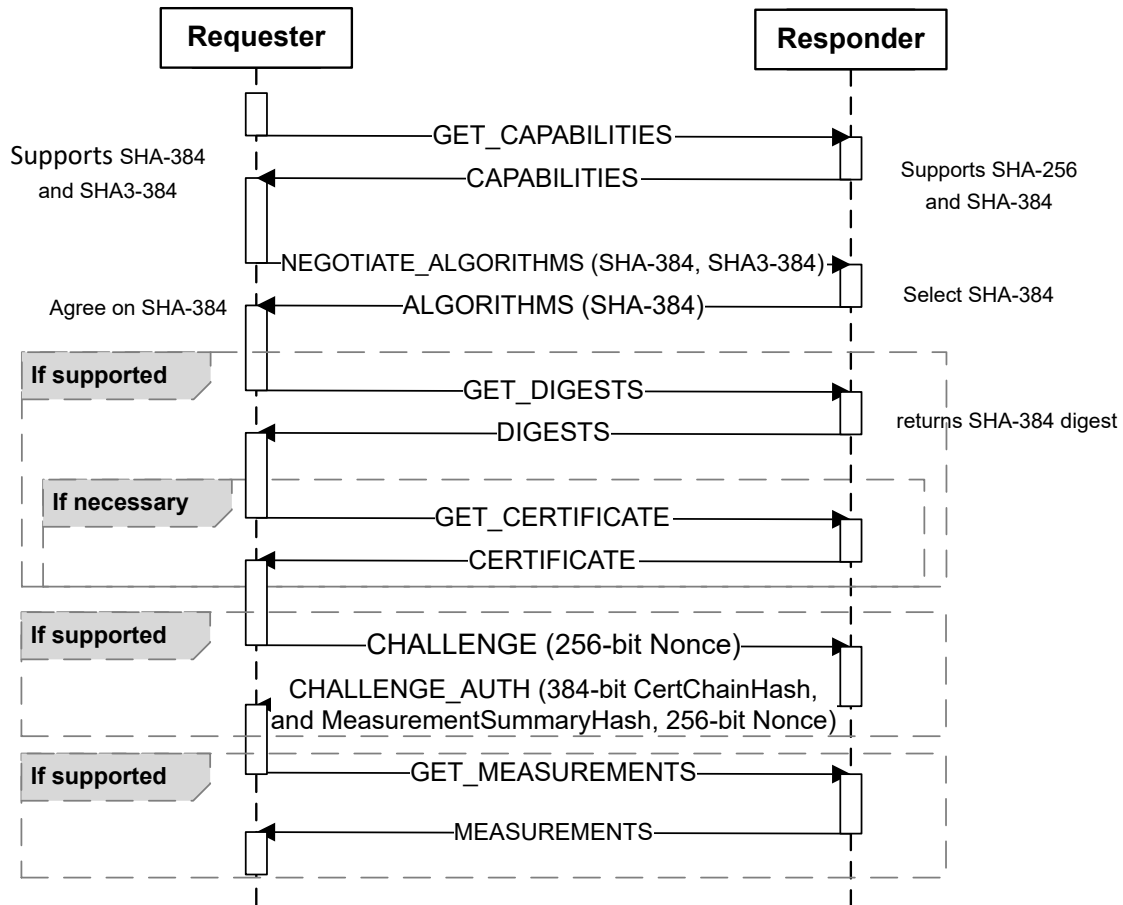
280 **10.4 NEGOTIATE_ALGORITHMS request and ALGORITHMS response** 281 **messages**

281 This request message shall negotiate cryptographic algorithms. In SPDM, the Requester issues
NEGOTIATE_ALGORITHMS to indicate which cryptographic algorithm(s) it supports for each type of cryptographic
operation, and the Responder selects one algorithm of each type using the ALGORITHMS response message. The
selected algorithms shall be used for all relevant cryptographic operations for the duration of the connection. The
criteria a Responder uses to determine which algorithm to select when more than one are supported by both
endpoints are outside the scope of this specification.

282 [Figure 7 — Hashing algorithm selection: Example 1](#) illustrates how two endpoints negotiate a base hashing algorithm.
Endpoint A issues a NEGOTIATE_ALGORITHMS request message, and endpoint B returns a selected mutually supported
algorithm in the ALGORITHMS response.

283 **Figure 7 — Hashing algorithm selection: Example 1**

284



285 If the Requester and Responder support no common algorithms of a particular type, the Responder shall issue an **ALGORITHMS** response message with all appropriate selection field values set to zero to indicate that no selection was made. The Responder should respond to all subsequent requests by this Requester with an **ERROR** message of **ErrorCode=RequestResynch**. The Responder may continue to operate with limited functionality for operations that do not require negotiated cryptographic algorithms.

286 A Requester shall not issue a **NEGOTIATE_ALGORITHMS** request message until it receives a successful **CAPABILITIES** response message.

287 After a Requester issues a **NEGOTIATE_ALGORITHMS** request, it shall not issue any other SPDM requests, with the exception of **GET_VERSION**, until it receives a successful **ALGORITHMS** response message.

288 A Responder shall not respond to a **NEGOTIATE_ALGORITHMS** request message with an **ERROR** message of **ErrorCode=ResponseNotReady**.

- 289 For each algorithm type, a Responder shall not select both an SPDM-enumerated algorithm and an extended algorithm.
- 290 The SPDM protocol accounts for the possibility that both endpoints issue `NEGOTIATE_ALGORITHMS` request messages independently of each other. In this case, the endpoint A Requester and endpoint B Responder communication pair might select a different algorithm from the one selected by the endpoint B Requester and endpoint A Responder communication pair.

291 [Table 15 — NEGOTIATE_ALGORITHMS request message format](#) shows the `NEGOTIATE_ALGORITHMS` request message format.

292 **Table 15 — NEGOTIATE_ALGORITHMS request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0xE3</code> = <code>NEGOTIATE_ALGORITHMS</code> . See Table 4 — SPDM request codes .
2	Param1	1	Shall be the number of algorithm structure tables in this request using <code>ReqAlgStruct</code> .
3	Param2	1	Reserved.
4	Length	2	Shall be the length of the entire request message, in bytes. Length shall be less than or equal to 128 bytes.
6	MeasurementSpecification	1	Bit mask. The Measurement specification field format table defines the format for this field. For each defined measurement specification a Requester supports, the Requester can set the appropriate bits.
7	OtherParamsSupport	1	Shall be the selection bit mask. Bits [3:0] - See Opaque Data Format Support and Selection Table Bit 4 - This field shall be the <code>ResponderMultiKeyConn</code> field as Multiple Asymmetric Key Negotiation describes. Bits [7:5] - Reserved.

Byte offset	Field	Size (bytes)	Description
8	BaseAsymAlgo	4	<p>Shall be the bit mask listing Requester-supported SPDM-enumerated asymmetric key signature algorithms for the purpose of signature verification. If the Requester does not support any request/response pair that requires signature verification, this value shall be set to zero. If the Requester will not send any requests that require a signature, this value should be set to zero. Let <code>SigLen</code> be the size of the signature in bytes.</p> <ul style="list-style-type: none"> • Byte 0 Bit 0. TPM_ALG_RSASSA_2048 where <code>SigLen</code> =256. • Byte 0 Bit 1. TPM_ALG_RSAPSS_2048 where <code>SigLen</code> =256. • Byte 0 Bit 2. TPM_ALG_RSASSA_3072 where <code>SigLen</code> =384. • Byte 0 Bit 3. TPM_ALG_RSAPSS_3072 where <code>SigLen</code> =384. • Byte 0 Bit 4. TPM_ALG_ECDSA_ECC_NIST_P256 where <code>SigLen</code> =64 (32-byte <code>r</code> followed by 32-byte <code>s</code>). • Byte 0 Bit 5. TPM_ALG_RSASSA_4096 where <code>SigLen</code> =512. • Byte 0 Bit 6. TPM_ALG_RSAPSS_4096 where <code>SigLen</code> =512. • Byte 0 Bit 7. TPM_ALG_ECDSA_ECC_NIST_P384 where <code>SigLen</code> =96 (48-byte <code>r</code> followed by 48-byte <code>s</code>). • Byte 1 Bit 0. TPM_ALG_ECDSA_ECC_NIST_P521 where <code>SigLen</code> =132 (66-byte <code>r</code> followed by 66-byte <code>s</code>). • Byte 1 Bit 1. TPM_ALG_SM2_ECC_SM2_P256 where <code>SigLen</code> =64 (32-byte <code>SM2_R</code> followed by 32-byte <code>SM2_S</code>). • Byte 1 Bit 2. EdDSA ed25519 where <code>SigLen</code> =64 (32-byte <code>R</code> followed by 32-byte <code>S</code>). • Byte 1 Bit 3. EdDSA ed448 where <code>SigLen</code> =114 (57-byte <code>R</code> followed by 57-byte <code>S</code>). • All other values reserved.
12	BaseHashAlgo	4	<p>Shall be the bit mask listing Requester-supported SPDM-enumerated cryptographic hashing algorithms. If the Requester does not support any request/response pair that requires hashing operations, this value shall be set to zero.</p> <ul style="list-style-type: none"> • 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 • Byte 0 Bit 6. TPM_ALG_SM3_256 • All other values reserved.
16	Reserved	12	Reserved.
28	ExtAsymCount	1	<p>Shall be the number of Requester-supported extended asymmetric key signature algorithms (=A) for the purpose of signature verification. $A + E + \text{ExtAlgCount2} + \text{ExtAlgCount3} + \text{ExtAlgCount4} + \text{ExtAlgCount5}$ shall be less than or equal to 20. If the Requester does not support any request/response pair that requires signature verification, this value shall be set to zero.</p>

Byte offset	Field	Size (bytes)	Description
29	ExtHashCount	1	Shall be the number of Requester-supported extended hashing algorithms (=E). $A + E + \text{ExtAlgCount2} + \text{ExtAlgCount3} + \text{ExtAlgCount4} + \text{ExtAlgCount5}$ shall be less than or equal to 20. If the Requester does not support any request/response pair that requires hashing operations, this value shall be set to zero.
30	Reserved	1	Reserved.
31	MELspecification	1	Shall be the bit mask. The Measurement Extension Log specification field format table defines the format for this field. The Requester shall set the corresponding bit for each supported measurement extension log (MEL) specification.
32	ExtAsym	$4 * A$	Shall be the list of Requester-supported extended asymmetric key signature algorithms for the purpose of signature verification. Table 27 — Extended Algorithm field format describes the format of this field.
$32 + 4 * A$	ExtHash	$4 * E$	Shall be the list of the extended hashing algorithms supported by Requester. Table 27 — Extended Algorithm field format describes the format of this field.
$32 + 4 * A + 4 * E$	ReqAlgStruct	AlgStructSize	See the AlgStructure request field.

293 AlgStructSize is the sum of the size of the following algorithm structure tables. The algorithm structure table shall be present only if the Requester supports that AlgType. AlgType shall monotonically increase for subsequent entries.

294 [Table 16 — Algorithm request structure](#) shows the Algorithm request structure:

295 **Table 16 — Algorithm request structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	Shall be the type of algorithm. <ul style="list-style-type: none"> • 0x00 and 0x01. Reserved. • 0x02. DHE. • 0x03. AEADCipherSuite. • 0x04. ReqBaseAsymAlg. • 0x05. KeySchedule. • All other values reserved.

Byte offset	Field	Size (bytes)	Description
1	AlgCount	1	Shall be the Requester-supported fixed algorithms. <ul style="list-style-type: none"> Bit [7:4]. Number of bytes required to describe Requester-supported SPDM-enumerated fixed algorithms (=FixedAlgCount). FixedAlgCount + 2 shall be a multiple of 4. Bit [3:0]. Number of Requester-supported extended algorithms (= ExtAlgCount).
2	AlgSupported	FixedAlgCount	Shall be the bit mask listing Requester-supported SPDM-enumerated algorithms.
2 + FixedAlgCount	AlgExternal	4 * ExtAlgCount	Shall be the list of Requester-supported extended algorithms. Table 27 — Extended Algorithm field format describes the format of this field.

296 The following tables describe the Algorithm request structures mapped to their respective types:

- [Table 17 — DHE structure](#)
- [Table 18 — AEAD structure](#)
- [Table 19 — ReqBaseAsymAlg structure](#)
- [Table 20 — KeySchedule structure](#)

297 **Table 17 — DHE structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	Shall be 0x02 = DHE
1	AlgCount	1	<ul style="list-style-type: none"> Bit [7:4]. Shall be a value of 2. Bit [3:0]. Number of Requester-supported extended DHE groups (= ExtAlgCount2).
2	AlgSupported	2	Shall be the bit mask listing Requester-supported SPDM-enumerated Diffie-Hellman Ephemeral (DHE) groups. Values in parentheses specify the size of the corresponding public values associated with each group. <ul style="list-style-type: none"> Byte 0 Bit 0. ffdhe2048 (D = 256). Byte 0 Bit 1. ffdhe3072 (D = 384). Byte 0 Bit 2. ffdhe4096 (D = 512). Byte 0 Bit 3. secp256r1 (D = 64, C = 32). Byte 0 Bit 4. secp384r1 (D = 96, C = 48). Byte 0 Bit 5. secp521r1 (D = 132, C = 66). Byte 0 Bit 6. SM2_P256 (Part 3 and Part 5 of GB/T 32918 specification) (D = 64, C = 32). All other values reserved.
4	AlgExternal	4 * ExtAlgCount2	Shall be the list of Requester-supported extended DHE groups. Table 27 — Extended Algorithm field format describes the format of this field.

298 **Table 18 — AEAD structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	Shall be the 0x03 = AEAD
1	AlgCount	1	<ul style="list-style-type: none"> Bit [7:4]. Shall be a value of 2. Bit [3:0]. Number of Requester-supported extended AEAD algorithms (= ExtAlgCount3).
2	AlgSupported	2	Shall be the bit mask listing Requester-supported SPDM-enumerated AEAD algorithms. <ul style="list-style-type: none"> Byte 0 Bit 0. AES-128-GCM. 128-bit key; 96-bit IV (initialization vector); tag size is specified by transport layer. Byte 0 Bit 1. AES-256-GCM. 256-bit key; 96-bit IV; tag size is specified by transport layer. Byte 0 Bit 2. CHACHA20_POLY1305. 256-bit key; 96-bit IV; 128-bit tag. Byte 0 Bit 3. AEAD_SM4_GCM. 128-bit key; 96-bit IV; tag size is specified by transport layer. All other values reserved.
4	AlgExternal	4 * ExtAlgCount3	Shall be the list of Requester-supported extended AEAD algorithms. Table 27 — Extended Algorithm field format describes the format of this field.

299 **Table 19 — ReqBaseAsymAlg structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	Shall be 0x04 = ReqBaseAsymAlg
1	AlgCount	1	<ul style="list-style-type: none"> Bit [7:4]. Shall be a value of 2. Bit [3:0]. Number of Requester-supported extended asymmetric key signature algorithms for the purpose of signature generation (= ExtAlgCount4).

Byte offset	Field	Size (bytes)	Description
2	AlgSupported	2	<p>Shall be the bit mask listing Requester-supported SPDM-enumerated asymmetric key signature algorithms for the purpose of signature generation. If the Requester does not support any request/response pair that requires signature generation, this value shall be set to zero.</p> <ul style="list-style-type: none"> 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. Byte 1 Bit 1. TPM_ALG_SM2_ECC_SM2_P256. Byte 1 Bit 2. EdDSA ed25519. Byte 1 Bit 3. EdDSA ed448. All other values reserved. <p>For details of <code>SigLen</code> for each algorithm, see Table 15 — NEGOTIATE_ALGORITHMS request message format.</p>
4	AlgExternal	4 * ExtAlgCount4	Shall be the list of Requester-supported extended asymmetric key signature algorithms for the purpose of signature generation. Table 27 — Extended Algorithm field format describes the format of this field.

300 **Table 20 — KeySchedule structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	Shall be <code>0x05</code> = KeySchedule
1	AlgCount	1	<ul style="list-style-type: none"> Bit [7:4]. Shall be a value of 2. Bit [3:0]. Number of Requester-supported extended key schedule algorithms (= <code>ExtAlgCount5</code>).
2	AlgSupported	2	<p>Shall be the bit mask listing Requester-supported SPDM-enumerated key schedule algorithms.</p> <ul style="list-style-type: none"> Byte 0 Bit 0. SPDM Key Schedule. All other values reserved.
4	AlgExternal	4 * ExtAlgCount5	Shall be the list of Requester-supported extended key schedule algorithms. Table 27 — Extended Algorithm field format describes the format of this field.

301 [Table 21 — ALGORITHMS response message format](#) shows the `ALGORITHMS` response message format.

302 **Table 21 — Successful ALGORITHMS response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x63</code> = <code>ALGORITHMS</code> . See Table 5 — SPDM response codes .
2	Param1	1	Shall be the number of algorithm structure tables in this request using <code>RespAlgStruct</code> .
3	Param2	1	Reserved.
4	Length	2	Shall be the length of the response message, in bytes.
6	MeasurementSpecificationSel	1	Bit mask. The Responder shall select one of the measurement specifications supported by the Requester and Responder. Thus, no more than one bit shall be set. The Measurement specification field format table defines the format for this field.
7	OtherParamsSelection	1	Shall be the selected Parameter Bit Mask. The Responder shall select one of the opaque data formats supported by the Requester. Thus, no more than one bit shall be set for the opaque data format. <ul style="list-style-type: none"> Bit [3:0]. See Opaque Data Format Support and Selection Table. Bit 4 - This field shall be the <code>RequesterMultiKeyConnSel</code> as Multiple Asymmetric Key Negotiation describes. Bit [7:5]. Reserved.
8	MeasurementHashAlgo	4	Shall be the bit mask indicating the SPDM-enumerated hashing algorithms used for measurements. <ul style="list-style-type: none"> Byte 0 Bit 0. Raw Bit Stream Only. Byte 0 Bit 1. TPM_ALG_SHA_256. Byte 0 Bit 2. TPM_ALG_SHA_384. Byte 0 Bit 3. TPM_ALG_SHA_512. Byte 0 Bit 4. TPM_ALG_SHA3_256. Byte 0 Bit 5. TPM_ALG_SHA3_384. Byte 0 Bit 6. TPM_ALG_SHA3_512. Byte 0 Bit 7. TPM_ALG_SM3_256. If the Responder supports measurements (<code>MEAS_CAP=01b</code> or <code>MEAS_CAP=10b</code> in its <code>CAPABILITIES</code> response) and if <code>MeasurementSpecificationSel</code> is non-zero, then exactly one bit in this bit field shall be set. Otherwise, the Responder shall set this field to <code>0</code>. All other values reserved. <p>A Responder shall select bit 0 only if it supports raw bit streams as the only form of measurement; otherwise, the Responder shall select one of the other bits.</p>

Byte offset	Field	Size (bytes)	Description
12	BaseAsymSel	4	Shall be the bit mask indicating the SPDM-enumerated asymmetric key signature algorithm selected for the purpose of signature generation. If the Responder does not support any request/response pair that requires signature generation, this value shall be set to zero. The Responder shall set no more than one bit.
16	BaseHashSel	4	Shall be the bit mask indicating the SPDM-enumerated hashing algorithm selected. If the Responder does not support any request/response pair that requires hashing operations, this value shall be set to zero. The Responder shall set no more than one bit.
20	Reserved	11	Reserved.
31	MELspecificationSel	1	Shall be the bit mask indicating MEL. The Responder shall select one of the MEL specifications supported by the Requester and Responder. No more than one bit shall be set. The Measurement Extension Log specification field format table defines the format for this field.
32	ExtAsymSelCount	1	Shall be the number of extended asymmetric key signature algorithms selected for the purpose of signature generation. Shall be either 0 or 1 (=A'). If the Responder does not support any request/response pair that requires signature generation, this value shall be set to zero.
33	ExtHashSelCount	1	Shall be the number of extended hashing algorithms selected. Shall be either 0 or 1 (=E'). If the Responder does not support any request/response pair that requires hashing operations, this value shall be set to zero.
34	Reserved	2	Reserved.
36	ExtAsymSel	4 * A'	Shall be the extended asymmetric key signature algorithm selected for the purpose of signature generation. 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'	Shall be the extended hashing algorithm selected. The Responder shall use this hashing algorithm during all subsequent response messages to the Requester. The Requester shall use this hashing algorithm during all subsequent applicable request messages to the Responder. The extended algorithm field format table describes the format of this field.
36 + 4 * A' + 4 * E'	RespAlgStruct	AlgStructSize	See Table 22 — Response AlgStructure field format .

303 AlgStructSize is the sum of the sizes of all the algorithm structure tables, as the following tables show. An algorithm structure table needs to be present only if the Responder supports that AlgType. AlgType shall monotonically increase for subsequent entries.

304 **Table 22 — Response AlgStructure field format**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	Shall be the type of algorithm. <ul style="list-style-type: none"> • 0x00 and 0x01. Reserved. • 0x02. DHE. • 0x03. AEADCipherSuite . • 0x04. ReqBaseAsymAlg . • 0x05. KeySchedule . • All other values reserved.
1	AlgCount	1	Shall be the bit mask listing Responder-supported fixed algorithm requested by the Requester. <ul style="list-style-type: none"> • Bit [7:4]. Number of bytes required to describe Requester-supported SPDM-enumerated fixed algorithms (=FixedAlgCount). FixedAlgCount + 2 shall be a multiple of 4. • Bit [3:0]. Number of Requester-supported, Responder-selected, extended algorithms (= ExtAlgCount'). This value shall be either 0 or 1.
2	AlgSupported	FixedAlgCount	Shall be the bit mask for indicating a Requester-supported, Responder-selected, SPDM-enumerated algorithm. Responder shall set at most one bit to 1.
2 + FixedAlgCount	AlgExternal	4 * ExtAlgCount'	If present: shall be a Requester-supported, Responder-selected, extended algorithm. Responder shall select at most one extended algorithm. Table 27 — Extended Algorithm field format describes the format of this field.

305 The following tables describe the algorithm types and their associated fixed fields:

- [Table 23 — DHE structure](#)
- [Table 24 — AEAD structure](#)
- [Table 25 — ReqBaseAsymAlg structure](#)
- [Table 26 — KeySchedule structure](#)
- [Table 27 — Extended Algorithm field format](#)

306 **Table 23 — DHE structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	Shall be 0x02 = DHE

Byte offset	Field	Size (bytes)	Description
1	AlgCount	1	<ul style="list-style-type: none"> Bit [7:4]. Shall be a value of 2. Bit [3:0]. Shall be the number of Requester-supported, Responder-selected, extended DHE groups (= <code>ExtAlgCount2'</code>). This value shall be either 0 or 1.
2	AlgSupported	2	<p>Shall be the bit mask for indicating a Requester-supported, Responder-selected, SPDM-enumerated DHE group. Values in parentheses specify the size of the corresponding public values associated with each group.</p> <ul style="list-style-type: none"> Byte 0 Bit 0. ffdhe2048 (D = 256). Byte 0 Bit 1. ffdhe3072 (D = 384). Byte 0 Bit 2. ffdhe4096 (D = 512). Byte 0 Bit 3. secp256r1 (D = 64, C = 32) Byte 0 Bit 4. secp384r1 (D = 96, C = 48). Byte 0 Bit 5. secp521r1 (D = 132, C = 66). Byte 0 Bit 6. SM2_P256 (Part 3 and Part 5 of GB/T 32918) (D = 64, C = 32). All other values reserved.
4	AlgExternal	4 * <code>ExtAlgCount2'</code>	If present: shall be a Requester-supported, Responder-selected, extended DHE algorithm. Table 27 — Extended Algorithm field format describes the format of this field.

307 **Table 24 — AEAD structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	Shall be <code>0x03</code> = AEAD
1	AlgCount	1	<ul style="list-style-type: none"> Bit [7:4]. Shall be a value of 2. Bit [3:0]. Shall be the number of Requester-supported, Responder-selected, extended AEAD algorithms (= <code>ExtAlgCount3'</code>). This value shall be either 0 or 1.
2	AlgSupported	2	<p>Shall be the bit mask for indicating a Requester-supported, Responder-selected, SPDM-enumerated AEAD algorithm.</p> <ul style="list-style-type: none"> Byte 0 Bit 0. AES-128-GCM. Byte 0 Bit 1. AES-256-GCM. Byte 0 Bit 2. CHACHA20_POLY1305. Byte 0 Bit 3. AEAD_SM4_GCM. All other values reserved.
4	AlgExternal	4 * <code>ExtAlgCount3'</code>	If present: shall be a Requester-supported, Responder-selected, extended AEAD algorithm. Table 27 — Extended Algorithm field format describes the format of this field.

308 **Table 25 — ReqBaseAsymAlg structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	Shall be 0x04 = ReqBaseAsymAlg
1	AlgCount	1	<ul style="list-style-type: none"> Bit [7:4]. Shall be a value of 2. Bit [3:0]. Number of Requester-supported, Responder-selected, extended asymmetric key signature algorithms (= ExtAlgCount4') for the purpose of signature verification. This value shall be either 0 or 1.
2	AlgSupported	2	<p>Shall be the bit mask for indicating a Requester-supported, Responder-selected, SPDM-enumerated asymmetric key signature algorithm for the purpose of signature verification. If the Responder does not support any request/response pair that requires signature verification, this value shall be set to zero. If the Responder will not send any messages that require a signature, this value should be set to zero.</p> <ul style="list-style-type: none"> 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. Byte 1 Bit 1. TPM_ALG_SM2_ECC_SM2_P256. Byte 1 Bit 2. EdDSA ed25519. Byte 1 Bit 3. EdDSA ed448. All other values reserved. <p>For details of SigLen for each algorithm, see Table 15 — NEGOTIATE_ALGORITHMS request message format.</p>
4	AlgExternal	4 * ExtAlgCount4'	If present: shall be a Requester-supported, Responder-selected extended asymmetric key signature algorithm for the purpose of signature verification. Table 27 — Extended Algorithm field format describes the format of this field.

309 **Table 26 — KeySchedule structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	Shall be 0x05 = KeySchedule
1	AlgCount	1	<ul style="list-style-type: none"> Bit [7:4]. Shall be a value of 2. Bit [3:0]. Shall be the number of Requester-supported, Responder-selected, extended key schedule algorithms (= ExtAlgCount5'). This value shall be either 0 or 1.

Byte offset	Field	Size (bytes)	Description
2	AlgSupported	2	Shall be the bit mask for indicating a Requester-supported, Responder-selected, SPDM-enumerated key schedule algorithm. <ul style="list-style-type: none"> Byte 0 Bit 0. SPDM key schedule. All other values reserved.
4	AlgExternal	4 * ExtAlgCount5'	If present: shall be a Requester-supported, Responder-selected, extended key schedule algorithm. Table 27 — Extended Algorithm field format describes the format of this field.

310 **Table 27 — Extended Algorithm field format**

Byte offset	Field	Size (bytes)	Description
0	Registry ID	1	Shall represent the registry or standards body. The ID column of Table 60 — Registry or standards body ID describes the value of this field.
1	Reserved	1	Reserved.
2	Algorithm ID	2	Shall indicate the desired algorithm. The registry or standards body owns the value of this field. See Table 60 — Registry or standards body ID .

311 **Table 28 — Opaque Data Format Support and Selection**

Bit offset	Field	Description
0	OpaqueDataFmt0	If set, this bit shall indicate that the format for all <code>OpaqueData</code> fields in this specification is defined by the device vendor or other standards body.
1	OpaqueDataFmt1	If set, this bit shall indicate that the format for all <code>OpaqueData</code> fields in this specification is defined by the General opaque data format .
[3:2]	Reserved	Reserved.

312 The Opaque Data Format Selection Table shows the bit definition for the format of the Opaque data fields. A Requester may set more than one bit in the table to indicate each supported format. A Responder shall select no more than one of the bits supported by the Requester in this table. If the Requester or the Responder does not set a bit, then all `OpaqueData` fields in this specification shall be absent by setting the respective `OpaqueDataLength` field to a value of zero.

313 **Table 29 — Measurement Specification Field Format**

Bit offset	Field	Description
0	DMTFmeasSpec	This bit shall indicate a DMTF-defined measurement specification. Table 54 — DMTF measurement specification format defines the format for this measurement specification.
[1:7]	Reserved	Reserved

314 The Measurement Specification Field Format Table describes the field format for Measurement specification related fields. The selected measurement specification (`MeasurementSpecificationSel`) is used in the `MEASUREMENTS` response. See [Measurement block](#) and `GET_MEASUREMENTS` for details.

315 **Table 30 — Measurement Extension Log Specification Field Format**

Bit offset	Field	Description
0	DMTFmelSpec	This bit indicates a DMTF-defined measurement extension log specification. Refer to the DMTF Measurement Extension Log Format clause for details. If the Responder supports the DMTF-defined measurement extension log specification, it shall set this bit to 1 in <code>MELspecification</code> . If the Responder selects the DMTF-defined measurement extension log specification for constructing the MEL, it shall set this bit to 1 in <code>MELspecificationSel</code> .
[1:7]	Reserved	Reserved

316 The Measurement Extension Log Specification Field Format Table describes the field format for MEL specification related fields. The selected MEL specification (`MELspecificationSel`) is used in construction of the MEL.

317 10.4.1 Connection behavior after VCA

318 With the successful completion of the `ALGORITHMS` message, all the parameters of the SPDM connection have been determined. Thus, all SPDM message exchanges after the VCA messages shall comply with the selected parameters in the `ALGORITHMS` message, with the exception of `GET_VERSION` and `VERSION` messages, or unless otherwise stated in this specification. To explain this behavior, suppose a Responder supports both RSA and ECDSA asymmetric algorithms. For an SPDM connection, the Responder selects the `TPM_ALG_RSASSA_2048` asymmetric algorithm in `BaseAsymSel` and the `TPM_ALG_SHA_256` hash algorithm in `BaseHashSel` . If the Requester on that same connection issues `GET_DIGESTS` , the Responder returns `TPM_ALG_SHA_256` digests only for those populated slots that can provide a `TPM_ALG_RSASSA_2048` signature for a `CHALLENGE_AUTH` response. The Responder would violate this requirement if it returns one or more digests of populated slots that perform ECDSA signatures or if it uses a different hash algorithm to create the digests.

319 Unless otherwise stated in this specification, and with the exception of `GET_VERSION` , if a Requester issues a request that violates one or more of the negotiated or selected parameters of a given connection, the Responder shall either silently discard the request or return an `ERROR` message with an appropriate error code.

320 10.4.2 Multiple asymmetric key negotiation

321 The Requester and Responder can negotiate the parameters of multiple asymmetric key support for the SPDM connection. As with other parameters in this request and response, the Responder makes the selection and the Requester indicates its support. There are two sets of multiple asymmetric key use parameters to negotiate: one for Responder authentication and one for Requester authentication.

322 10.4.3 Multiple asymmetric key use for Responder authentication

323 The Responder shall report the multiple asymmetric keys capability in the `MULTI_KEY_CAP` field of `CAPABILITIES`.

324 If `MULTI_KEY_CAP` is `10b`, the `ResponderMultiKeyConn` field in `NEGOTIATE_ALGORITHMS` determines whether or not the SPDM connection uses multiple asymmetric keys for Responder authentication. The Requester makes the decision for the SPDM connection in the `ResponderMultiKeyConn` field. If the Requester sets the `ResponderMultiKeyConn` field, the Responder shall support multiple asymmetric keys in the SPDM connection for Responder authentication. If `ResponderMultiKeyConn` is not set, the Responder shall support only one key pair per supported asymmetric algorithm for this SPDM connection.

325 If `MULTI_KEY_CAP` is `01b`, the Responder determines that the SPDM connection uses multiple asymmetric keys. The `ResponderMultiKeyConn` field in `NEGOTIATE_ALGORITHMS` shall be set to acknowledge the Responder capability.

326 If `MULTI_KEY_CAP` is `00b`, the Responder determines that the SPDM connection does not use multiple asymmetric keys. The `ResponderMultiKeyConn` field in `NEGOTIATE_ALGORITHMS` shall be cleared.

327 10.4.4 Multiple asymmetric key use for Requester authentication

328 The Requester shall report the multiple asymmetric keys capability for Requester authentication in the `MULTI_KEY_CAP` field of `GET_CAPABILITIES`.

329 If `MULTI_KEY_CAP` is `10b`, the `RequesterMultiKeyConnSel` field in the `ALGORITHMS` message determines whether or not the SPDM connection uses multiple asymmetric keys for Requester authentication, such as in mutual authentication. The Responder makes the decision for the SPDM connection in `RequesterMultiKeyConnSel`. If the Responder sets the `RequesterMultiKeyConnSel` field, the Requester shall support multiple asymmetric keys in this SPDM connection for Requester authentication. If `RequesterMultiKeyConnSel` is not set, the Requester shall support only one key pair per supported asymmetric algorithm for this SPDM connection.

330 If `MULTI_KEY_CAP` is `01b`, the Requester determines that the SPDM connection uses multiple asymmetric keys. The `RequesterMultiKeyConnSel` field in the `ALGORITHMS` message shall be set to acknowledge the Requester capability.

331 If `MULTI_KEY_CAP` is `00b`, the Requester determines that the SPDM connection does not use multiple asymmetric keys. The `RequesterMultiKeyConnSel` field in the `ALGORITHMS` message shall be cleared.

10.4.5 Multiple asymmetric key connection

For the remainder of this specification, the boolean variables `MULTI_KEY_CONN_REQ` and `MULTI_KEY_CONN_RSP` indicate whether or not the responding SPDM endpoint supports more than one key pair for one or more asymmetric algorithms for key pairs belonging to it in this SPDM connection. If the responding endpoint is the Requester, then `MULTI_KEY_CONN_REQ` is used. See [Table 31 — MULTI_KEY_CONN_REQ value calculation](#). If the responding endpoint is the Responder, then `MULTI_KEY_CONN_RSP` is used. See [Table 32 — MULTI_KEY_CONN_RSP value calculation](#).

Table 31 — MULTI_KEY_CONN_REQ value calculation

MULTI_KEY_CAP in GET_CAPABILITIES	RequesterMultiKeyConnSel in ALGORITHMS	MULTI_KEY_CONN_REQ
00b	0	false
00b	1	invalid
01b	0	invalid
01b	1	true
10b	0	false
10b	1	true

Table 32 — MULTI_KEY_CONN_RSP value calculation

MULTI_KEY_CAP in CAPABILITIES	ResponderMultiKeyConn in NEGOTIATE_ALGORITHMS	MULTI_KEY_CONN_RSP
00b	0	false
00b	1	invalid
01b	0	invalid
01b	1	true
10b	0	false
10b	1	true

If the responding SPDM endpoint has `MULTI_KEY_CAP` set to 00b, then the corresponding `MULTI_KEY_CONN_REQ` or `MULTI_KEY_CONN_RSP` shall be false.

If the responding SPDM endpoint has `MULTI_KEY_CAP` set to 01b, then the corresponding `MULTI_KEY_CONN_REQ` or `MULTI_KEY_CONN_RSP` shall be true.

If the responding SPDM endpoint has `MULTI_KEY_CAP` set to 10b, then the value of the corresponding

`MULTI_KEY_CONN_REQ` or `MULTI_KEY_CONN_RSP` depends on the peer endpoint. If the responding SPDM endpoint is the Requester and if `RequesterMultiKeyConnSel` is set by the Responder, then the value of `MULTI_KEY_CONN_REQ` shall be true. If the responding SPDM endpoint is the Responder and if `ResponderMultiKeyConn` is set by the Requester, then the value of `MULTI_KEY_CONN_RSP` shall be true. In all other cases, the value of the corresponding `MULTI_KEY_CONN_REQ` or `MULTI_KEY_CONN_RSP` shall be false.

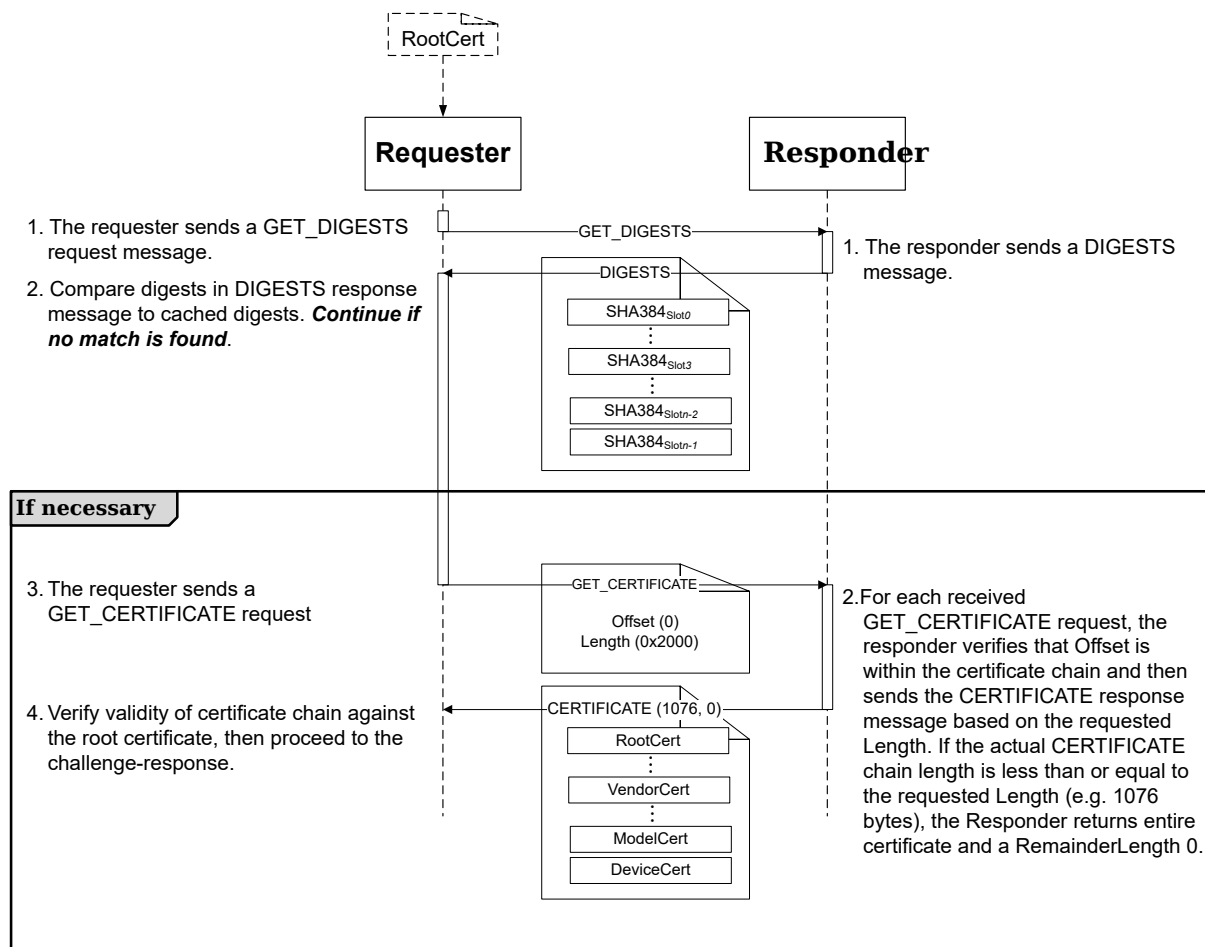
339 10.5 Responder identity authentication

340 This clause describes request messages and response messages associated with the identity of the Responder's authentication operations. The `GET_DIGESTS` and `GET_CERTIFICATE` messages shall be supported by a Responder that returns `CERT_CAP=1` in its `CAPABILITIES` response message. The `CHALLENGE` message that this clause defines shall be supported by a Responder that returns `CHAL_CAP=1` in its `CAPABILITIES` response message. The `GET_DIGESTS` and `GET_CERTIFICATE` messages are not applicable if the public key of the Responder was provisioned to the Requester in a trusted environment.

341 [Figure 8 — Responder authentication: Example certificate retrieval flow](#) shows the high-level request-response message flow and sequence for [certificate](#) retrieval.

342 **Figure 8 — Responder authentication: Example certificate retrieval flow**

343



344 The GET_DIGESTS request message and DIGESTS response message can 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 every certificate chain stored on an endpoint are returned with the DIGESTS response message, enabling the Requester to compare these values to previously retrieved and cached certificate chain hash values and detect any changes to the certificate chains stored on the device before issuing a GET_CERTIFICATE request message.

345 For the runtime challenge-response flow, the signature field in the CHALLENGE_AUTH response message payload shall be signed by using the private key associated with the leaf certificate over the hash of the message transcript. See [Table 47 — Request ordering and message transcript computation rules for M1/M2](#).

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

347 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 message transcript generation for the signature computation is restarted as of the most recent `GET_VERSION` request received.

348 10.6 Requester identity authentication

349 If a Requester supports mutual authentication, it shall comply with all requirements placed on a Responder as specified in [Responder identity authentication](#).

350 If a Responder supports mutual authentication, it shall comply with all requirements placed on a Requester as specified in [Responder identity authentication](#). The preceding two statements essentially describe a role reversal.

351 10.6.1 Certificates and certificate chains

352 Each SPDM endpoint that supports identity authentication using certificates shall carry at least one complete certificate chain. A certificate chain contains an ordered list of certificates, presented as the binary (byte) concatenation of the fields that [Table 33 — Certificate chain format](#) shows. In the context of this specification, a complete certificate chain is one where: (i) the first certificate either is signed by a Root Certificate (a certificate that specifies a trust anchor) or is a Root Certificate itself, (ii) each subsequent certificate is signed by the preceding certificate, and (iii) the final certificate contains the public key used to authenticate the SPDM endpoint. The final certificate is called the [leaf certificate](#).

353 If an SPDM endpoint does not support multiple asymmetric keys (`MULTI_KEY_CAP=0`), the SPDM endpoint shall contain a single public-private key pair per supported algorithm for its leaf certificates, regardless of how many certificate chains are stored on the device. The Responder selects a single asymmetric key signature algorithm per Requester regardless of the value of `MULTI_KEY_CAP` field.

354 Certificate chains are stored in logical locations called *slots*. Each supported slot shall either be empty or contain one complete certificate chain. A device shall not contain more than eight slots. Slots are numbered 0 through 7 inclusive. Slot 0 is populated by default. If a device uses the `DeviceCert` model (`ALIAS_CERT_CAP=0b` in its `CAPABILITIES` response) and if `MULTI_KEY_CAP` is not set, then the certificate chain in every populated slot shall use the `DeviceCert` model. If a device uses the `AliasCert` model (`ALIAS_CERT_CAP=1b` in its `CAPABILITIES` response) and if `MULTI_KEY_CAP` is not set, then the certificate chain in every populated slot shall use the `AliasCert` model.

355 If the `MULTI_KEY_CAP` is set, the certificate model for each populated certificate slot can be different. Multiple asymmetric key support allows the use of the generic certificate model. The use of the `GenericCert` model shall be prohibited when `MULTI_KEY_CAP` is not set.

356 In all cases, the certificate model for slot 0 shall be either the device certificate model or the alias certificate model.

357 Additional slots may be populated through the supply chain such as by a platform integrator or by an end user such as an IT administrator. A slot mask identifies the certificate chains in the eight slots. Similarly, if the Requester supports mutual authentication and if `MULTI_KEY_CONN_REQ` is not set, a Requester device shall use either the

`DeviceCert` model or the `AliasCert` model and the certificate chain in every populated slot shall use the same model. Note that the Requester does not have capability flags to indicate the certificate model.

358 If an endpoint supports certificates, then Slot 0 is the default certificate chain slot. Slot 0 shall contain a valid certificate chain unless the device has not yet had a certificate chain provisioned and is in a trusted environment.

359 Each certificate in a chain shall be in ASN.1 DER-encoded [X.509](#) v3 format as [RFC 5280](#) defines. The ASN.1 DER encoding of each individual certificate can be analyzed to determine its length.

360 To allow for flexibility in supporting multiple certificate models, the minimum number of certificates within a certificate chain shall be one and a chain shall contain a leaf certificate.

361 The leaf certificate in the device certificate model shall be the `DeviceCert` leaf certificate. The leaf certificate in an alias certificate model shall be the `AliasCert` leaf certificate. In a generic certificate model, the leaf certificate shall be the `GenericCert` leaf certificate. When `MULTI_KEY_CAP` is not set and a certificate chain consists of a single certificate, that certificate can only be a `DeviceCert` leaf certificate. When `MULTI_KEY_CAP` is set and a certificate chain consists of a single certificate, that certificate is either a `DeviceCert` or a `GenericCert` leaf certificate.

362 [Table 33 — Certificate chain format](#) describes the certificate chain format:

363 **Table 33 — Certificate chain format**

Byte offset	Field	Size (bytes)	Description
0	Length	2	Shall be the 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	Shall be the digest of the Root Certificate. Note that the Root Certificate is ASN.1 DER-encoded for this digest. This field shall be in hash byte order . H is the output size, in bytes, of the hash algorithm selected by the most recent <code>ALGORITHMS</code> response.
4 + H	Certificates	Length - (4 + H)	Shall be a complete certificate chain consisting of one or more ASN.1 DER-encoded X.509 v3 certificates. This field shall be in Encoded ASN.1 byte order .

364 10.7 GET_DIGESTS request and DIGESTS response messages

365 This request message shall retrieve the certificate chain digests.

366 [Table 34 — GET_DIGESTS request message format](#) shows the `GET_DIGESTS` request message format.

367 The digests in [Table 35 — Successful DIGESTS response message format](#) shall be computed over the certificate chain as [Table 33 — Certificate chain format](#) shows.

368 When the corresponding `MULTI_KEY_CONN_REQ` or `MULTI_KEY_CONN_RSP` is true, certificate slots have four states that can be reported by the endpoint. The sub-bullet of each state describes how the state is represented in the `DIGESTS` response.

1. Does not exist
 - The corresponding bit in `SupportedSlotMask` is not set.
2. Exists and empty
 - The corresponding bit in `SupportedSlotMask` is set and the corresponding bit in `ProvisionedSlotMask` is not set.
3. Exists with key
 - The corresponding bits in `SupportedSlotMask` and `ProvisionedSlotMask` are set, but the value of the corresponding `Digest` field is zero.
4. Exists with key and cert
 - The corresponding bits in `SupportedSlotMask` and `ProvisionedSlotMask` are set, and the value of the corresponding `Digest` field is non-zero.

369 When a certificate slot does not exist, it shall remain in this state for the remainder of the SPDM connection. The "exists and empty" state indicates the presence of a certificate slot where neither a key nor a certificate has been provisioned yet. The "exists with key" state indicates the certificate slot has only an asymmetric key associated with it but no certificate chain. The "exists with key and cert" state indicates the certificate has both an asymmetric key assigned to it and a certificate chain. The "exists with key and cert" state is a fully provisioned state. When a certificate slot exists, the typical progression of states starts at "exists and empty", followed by "exists with key", and ends with "exists with key and cert".

370 **Table 34 — GET_DIGESTS request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x81</code> = <code>GET_DIGESTS</code> . See Table 4 — SPDM request codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.

371 **Table 35 — Successful DIGESTS response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	<code>0x01</code> = <code>DIGESTS</code> . See Table 5 — SPDM response codes .

Byte offset	Field	Size (bytes)	Description
2	Param1	1	SupportedSlotMask. This field indicates which slots the responding SPDM endpoint supports. If certificate slot X exists in the responding SPDM endpoint, the bit in position X of this byte shall be set. (Bit 0 is the least significant bit of the byte.) Likewise, if certificate slot X does not exist in the responding SPDM endpoint, bit X of this byte shall not be set and certificate slot X shall be an invalid value in various slot ID fields (SlotID) across all SPDM request messages that contain this field.
3	Param2	1	ProvisionedSlotMask. If slot K contains a certificate chain that supports the currently negotiated algorithms for the connection, bit K of this byte shall be set. (Bit 0 is the least significant bit of the byte.) Additionally, if the corresponding MULTI_KEY_CONN_REQ or MULTI_KEY_CONN_RSP is true and if slot K contains an associated key pair, bit K of this byte shall be set. For all fields from Digest to KeyUsageMask inclusive, the number of fields returned (denoted by n) shall be equal to the number of bits set in this byte. These fields shall be returned in order of increasing slot number. If a bit is set in this field, the corresponding bit in SupportedSlotMask shall also be set.
4	Digest[0]	H	Digest of the certificate chain in CertSlot[0] . This field shall be in hash byte order .
...
4 + H * (n - 1)	Digest[n-1]	H	Digest of the certificate chain in CertSlot[n-1] . This field shall be in hash byte order . If a certificate chain is not present in this slot, the value of this field shall be all zeros.
4 + (H * n)	KeyPairID[0]	1	Shall be the KeyPairID of the key pair associated with CertSlot[0] . This field shall be present if the corresponding MULTI_KEY_CONN_REQ or MULTI_KEY_CONN_RSP is true. Otherwise, it shall be absent.
...
3 + (H + 1) * n	KeyPairID[n-1]	1	Shall be the KeyPairID of the key pair associated with CertSlot[n-1] . This field shall be present if the corresponding MULTI_KEY_CONN_REQ or MULTI_KEY_CONN_RSP is true. Otherwise, it shall be absent.
4 + (H + 1) * n	CertificateInfo[0]	1	Shall be the certificate information for CertSlot[0] . The format of this field shall be the format that the certificate info table defines. This field shall be present if the corresponding MULTI_KEY_CONN_REQ or MULTI_KEY_CONN_RSP is true. Otherwise, it shall be absent.
...

Byte offset	Field	Size (bytes)	Description
3 + (H + 2) * n	CertificateInfo[n-1]	1	<p>Shall be the certificate information for <code>CertSlot[n-1]</code> . The format of this field shall be the format that the certificate info table defines.</p> <p>This field shall be present if the corresponding <code>MULTI_KEY_CONN_REQ</code> or <code>MULTI_KEY_CONN_RSP</code> is true. Otherwise, it shall be absent.</p>
4 + (H + 2) * n	KeyUsageMask[0]	2	<p>Shall be the key usage bit mask for <code>CertSlot[0]</code> . The format of this field shall be the format that the key usage bit mask table defines.</p> <p>This field shall be present if the corresponding <code>MULTI_KEY_CONN_REQ</code> or <code>MULTI_KEY_CONN_RSP</code> is true. Otherwise, it shall be absent.</p>
...
2 + (H + 4) * n	KeyUsageMask[n-1]	2	<p>Shall be the key usage bit mask for <code>CertSlot[n-1]</code> . The format of this field shall be the format that the key usage bit mask table defines.</p> <p>This field shall be present if the corresponding <code>MULTI_KEY_CONN_REQ</code> or <code>MULTI_KEY_CONN_RSP</code> is true. Otherwise, it shall be absent.</p>

372 [Table 36 — Certificate info](#) shows the format for the `CertificateInfo` fields.

373 **Table 36 — Certificate info**

Bit offset	Field	Description
[2:0]	CertModel	<p>The value of this field shall indicate the certificate model that the certificate slot uses.</p> <ul style="list-style-type: none"> Value of 0 indicates either that the certificate slot does not contain any certificates or that the corresponding <code>MULTI_KEY_CONN_REQ</code> or <code>MULTI_KEY_CONN_RSP</code> is false. Value of 1 indicates that the certificate slot uses the <code>DeviceCert</code> model. Value of 2 indicates that the certificate slot uses the <code>AliasCert</code> model. Value of 3 indicates that the certificate slot uses the <code>GenericCert</code> model. All other values reserved.
[7:3]	Reserved	Reserved

374 [Table 37 — Key usage bit mask](#) shows the format for the `KeyUsageMask` fields.

375 **Table 37 — Key usage bit mask**

Bit offset	Field	Description
0	KeyExchangeUse	If set, the SlotID fields in KEY_EXCHANGE and KEY_EXCHANGE_RSP can specify this certificate slot. If not set, the SlotID fields in KEY_EXCHANGE and KEY_EXCHANGE_RSP shall not specify this certificate slot.
1	ChallengeUse	If set, the SlotID fields in CHALLENGE and CHALLENGE_AUTH can specify this certificate slot. If not set, the SlotID fields in CHALLENGE and CHALLENGE_AUTH shall not specify this certificate slot.
2	MeasurementUse	If set, the SlotID fields in GET_MEASUREMENTS and MEASUREMENTS can specify this certificate slot. If not set, the SlotID fields in GET_MEASUREMENTS and MEASUREMENTS shall not specify this certificate slot.
3	EndpointInfoUse	If set, the SlotID fields in GET_ENDPOINT_INFO and ENDPOINT_INFO can specify this certificate slot. If not set, the SlotID fields in GET_ENDPOINT_INFO and ENDPOINT_INFO shall not specify this certificate slot.
[13:4]	Reserved	Reserved
14	StandardsKeyUse	If set, this field shall indicate usage defined by standards other than specifications defined by DMTF.
15	VendorKeyUse	If set, this field shall indicate usage defined by a vendor.

For slot 0, at least one of KeyExchangeUse, ChallengeUse, MeasurementUse, and EndpointInfoUse shall be set. The corresponding capability bits shall be set appropriately.

10.8 GET_CERTIFICATE request and CERTIFICATE response messages

This request message shall retrieve the certificate chain from the specified slot number.

Table 38 — GET_CERTIFICATE request message format shows the GET_CERTIFICATE request message format.

GET_CERTIFICATE request attributes shows the GET_CERTIFICATE request attributes.

Table 40 — Successful CERTIFICATE response message format shows the CERTIFICATE response message format.

Table 141 — CERTIFICATE response attributes shows the CERTIFICATE response attributes.

The Requester sends one or more GET_CERTIFICATE requests to retrieve the certificate chain of the Responder.

Table 38 — GET_CERTIFICATE request message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in SPDM version .
1	RequestResponseCode	1	Shall be 0x82 = GET_CERTIFICATE. See Table 4 — SPDM request codes .

Byte offset	Field	Size (bytes)	Description
2	Param1	1	Bit [7:4]. Reserved. Bit [3:0]. Shall be the <code>SlotID</code> . Slot number of the Responder certificate chain to read. The value in this field shall be between 0 and 7 inclusive.
3	Param2	1	Request attributes. See GET_CERTIFICATE request attributes .
4	Offset	2	Shall be the offset in bytes from the start of the certificate chain to where the read request message begins. The Responder shall send its certificate chain starting from this offset. For the first <code>GET_CERTIFICATE</code> request for a given slot, the Requester shall set this field to 0. For subsequent requests, <code>Offset</code> is set to the next portion of the certificate in that slot.
6	Length	2	Shall be the length of certificate chain data, in bytes, to be returned in the corresponding response. This field is an unsigned 16-bit integer.

385 **Table 39 — GET_CERTIFICATE request attributes**

Bit offset	Field	Description
0	SlotSizeRequested	When <code>SlotSizeRequested=1b</code> in the <code>GET_CERTIFICATE</code> request, the Responder shall return the number of bytes available for certificate chain storage in the <code>RemainderLength</code> field of the response. When <code>SlotSizeRequested=1b</code> , the <code>Offset</code> and <code>Length</code> fields in the <code>GET_CERTIFICATE</code> request shall be ignored by the Responder.
[7:1]	Reserved	Reserved.

386 **Table 40 — Successful CERTIFICATE response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x02 = CERTIFICATE</code> . See Table 5 — SPDM response codes .
2	Param1	1	Bit [7:4]. Reserved. Bit [3:0]. Shall be the <code>SlotID</code> . Slot number of the certificate chain returned.
3	Param2	1	The format of this field shall be the format that Table 141 — CERTIFICATE response attributes defines.

Byte offset	Field	Size (bytes)	Description
4	PortionLength	2	Shall be the number of bytes of this portion of the certificate chain. This should be less than or equal to <code>Length</code> received as part of the request. For example, the Responder might set this field to a value less than <code>Length</code> received as part of the request due to limitations on the transmit buffer of the Responder. If the requested <code>Length</code> field is <code>0</code> then this field shall be set to 0. If <code>SlotSizeRequested=1b</code> in the request, this field shall be set to zero.
6	RemainderLength	2	Shall be the 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. If the requested <code>Length</code> field is <code>0</code> and <code>SlotSizeRequested=0b</code> in the request, then this field shall return the actual size of the certificate chain in the slot. See Table 39 — GET_CERTIFICATE request attributes for more detail.
8	CertChain	<code>PortionLength</code> or 0	Shall be the requested contents of the target certificate chain, as described in Certificates and certificate chains . If <code>SlotSizeRequested=1b</code> in the request, this field shall be absent. If the requested <code>Length</code> field is <code>0</code> , then this field shall be absent.

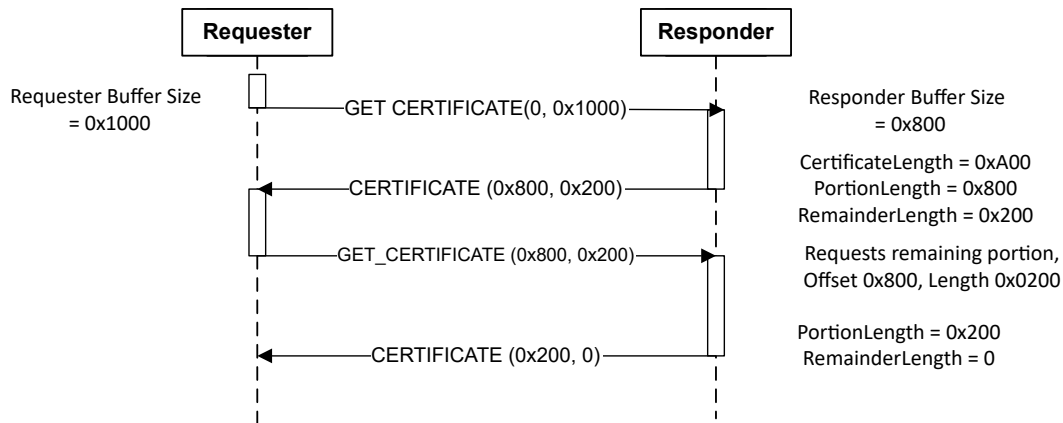
387 **Table 41 — CERTIFICATE response attributes**

Bit offset	Field	Description
[2:0]	CertificateInfo	The value of this field shall be the certificate model of the slot. The format of this field shall be the format of the <code>CertModel</code> field that the certificate info table defines.
All other bits	Reserved	Reserved.

388 [Figure 9 — Responder cannot return full length data flow](#) shows the high-level request-response message flow when the Responder cannot return the entire data requested by the Requester in the first response.

389 **Figure 9 — Responder cannot return full length data flow**

390



391 Endpoints that support the [large SPDM message transfer mechanism](#) message set shall use the large SPDM message transfer mechanism messages to manage the transfer of the requested certificate chain when the `CERTIFICATE` response is larger than either the `DataTransferSize` of the Requester or the transmit buffer of the Responder. Specifically:

- If the size of the `CERTIFICATE` response is greater than `DataTransferSize` and less than or equal to the `MaxSPDMmsgSize` of the Requester or if the response is greater than the transmit buffer of the Responder, then the Responder shall reply with an `ERROR` message of `ErrorCode=LargeResponse`.
- If the Requester sets `Offset` to 0 and `Length` to 0xFFFF in the `GET_CERTIFICATE` request, the Responder shall set `PortionLength` equal to the size of the complete certificate chain stored in the requested slot, shall set `RemainderLength` to 0, and shall store the contents of the complete certificate chain in `CertChain` in the `CERTIFICATE` response. Then the Responder shall fragment and return this response message in chunks, as per the clauses presented in [CHUNK_GET request and CHUNK_RESPONSE response message](#).

392 By setting `SlotSizeRequested=1b` in the request attributes, the Requester can query the size of the Responder's certificate slot. The Requester should query the slot size before any action that uses slot storage, because the Responder might change the value of the slot size based on other actions.

393 10.8.1 Mutual authentication requirements for `GET_CERTIFICATE` and `CERTIFICATE` messages

394 If the Requester supports mutual authentication, the requirements placed on the Responder in [GET_CERTIFICATE request and CERTIFICATE response messages](#) clause shall also apply to the Requester. If the Responder supports mutual authentication, the requirements placed on the Requester in the [GET_CERTIFICATE request and CERTIFICATE response messages](#) clauses shall also apply to the Responder. The preceding two sentences essentially describe a role reversal.

10.8.2 SPDM certificate requirements and recommendations

This specification defines a number of X.509 v3 [required](#) and [optional](#) fields for compliant SPDM certificates. SPDM certificates also adhere to the requirements as [RFC 5280](#) defines. Unless stated otherwise, the following clauses apply to those certificates in the chain that are specific to a device instance, that is, the leaf certificate in the `DeviceCert` model or the `DeviceCert`, all intermediate `AliasCert`s, and the leaf certificate in the `AliasCert` model. See [identity provisioning](#).

In addition, the `Subject Alternative Name` certificate extension `otherName` field is recommended for providing device information. See the [Definition of otherName using the DMTF OID](#).

In [Table 42 — Field requirements](#), the requirements columns define the requirement for the corresponding certificate models. In these columns, the corresponding field with a value of "Mandatory" shall be present in the leaf certificate. Likewise, the corresponding field with a value of "Optional" can be present or absent in the leaf certificate. As a note, this table reflects the minimum requirements from the perspective of this specification. The vendor, users of the SPDM endpoint, and other standards such as [RFC 5280](#) can place additional or more-restrictive requirements.

Table 42 — Field requirements

Field	DeviceCert / AliasCert Requirements	GenericCert Requirements	Description
Basic Constraints	Mandatory	Mandatory	If present in the leaf certificate, the CA value shall be <code>FALSE</code> . The CA value shall be present and set to <code>TRUE</code> for intermediate and root certificates.
Version	Mandatory	Mandatory	If present, the version of the encoded certificate shall be present and shall be <code>3</code> (encoded as value <code>2</code>).
Serial Number	Mandatory	Mandatory	If present, the CA-assigned serial number shall be present with a positive integer value.
Signature Algorithm	Mandatory	Optional	If present, the <code>Signature</code> algorithm that the CA uses shall be present.
Issuer	Mandatory	Optional	If present, the CA distinguished name shall be specified.
Subject Name	Mandatory	Optional	If present, the subject name shall be present and shall represent the distinguished name associated with the leaf certificate.
Validity	Mandatory	Optional	If present, see Certificate validity details , and RFC 5280 .
Subject Public Key Info	Mandatory	Mandatory	If present, the device public key and the algorithm shall be present.
Key Usage	Mandatory	Optional	If present, the key usage bit for digital signature shall be set.

Table 43 — Optional fields

Field	Description
Subject Alternative Name otherName	In some cases, it might be desirable to provide device-specific information as part of the leaf 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 leaf certificate. See the Definition of otherName using the DMTF OID . Note that otherName field formats specified by other standards are permissible in the certificate.
Extended Key Usage (EKU)	If present in a certificate, the Extended Key Usage extension indicates one or more purposes for which the public key should be used. See Extended Key Usage authentication OIDs .
SPDM Non-critical Certificate Extension	If present in a certificate, the SPDM Non-critical Certificate Extension indicates one or more non-critical OIDs associated with the certificate. See SPDM Non-Critical Certificate Extension OID .

401 Certificate validity details

402 As per [RFC 5280](#), the certificate validity period is the time interval during which the CA warrants that it will maintain information about the status of the certificate. The field is represented as an ASN.1-encoded SEQUENCE of two dates: the date when the certificate validity period begins (notBefore) and the date when the certificate validity period ends (notAfter).

403 For a leaf certificate whose chain is stored in Slot 0, the notBefore date should be the date of certificate creation, and the notAfter date should be set to GeneralizedTime value 99991231235959Z . Immutable leaf certificates' notAfter dates should be set appropriately to ensure that the leaf certificate will not expire during the practical lifetime of the device.

404 For leaf certificates whose chains are stored in Slots 1-7, the notBefore date should be the date of certificate creation. The notAfter date can be set according to end user requirements, including values that will result in certificate expiration and thus require certificate renewal and device recertification during the lifetime of the device.

405 [Definition of otherName using the DMTF OID](#) shows the definition of otherName using the DMTF OID:

406 Definition of otherName using the DMTF OID

```
id-DMTF OBJECT IDENTIFIER ::= { 1 3 6 1 4 1 412 }

id-DMTF-spdn OBJECT IDENTIFIER ::= { id-DMTF 274 }

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})

```

407 The [Leaf certificate example](#) shows an example leaf certificate.

408 10.8.2.1 Extended Key Usage authentication OIDs

409 The following Extended Key Usage purposes are defined for SPDM certificate authentication:

- SPDM Responder Authentication { id-DMTF-spdm 3 }: The presence of this OID shall indicate that a leaf certificate can be used for Responder authentication purposes.
- SPDM Requester Authentication { id-DMTF-spdm 4 }: The presence of this OID shall indicate that a leaf certificate can be used for Requester authentication purposes.

410 The presence of both OIDs shall indicate that the leaf certificate can be used for both Requester and Responder authentication purposes. If present, these OIDs shall appear in the leaf certificate.

411 A Responder device that supports mutual authentication should include the `SPDM Responder Authentication` OID in the Extended Key Usage field of its leaf certificate. A Requester device that supports mutual authentication should include the `SPDM Requester Authentication` OID in the Extended Key Usage field of its leaf certificate. Note that alternate OIDs specified by other standards are permissible in the certificate.

412 10.8.2.2 SPDM Non-Critical Certificate Extension OID

413 The `id-DMTF-spdm-extension` OID is a container of non-critical SPDM OIDs and their corresponding values. The OID value for `id-DMTF-spdm-extension` shall be { id-DMTF-spdm 6 }. Furthermore, this OID is a Certificate Extension as defined in [RFC 5280](#), and its encoding shall follow the `Extension` syntax also defined in RFC 5280. The `Extension`

syntax defines three parameters: `extnID`, `critical`, and `extnValue`. The values of these three parameters for `id-DMTF-spdm-extension` shall be the DER encoding of the ASN.1 value as the [DMTF SPDM Extension Format](#) defines.

414 Definition of DMTF SPDM Extension Format

```
id-DMTF-spdm-extension Extension ::=
{
    extnID      { id-DMTF-spdm 6 }
    critical    FALSE
    extnValue   id-spdm-cert-oids
}

id-spdm-cert-oids ::= SEQUENCE SIZE (1..MAX) OF id-spdm-cert-oid

id-spdm-cert-oid ::= SEQUENCE
{
    spdmOID      OBJECT IDENTIFIER
    spdmOIDdefinition OCTET STRING OPTIONAL
}
```

415 The `spdmOID` field shall contain an OID defined in this specification. Only designated OIDs, permitted by this specification, shall be allowed in `spdmOID`. The `spdmOIDdefinition` field shall be a DER encoding of the ASN.1 value of the definition indicated by `spdmOID`.

416 These clauses describe the definitions and formats of the SPDM OIDs contained in `id-DMTF-spdm-extension`. If present, these OIDs shall only be contained in `id-DMTF-spdm-extension`.

417 10.8.2.2.1 Hardware identity OID

418 The `id-DMTF-hardware-identity` OID is defined to help identify the hardware identity certificate in a chain regardless of the [certificate chain model](#) used (`DeviceCert` or `AliasCert`). If the `AliasCert` model is used, this OID shall not be present in any alias certificates in the chain. The `id-DMTF-hardware-identity` OID shall have a format as [Hardware identity OID format](#) defines.

419 Hardware identity OID format

```
id-DMTF-hardware-identity id-spdm-cert-oid ::= {
    spdmOID      { id-DMTF-spdm 2 }
    spdmOIDdefinition ABSENT
}
```

420 10.8.2.2.2 Mutable certificate OID

421 Mutable certificates may include the `id-DMTF-mutable-certificate` OID to identify them as mutable. If used, this
 422 OID shall be present in all mutable certificates in the chain. The `id-DMTF-mutable-certificate` OID shall have a
 format as [Mutable certificate OID format](#) defines.

422 Mutable certificate OID format

```
id-DMTF-mutable-certificate id-spdm-cert-oid ::= {
    spdmOID          { id-DMTF-spdm 5 }
    spdmOIDdefinition ABSENT
}
```

423 10.9 CHALLENGE request and CHALLENGE_AUTH response messages

424 This request message shall authenticate a Responder through the challenge-response protocol.

425 [Table 44 — CHALLENGE request message format](#) shows the `CHALLENGE` request message format.

426 [Table 45 — Successful CHALLENGE_AUTH response message format](#) shows the `CHALLENGE_AUTH` response message
 format.

427 [Table 46 — CHALLENGE_AUTH response attribute](#) shows the `CHALLENGE_AUTH` response attribute.

428 Table 44 — CHALLENGE request message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x83 = CHALLENGE</code> . See Table 4 — SPDM request codes .
2	Param1	1	Shall be the <code>SlotID</code> . Slot number of the Responder certificate chain that shall be used for authentication. If the public key of the Responder was provisioned to the Requester in a trusted environment, the value in this field shall be <code>0xFF</code> ; otherwise it shall be between 0 and 7 inclusive.

Byte offset	Field	Size (bytes)	Description
3	Param2	1	<p>Shall be the type of measurement summary hash requested:</p> <ul style="list-style-type: none"> • 0x0 . No measurement summary hash requested. • 0x1 . TCB measurements only. • 0xFF . All measurements. • All other values reserved. <p>If a Responder does not support measurements (MEAS_CAP=00b in its CAPABILITIES response), the Requester shall set this value to 0x0 .</p>
4	Nonce	32	The Requester should choose a random value.
36	Context	8	The Requester can include application-specific information in Context. The Requester should fill this field with zeros if it has no context to provide.

429

Table 45 — Successful CHALLENGE_AUTH response message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in SPDM version .
1	RequestResponseCode	1	Shall be 0x03 = CHALLENGE_AUTH . See Table 5 — SPDM response codes .
2	Param1	1	Shall be the Response Attribute Field. See Table 46 — CHALLENGE_AUTH response attribute .
3	Param2	1	Shall be the 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.) This field is reserved if the public key of the Responder was provisioned to the Requester in a trusted environment.
4	CertChainHash	H	<p>Shall be either the hash of the certificate chain as Table 33 — Certificate chain format describes or, if the public key of the Responder was provisioned to the Requester in a trusted environment, the public key used for authentication. The Requester can use this value to check that the certificate chain or public key matches the one requested.</p> <p>This field shall be in hash byte order.</p>
4 + H	Nonce	32	Shall be the Responder-selected random value.

Byte offset	Field	Size (bytes)	Description
36 + H	MeasurementSummaryHash	MSHLength = H or 0	<p>If the Responder does not support measurements (<code>MEAS_CAP=00b</code> in its <code>CAPABILITIES</code> response) or if the requested <code>Param2 = 0x0</code> , this field shall be absent.</p> <p>If the requested <code>Param2 = 0x1</code> , this field shall be the combined hash of measurements of all measurable components considered to be in the TCB required to generate this response, computed as <code>hash(Concatenate(MeasurementBlock[0], MeasurementBlock[1], ...))</code> , where <code>MeasurementBlock[x]</code> denotes a measurement of an element in the TCB and <code>hash</code> is the negotiated base hashing algorithm. Measurements are concatenated in ascending order based on their measurement index as Table 53 — Measurement block format describes.</p> <p>If the requested <code>Param2 = 0x1</code> and if there are no measurable components in the TCB required to generate this response, this field shall be <code>0</code> .</p> <p>If the requested <code>Param2 = 0xFF</code> , this field shall be computed as <code>hash(Concatenate(MeasurementBlock[0], MeasurementBlock[1], ..., MeasurementBlock[n]))</code> of all supported measurement blocks available in the measurement index range <code>0x01 - 0xFE</code> , concatenated in ascending index order. Indices with no associated measurements shall not be included in the hash calculation. See the Measurement index assignments clause.</p> <p>If the Responder supports both raw bit stream and digest representations for a given measurement index, the Responder shall use the digest form.</p> <p>This field shall be in hash byte order.</p>
36 + H + MSHLength	OpaqueDataLength	2	Shall be the size of the <code>OpaqueData</code> field that follows in bytes. The value should not be greater than 1024 bytes. Shall be <code>0</code> if no <code>OpaqueData</code> is provided.
38 + H + MSHLength	OpaqueData	OpaqueDataLength	The Responder can include Responder-specific information and/or information that its transport defines. If present, this field shall conform to the selected opaque data format in <code>OtherParamsSelection</code> .
38 + H + MSHLength + OpaqueDataLength	RequesterContext	8	This field shall be identical to the <code>Context</code> field of the corresponding request message.

Byte offset	Field	Size (bytes)	Description
46 + H + MSHLength + OpaqueDataLength	Signature	SigLen	Shall be the Responder's signature. SigLen is the size of the asymmetric-signing algorithm output that the Responder selected in 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.

430 **Table 46 — CHALLENGE_AUTH response attribute**

Bit offset	Field	Description
[3:0]	SlotID	Shall contain the SlotID in the Param1 field of the corresponding CHALLENGE request. If the Responder's public key was provisioned to the Requester previously, this field shall be 0xF. The Requester can use this value to check that the certificate matched what was requested.
[6:4]	Reserved	Reserved.
7	DEPRECATED: BasicMutAuthReq	DEPRECATED: When mutual authentication is supported by both Responder and Requester, the Responder shall set this bit to indicate that the Responder wants to authenticate the identity of the Requester using the basic mutual authentication flow. The Requester shall not set this bit in a basic mutual authentication flow. See Basic mutual authentication flow. If mutual authentication is not supported, this bit shall be zero.

431 10.9.1 CHALLENGE_AUTH signature generation

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

- 433 1. The Responder shall construct M1, and the Requester shall construct M2 message transcripts. For Responder authentication, see the request ordering and message transcript computation rules for M1/M2 table. For Requester authentication in the mutual authentication scenario, see the Mutual authentication message transcript clause.

434 ◦ If a response contains ErrorCode=ResponseNotReady :

435 Concatenation function is performed on the contents of both the original request and the successful response received during RESPOND_IF_READY. Neither the error response (ResponseNotReady) nor the RESPOND_IF_READY request shall be included in M1/M2.

436 ◦ If a response contains an ErrorCode other than ResponseNotReady :

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

- 438 2. The Responder shall generate:

```
Signature = SPDMsign(PrivKey, M1, "challenge_auth signing");
```

439 where:

- `SPDMsign` is described in [Signature generation](#).
- `PrivKey` shall be the private key associated with the leaf certificate of the Responder in `slot=Param1` of the `CHALLENGE` request message. If the public key of the Responder was provisioned to the Requester, then `PrivKey` shall be the associated private key.

440 10.9.2 CHALLENGE_AUTH signature verification

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

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

443 1. The Requester shall perform:

```
result = SPDMsignatureVerify(PubKey, Signature, M2, "challenge_auth signing");
```

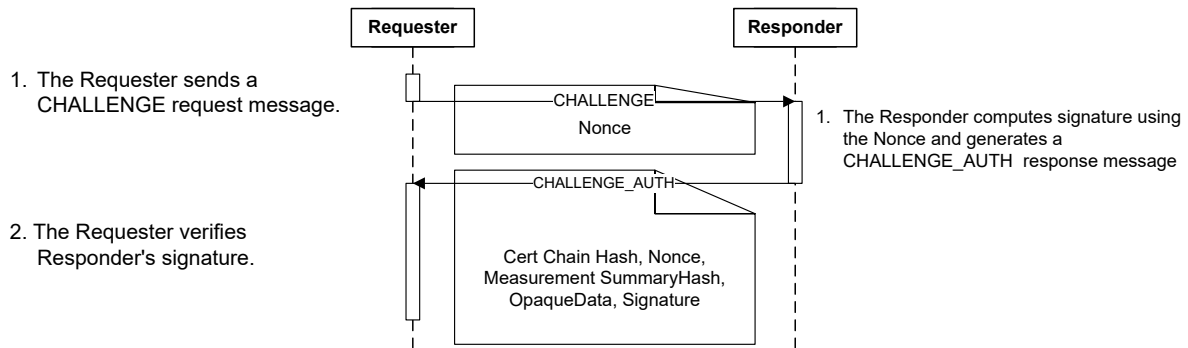
444 where:

- `SPDMsignatureVerify` is described in [Signature verification](#). If `result` is `success`, the verification was successful.
- `PubKey` shall be the public key associated with the leaf certificate of the Responder with `slot=Param1` of the `CHALLENGE` request message. If the public key of the Responder was provisioned to the Requester, `PubKey` is the provisioned public key.

445 [Figure 10 — Responder authentication: Runtime challenge-response flow](#) shows the high-level request-response message flow and sequence for the authentication of the Responder for runtime challenge-response.

446 **Figure 10 — Responder authentication: Runtime challenge-response flow**

447



448 10.9.2.1 Request ordering and message transcript computation rules for M1 and M2

449 This clause applies to Responder-only authentication.

450 [Table 47 — Request ordering and message transcript computation rules for M1/M2](#) 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.

451 The possible request orderings leading up to and including CHALLENGE shall be:

- GET_VERSION , GET_CAPABILITIES , NEGOTIATE_ALGORITHMS , GET_DIGESTS , GET_CERTIFICATE , CHALLENGE (A1, B1, C1)
- GET_VERSION , GET_CAPABILITIES , NEGOTIATE_ALGORITHMS , GET_DIGESTS , CHALLENGE (A1, B3, C1)
- GET_VERSION , GET_CAPABILITIES , NEGOTIATE_ALGORITHMS , GET_CERTIFICATE , CHALLENGE (A1, B4, C1)
- GET_VERSION , GET_CAPABILITIES , NEGOTIATE_ALGORITHMS , CHALLENGE (A1, B2, C1)
- GET_DIGESTS , GET_CERTIFICATE , CHALLENGE (A2, B1, C1)
- GET_DIGESTS , CHALLENGE (A2, B3, C1)
- GET_CERTIFICATE , CHALLENGE (A2, B4, C1)
- CHALLENGE (A2, B2, C1)

452 Immediately after Reset, M1 and M2 shall be null .

453 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 . If a Negotiated State has been established, this will remain intact.

454 If a Requester sends a GET_VERSION message, the Requester and Responder shall set M1 and M2 to null , clear all Negotiated State and recommence construction of M1 and M2 starting with the new GET_VERSION message.

455 For additional rules, see [general ordering rules](#).

456

Table 47 — Request ordering and message transcript computation rules for M1/M2

Requests	Implementation requirements	M1/M2=Concatenate(A, B, C)
Initial value	N/A	M1/M2=null
GET_VERSION issued	Requester issues this request to allow the Requester and Responder to determine an agreed-upon Negotiated State. Also issued when the Requester detects an out-of-sync condition, or when the signature verification fails, or when the Responder provides an unexpected error response.	M1/M2=null
GET_VERSION , GET_CAPABILITIES , NEGOTIATE_ALGORITHMS issued	Requester shall always issue these requests in this order.	A1= VCA
GET_VERSION , GET_CAPABILITIES , NEGOTIATE_ALGORITHMS skipped	After M1/M2 were re-initialized to null due to a Reset or a completed CHALLENGE_AUTH response, Requester skipped these requests if the Responder had previously indicated CACHE_CAP=1. In this case, the Requester and Responder shall proceed with the previously determined Negotiated State. These requests and responses are still required for M1/M2 construction.	A2= VCA
GET_DIGESTS , GET_CERTIFICATE issued	After NEGOTIATE_ALGORITHMS request completion or after M1/M2 were re-initialized to null due to a Reset or a completed CHALLENGE_AUTH response, Requester issued these requests in this order if it had skipped the previous three requests.	B1=Concatenate(GET_DIGESTS , DIGESTS , GET_CERTIFICATE , CERTIFICATE)
GET_DIGESTS , GET_CERTIFICATE skipped	After M1/M2 were re-initialized to null due to a Reset or a completed CHALLENGE_AUTH response, Requester skipped these requests because it could use previously cached certificate information.	B2=null
GET_DIGESTS issued, GET_CERTIFICATE skipped	After M1/M2 were re-initialized to null due to a Reset or a completed CHALLENGE_AUTH response, Requester skipped the GET_CERTIFICATE request because it could use the previously cached CERTIFICATE response.	B3=Concatenate(GET_DIGESTS , DIGESTS)
GET_DIGESTS skipped, GET_CERTIFICATE issued	After M1/M2 were re-initialized to null due to a Reset or a completed CHALLENGE_AUTH response, Requester skipped the GET_DIGESTS request because it could use the previously cached CERTIFICATE response to make a byte-by-byte comparison.	B4=Concatenate(GET_CERTIFICATE , CERTIFICATE)
CHALLENGE issued	Requester issued this request to complete security verification of current requests and responses. The Signature bytes of CHALLENGE_AUTH shall not be included in C.	C1=Concatenate(CHALLENGE , CHALLENGE_AUTH(excluding Signature)) . See Table 44 — CHALLENGE request message format.
CHALLENGE completion	Completion of CHALLENGE sets M1/M2 to null.	M1/M2=null
Other issued	If the Requester issued commands other than GET_DIGESTS , GET_CERTIFICATE , and CHALLENGE and skipped CHALLENGE completion, then M1/M2 are set to null.	M1/M2=null

457 The Basic mutual authentication flow is DEPRECATED. Implementations should use session-based mutual authentication as [Figure 21 — Session-based mutual authentication example](#) shows or optimized session-based mutual authentication as [Figure 22 — Optimized session-based mutual authentication example](#) shows.

458 DEPRECATED

459 10.9.3 Basic mutual authentication

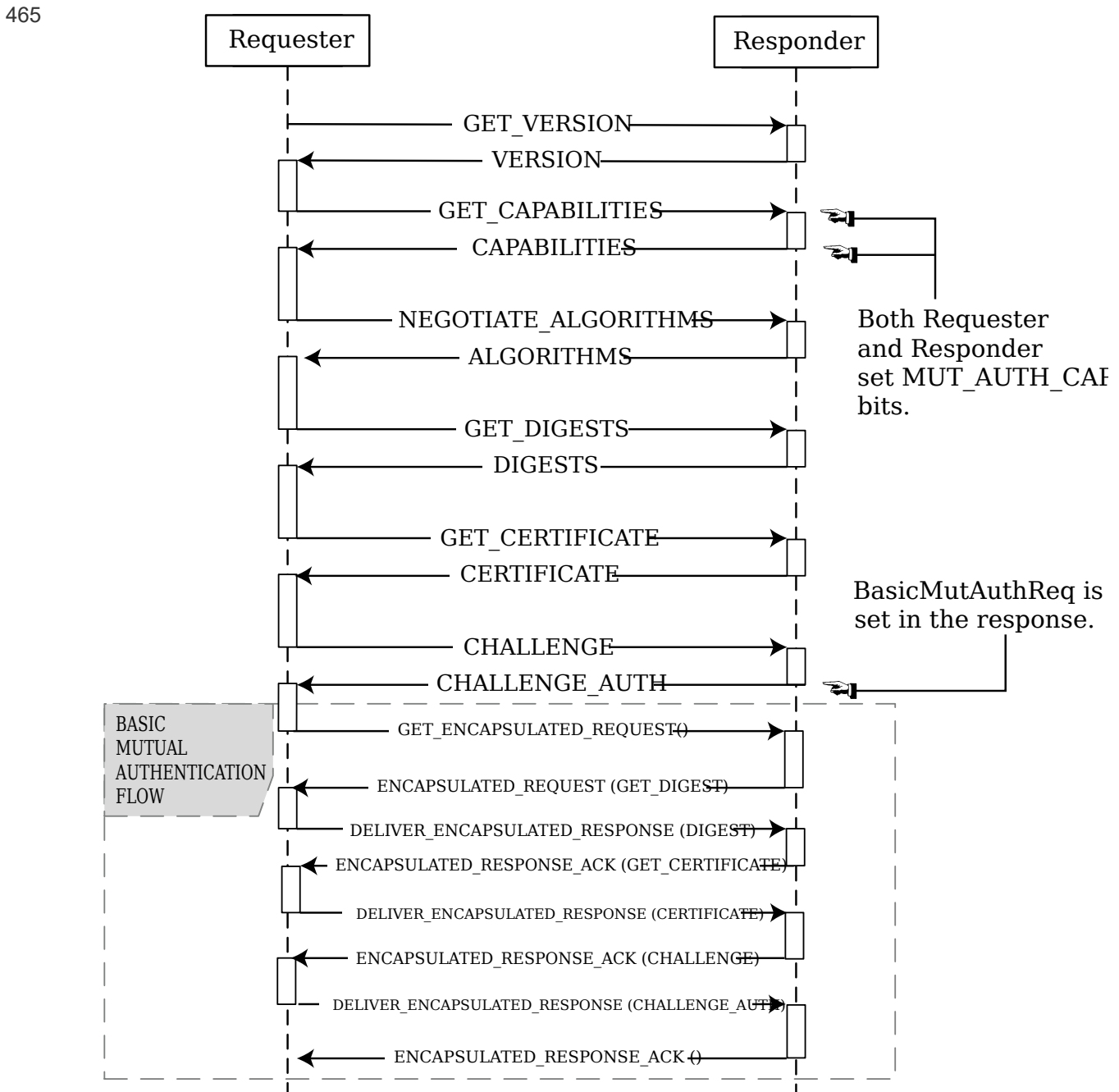
460 Unless otherwise stated, if the Requester supports mutual authentication, the requirements placed on the Responder in the [CHALLENGE request and CHALLENGE_AUTH response messages](#) clause shall also apply to the Requester. Unless otherwise stated, if the Responder supports mutual authentication, the requirements placed on the Requester in the [CHALLENGE request and CHALLENGE_AUTH response messages](#) clause shall also apply to the Responder. The preceding two sentences essentially describe a role reversal, unless otherwise stated.

461 The basic mutual authentication flow shall start when the Requester successfully receives a `CHALLENGE_AUTH` with **BasicMutAuthReq** set. This flow shall utilize message encapsulation as described in the [GET_ENCAPSULATED_REQUEST request and ENCAPSULATED_REQUEST response messages](#) clauses to retrieve request messages. The basic mutual authentication flow shall end when the encapsulated request flow ends.

462 This flow shall only allow `GET_DIGESTS` , `GET_CERTIFICATE` , `CHALLENGE` , and their corresponding responses to be encapsulated. If other requests are encapsulated, the Requester can send an `ERROR` message of `ErrorCode=UnexpectedRequest` and shall terminate the flow.

463 [Figure 11 — Mutual authentication basic flow](#) illustrates, as an example, the basic mutual authentication flow.

464 **Figure 11 — Mutual authentication basic flow**



466 10.9.3.1 Mutual authentication message transcript

467 This clause applies to the Responder authenticating the Requester in a basic mutual authentication scenario.

468 [Table 39 — Basic mutual authentication message transcript](#) 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 when the Responder authenticates the Requester.

469 The possible request orderings for the basic mutual authentication flow shall be one of the following (the Flow ID is in parenthesis):

- `GET_DIGESTS` , `GET_CERTIFICATE` , `CHALLENGE` (*BMAF0*)
- `GET_DIGESTS` , `CHALLENGE` (*BMAF1*)
- `GET_CERTIFICATE` , `CHALLENGE` (*BMAF2*)
- `CHALLENGE` (*BMAF3*)

470 When the basic mutual authentication flow starts, that is, when `GET_ENCAPSULATED_REQUEST` is issued, M1 and M2 shall be set to `null`.

471 **Table 48 — Basic mutual authentication message transcript**

Flow ID	M1/M2
BMAF0	Concatenate(<code>VCA</code> , <code>GET_DIGESTS</code> , <code>DIGESTS</code> , <code>GET_CERTIFICATE</code> , <code>CERTIFICATE</code> , <code>CHALLENGE</code> , <code>CHALLENGE_AUTH</code> without the signature)
BMAF1	Concatenate(<code>VCA</code> , <code>GET_DIGESTS</code> , <code>DIGESTS</code> , <code>CHALLENGE</code> , <code>CHALLENGE_AUTH</code> without the signature)
BMAF2	Concatenate(<code>VCA</code> , <code>GET_CERTIFICATE</code> , <code>CERTIFICATE</code> , <code>CHALLENGE</code> , <code>CHALLENGE_AUTH</code> without the signature)
BMAF3	Concatenate(<code>VCA</code> , <code>CHALLENGE</code> , <code>CHALLENGE_AUTH</code> without the signature)

472 For `GET_CERTIFICATE` and `CERTIFICATE` , these messages might need to be issued multiple times to retrieve the entire certificate chain. Thus, each instance of the request and response shall be part of M1/M2 in the order that they are issued.

473 DEPRECATED

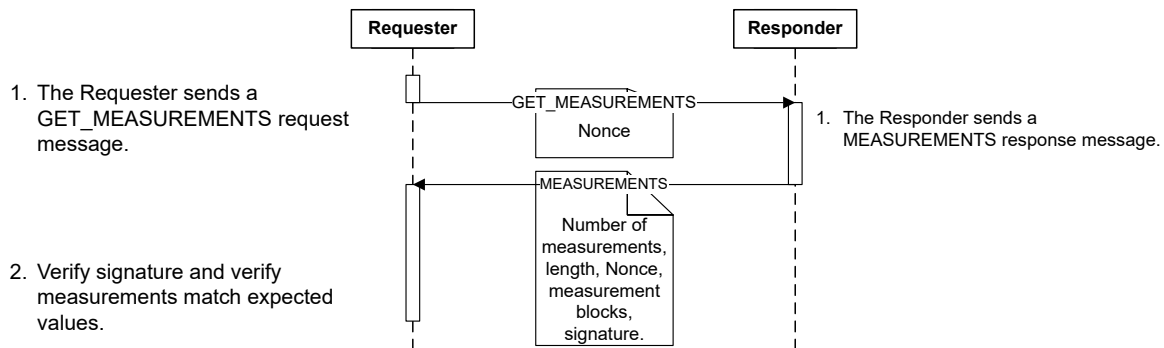
474 10.10 Firmware and other measurements

475 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 its `CAPABILITIES` response.

476 [Figure 12 — Measurement retrieval flow](#) shows the high-level request-response flow and sequence for endpoint measurement. If the `MEAS_FRESH_CAP` bit in the `CAPABILITIES` response message returns 0 and if the Requester

requires fresh measurements, the Responder shall be Reset before `GET_MEASUREMENTS` is resent. The mechanisms employed for Resetting the Responder are outside the scope of this specification.

Figure 12 — Measurement retrieval flow



10.11 `GET_MEASUREMENTS` request and `MEASUREMENTS` response messages

Measurements in SPDM are represented in the form of measurement *blocks*. A [measurement block](#) defines the measurement block structure. A device can present measurements of different elements of its internal state, as well as metadata to assist in the attestation of its state via measurements, as separate blocks. The `GET_MEASUREMENTS` request message enables a Requester to query a Responder for the number of individual measurement blocks it supports and request either specific blocks or all available blocks. The `MEASUREMENTS` response message returns the requested blocks. A collection of one or more measurement blocks is called a *measurement record*.

Because issuing `GET_MEASUREMENTS` clears the [M1/M2 message transcript](#), it is recommended that a Requester does not send this message until it has received at least one successful `CHALLENGE_AUTH` response message from the Responder. This ensures that the information in message pairs `GET_DIGESTS` / `DIGESTS` and `GET_CERTIFICATE` / `CERTIFICATE` has been authenticated at least once.

[Table 49 — `GET_MEASUREMENTS` request message format](#) shows the `GET_MEASUREMENTS` request message format.

[Table 50 — `GET_MEASUREMENTS` request attributes](#) shows the `GET_MEASUREMENTS` request message attributes.

[Table 52 — Successful `MEASUREMENTS` response message format](#) shows the `MEASUREMENTS` response message format. The measurement blocks in `MeasurementRecord` shall be sorted in ascending order by index.

Table 49 — `GET_MEASUREMENTS` request message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0xE0</code> = <code>GET_MEASUREMENTS</code> . See Table 4 — SPDM request codes .
2	Param1	1	Shall be Request attributes. See Table 50 — GET_MEASUREMENTS request attributes .
3	Param2	1	Shall be a <code>Measurement</code> operation. <ul style="list-style-type: none"> A value of <code>0x0</code> shall query the Responder for the total number of measurement blocks available. A value of <code>0xFF</code> shall request all measurement blocks. A value between <code>0x1</code> and <code>0xFE</code>, inclusive, shall request the measurement block at the index corresponding to that value.
4	Nonce	NL = 32 or 0	The Requester should choose a random value. This field is only present if Bit [0] of <code>Param1</code> is <code>1</code> . See Table 50 — GET_MEASUREMENTS request attributes .
4 + NL	SlotIDParam	1	This field is only present if Bit [0] of <code>Param1</code> is <code>1</code> . <ul style="list-style-type: none"> Bit [7:4]. Reserved. Bit [3:0]. Shall be the <code>SlotID</code>. Slot number of the Responder certificate chain that shall be used for authenticating the measurement(s). If the Responder's public key was provisioned to the Requester previously, this field shall be <code>0xF</code>. See Table 50 — GET_MEASUREMENTS request attributes.
5 + NL	Context	8	The Requester can include application-specific information in Context. The Requester should fill this field with zeros if it has no context to provide.

486 **Table 50 — GET_MEASUREMENTS request attributes**

Bit offset	Field	Description
0	SignatureRequested	<p>If the Responder can generate a signature (<code>MEAS_CAP</code> is <code>10b</code> in the CAPABILITIES response and either <code>BaseAsymSel</code> or <code>ExtAsymSelCount</code> is non-zero) a value of <code>1</code> indicates that a signature on the measurement log is required. The <code>Nonce</code> field shall be present in the request when this bit is set. The Responder shall generate and send a signature in the response.</p> <p>A value of <code>0</code> indicates that the Requester does not require a signature. The Responder shall not generate a signature in the response. The <code>Nonce</code> field shall be absent in the request.</p> <p>For Responders that cannot generate a signature (<code>MEAS_CAP</code> is <code>01b</code> in the CAPABILITIES response or both <code>BaseAsymSel</code> and <code>ExtAsymSelCount</code> are zero), the Requester shall always use <code>0</code>.</p>

Bit offset	Field	Description
1	RawBitStreamRequested	<p>This bit is applicable only if the measurement specification supports only two representations, raw bit stream and digest, such as when <code>MeasurementSpecification</code> of the Measurement block format is set to <code>DMTF</code>, as Table 53 — Measurement block format describes. If the measurement specification supports other representations, this bit is ignored.</p> <p>If the Responder can return either a raw bit stream or a hash for the requested measurement, value <code>1</code> shall request the Responder to return the raw bit stream version of such measurement. If the Responder cannot return a raw bit stream for the measurement (for example, if the raw bit stream contains confidential data that the Responder cannot expose), it shall return the corresponding hash. Another scenario in which the Responder cannot return a raw bit stream is when the <code>MEASUREMENTS</code> message is greater than the <code>MaxSPDMmsgSize</code> of the Requester. In cases where the Responder cannot return a raw bit stream, the Requester can simply request a digest.</p> <p>Value <code>0</code> shall request the Responder to return a hash version of the measurement. If the Responder cannot return a hash of the measurement (for example, if the measurement represents a data structure where a digest is not applicable), it shall return the corresponding raw bit stream.</p>
2	NewMeasurementRequested	<p>If the Responder has pending updates to measurement blocks that have not yet taken effect, then value <code>1</code> shall be used to request the Responder to return new values of the measurement blocks at the indices requested in <code>Param2</code>.</p> <p>Value <code>0</code> shall be used to request the Responder to return the current values of the measurement blocks at the requested indices.</p> <p>If the Responder has no pending updates to the measurement blocks at the requested indices, then the Responder shall return the current values of the measurement blocks, regardless of the value of <code>NewMeasurementRequested</code>.</p>
[7:3]	Reserved	Reserved.

487 Measurement index assignments

488 This specification imposes no requirements on the scope, type, or format of measurement a device associates with a particular measurement index in the range `0x1` to `0xEF`. As a result, Responders can use the same index to report different types of measurements based on their implementation. If available, a Requester can use a measurement manifest to discover information about the specific measurement types available from a particular Responder and the indices to which they correspond. When measurements follow the DMTF measurement specification format that [Table 54 — DMTF measurement specification format](#) describes, a measurement with a `DMTFSpecMeasurementValueType[6:0]` equal to either `0x04` or `0x0A` is the measurement manifest.

489 To aid interoperability, this specification reserves indices `0xF0` to `0xFE` inclusive for specific purposes. If a Responder supports a type of measurement that [Table 51 — Measurement index assigned range](#) defines, it shall always assign to it the corresponding index value. A Responder shall not assign indices `0xF0` to `0xFE` to measurements types other than those that [Table 51 — Measurement index assigned range](#) defines.

490 Table 51 — Measurement index assigned range

Measurement Index	Measurement type	Description
0xF0 - 0xFC	Reserved	Reserved.
0xFD	Measurement manifest	Shall be the metadata on available measurements, as type <code>DMTFSpecMeasurementValueType[6:0] = 0x04</code> or <code>DMTFSpecMeasurementValueType[6:0] = 0x0A</code> defines.
0xFE	Device mode	Shall be structured device mode information, as type <code>DMTFSpecMeasurementValueType[6:0] = 0x05</code> defines.

491

Table 52 — Successful MEASUREMENTS response message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x60</code> = MEASUREMENTS . See Table 5 — SPDM response codes .
2	Param1	1	When <code>Param2</code> in the requested measurement operation is <code>0</code> , this parameter shall return the total number of measurement indices on the device. Otherwise, this field is reserved.
3	Param2	1	<p>Bit [7:6]. Reserved.</p> <p>Bit [5:4]. Content changed. If this message contains a signature, this field shall indicate if one or more <code>MeasurementRecord</code> fields of previous <code>MEASUREMENTS</code> responses in the same measurement log have changed.</p> <p><code>00b</code> : The Responder does not detect changes of <code>MeasurementRecord</code> fields of previous <code>MEASUREMENTS</code> responses in the same measurement log, or this message does not contain a signature.</p> <p><code>01b</code> : The Responder detected that one or more <code>MeasurementRecord</code> fields of previous <code>MEASUREMENTS</code> responses in the measurement log being signed have changed. The Requester might consider issuing <code>GET_MEASUREMENTS</code> again to acquire latest measurements.</p> <p><code>10b</code> : The Responder detected no change in <code>MeasurementRecord</code> fields of previous <code>MEASUREMENTS</code> responses in the measurement log being signed.</p> <p><code>11b</code> : Reserved.</p> <p>Bit [3:0]. Shall be the <code>SlotID</code> . If this message contains a signature, this field shall contain the slot number of the certificate chain specified in the <code>GET_MEASUREMENTS</code> request, or <code>0xF</code> if the Responder's public key was provisioned to the Requester previously. If this message does not contain a signature, this field shall be set to <code>0x0</code> .</p>

Byte offset	Field	Size (bytes)	Description
4	NumberOfBlocks	1	Shall be the number of measurement blocks in the MeasurementRecord . If Param2 in the requested measurement operation is 0 , this field shall be 0 .
5	MeasurementRecordLength	3	Shall be the size of the MeasurementRecord in bytes. If Param2 in the requested measurement operation is 0 , this field shall be 0 .
8	MeasurementRecord	$L = \text{MeasurementRecordLength}$	Shall be the 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. This field shall always be present.
$40 + L$	OpaqueDataLength	2	Shall be the size of the OpaqueData field that follows in bytes. The value should not be greater than 1024 bytes. Shall be 0 if no OpaqueData is provided.
$42 + L$	OpaqueData	OpaqueDataLength	The Responder can include Responder-specific information and/or information that its transport defines. If present, this field shall conform to the selected opaque data format in OtherParamsSelection .
$42 + L + \text{OpaqueDataLength}$	RequesterContext	8	This field shall be identical to the Context field of the corresponding request message.
$50 + L + \text{OpaqueDataLength}$	Signature	SigLen	Shall be Signature of the measurement log, excluding the Signature field and signed using the private key associated with the leaf certificate. The Responder shall use the asymmetric signing algorithm it selected during the last ALGORITHMS response message to the Requester, and SigLen is the size of that asymmetric signing algorithm output. This field is conditional and is only present in the MEASUREMENTS response corresponding to a GET_MEASUREMENTS request with Param1[0] set to 1 .

492 10.11.1 Measurement block

493 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 corresponds to a particular measurement index and measurement type.

494 [Table 53 — Measurement block format](#) shows the format for a measurement block:

495 **Table 53 — Measurement block format**

Byte offset	Field	Size (bytes)	Description
0	Index	1	Shall be the index. When <code>Param2</code> of the <code>GET_MEASUREMENTS</code> request is between <code>0x1</code> and <code>0xFE</code> , inclusive, this field shall match the request. Otherwise, this field shall represent the index of the measurement block, where the index starts at 1 and ends at the index of the last measurement block.
1	MeasurementSpecification	1	Bit mask. The value shall indicate the measurement specification that the requested <code>Measurement</code> follows and shall match the selected measurement specification (<code>MeasurementSpecificationSel</code>) in the <code>ALGORITHMS</code> message. See Table 21 — Successful ALGORITHMS response message format . Only one bit shall be set. The Measurement specification field format table defines the format for this field.
2	MeasurementSize	2	Shall be the size of <code>Measurement</code> , in bytes.
4	Measurement	<code>MeasurementSize</code>	Shall be the measurement value whose format the selected measurement specification (<code>MeasurementSpecificationSel</code>) defines. If <code>DMTFmeasSpec</code> is selected, the format of this field shall be as Table 54 — DMTF measurement specification format defines.

496 10.11.1.1 DMTF specification for the Measurement field of a measurement block

497 The present clause is the specification for the format of the `Measurement` field in a measurement block when the `MeasurementSpecification` field's Bit 0 (DMTF) is set. [Table 54 — DMTF measurement specification format](#) specifies this format.

498 10.11.1.1.1 Measurement manifest

499 A measurement manifest refers to a data structure that describes the contents of other indices or itself contains measurements. For instance, a manifest may describe which indices describe the different firmware modules' measurements. When the [Table 54 — DMTF measurement specification format](#) is in use, this specification defines multiple overarching manifest formats, as described in the [DMTFSpecMeasurementValueType values table](#).

500 When `DMTFSpecMeasurementValueType[6:0]=0x04`, the measurement manifest type is a freeform manifest. When read, the manifest data is placed in the `Measurement` field of the [Table 53 — Measurement block format](#). The format of a freeform manifest is implementation specific and outside the scope of this specification.

501 When `DMTFSpecMeasurementValueType[6:0]=0x0A`, the measurement manifest type is a structured measurement manifest. The structured manifest starts with an `SVH` header as [Table 56 — Manifest measurement block format](#) describes. The `SVH` header is used to indicate the standards body or vendor that defines the manifest format. The format of the `Manifest` data in a structured measurement manifest is outside the scope of this specification.

502 10.11.1.1.2 Hash-extend measurements

503 A device may support reporting of measurements through an "extend" scheme, which works as follows:

```

initialize HEM = MH bytes of 0s
for each extend operation, perform HEM = hash(Concatenate(HEM, DataToExtend)) for all data
elements to extend

```

504 An example of such a scheme is the Platform Configuration Register "extend" function in Trusted Platform Modules. The `hash()` function is the measurement hash algorithm specified by the most recent `ALGORITHMS` response message. The initial value of a hash-extend measurement (HEM) shall be `MH` bytes whose bits are all set to `0`, where `MH` is the size of `MeasurementHashAlgo` in the most recent `ALGORITHMS` response message. The hash-extend measurement is updated by "extending" the current value to include the next data to extend (`DataToExtend`). The extend operation is calculating the digest of the current value concatenated with the data to extend. Then repeat the extend operation for additional data to extend.

505 The hash extended measurements is reported in a measurement block. A Responder that reports hash-extend measurements shall set `DMTFSpecMeasurementValueType[6:0]` to `0x8` for the corresponding measurement index.

506 **Table 54 — DMTF measurement specification format**

Byte offset	Field	Size (bytes)	Description
0	<code>DMTFSpecMeasurementValueType</code>	1	<p>Composed of:</p> <ul style="list-style-type: none"> Bit [7]. Shall indicate the representation in <code>DMTFSpecMeasurementValue</code>. Bit [6:0]. Indicates what is being measured by <code>DMTFSpecMeasurementValue</code>. <p>These values are set independently and are interpreted as follows:</p> <ul style="list-style-type: none"> [7]=0b . Digest. [7]=1b . Raw bit stream. The Responder should ensure the raw bit stream does not contain secrets. See DMTFSpecMeasurementValueType values for defined values for <code>DMTFSpecMeasurementValueType[6:0]</code>.
1	<code>DMTFSpecMeasurementValueSize</code>	2	<p>Shall be the size of <code>DMTFSpecMeasurementValue</code>, in bytes.</p> <p>When <code>DMTFSpecMeasurementValueType[7]=0b</code>, the <code>DMTFSpecMeasurementValueSize</code> shall be derived from the measurement hash algorithm that the <code>ALGORITHMS</code> response message returns.</p>
3	<code>DMTFSpecMeasurementValue</code>	MS	<p>Shall be the cryptographic hash or raw bit stream, as indicated in <code>DMTFSpecMeasurementValueType[7]</code>. For cryptographic hashes or digests, this field shall be in hash byte order. The vendor defines the byte order for raw bit streams.</p>

507 **Table 55 — DMTFSpecMeasurementValueType values**

DMTFSpecMeasurementValueType[6:0]	Description
0x0	Immutable ROM.
0x1	Mutable firmware.
0x2	Hardware configuration, such as straps.
0x3	Firmware configuration, such as configurable firmware policy.
0x4	Freeform measurement manifest. When <code>DMTFSpecMeasurementValueType[6:0]=0x4</code> , the Responder should support setting <code>DMTFSpecMeasurementValueType[7]</code> to either <code>0b</code> or <code>1b</code> . The format of this manifest is device specific.
0x5	Structured representation of debug and device mode. See Device mode field of a measurement block . When <code>DMTFSpecMeasurementValueType[6:0]=0x5</code> , <code>DMTFSpecMeasurementValueType[7]</code> shall be set to <code>1b</code> .
0x6	Mutable firmware's version number. This specification does not mandate a format for firmware version number. When <code>DMTFSpecMeasurementValueType[6:0]=0x6</code> , <code>DMTFSpecMeasurementValueType[7]</code> should be set to <code>1b</code> .
0x7	Mutable firmware's security version number, which should be formatted as an 8-byte unsigned integer. When <code>DMTFSpecMeasurementValueType[6:0]=0x7</code> , <code>DMTFSpecMeasurementValueType[7]</code> should be set to <code>1b</code> .
0x8	Hash-extend measurement. The measurement reported is an <code>HEM</code> value as defined in Hash-extend measurements . When <code>DMTFSpecMeasurementValueType[6:0]=0x8</code> , <code>DMTFSpecMeasurementValueType[7]</code> shall be set to <code>0b</code> .
0x9	Informational. The measurement is for the Requester's information only and does not carry sensitive security attributes. For example, human-readable boot progress information. When <code>DMTFSpecMeasurementValueType[6:0]=0x9</code> , <code>DMTFSpecMeasurementValueType[7]</code> shall be set to <code>1b</code> .
0xA	Structured measurement manifest. When <code>DMTFSpecMeasurementValueType[6:0]=0xA</code> , the Responder shall support setting <code>DMTFSpecMeasurementValueType[7]</code> to <code>1b</code> , and should support setting <code>DMTFSpecMeasurementValueType[7]</code> to <code>0b</code> . The manifest shall follow the format described in Manifest format for a measurement block .
All other values	Reserved.

508 10.11.1.2 Device mode field of a measurement block

Byte offset	Field	Size (bytes)	Description
0	OperationalModeCapabilities	4	<p>Fields with bits set to 1 indicate support for reporting the associated state in <code>OperationalModeState</code>.</p> <ul style="list-style-type: none"> Bit [0]. Shall indicate support for reporting device in manufacturing mode. Bit [1]. Shall indicate support for reporting device in validation mode. Bit [2]. Shall indicate support for reporting device in normal operational mode. Bit [3]. Shall indicate support for reporting device in recovery mode. Bit [4]. Shall indicate support for reporting device in Return Merchandise Authorization (RMA) mode. Bit [5]. Shall indicate support for reporting device in decommissioned mode. All other values reserved.
4	OperationalModeState	4	<p>Fields with bits set to 1 indicate true for the reported state.</p> <ul style="list-style-type: none"> Bit [0]. Shall indicate the device is in manufacturing mode. Bit [1]. Indicates the device is in validation mode. Bit [2]. Shall indicate the device is in normal operational mode. Bit [3]. Shall indicate the device is in recovery mode. Bit [4]. Shall indicate the device is in RMA mode. Bit [5]. Shall indicate the device is in decommissioned mode. All other values reserved.
8	DeviceModeCapabilities	4	<p>Fields with bits set to 1 indicate support for reporting the associated state in <code>DeviceModeState</code>.</p> <ul style="list-style-type: none"> Bit [0]. Shall indicate support for reporting <code>non-invasive debug mode</code> is active. Bit [1]. Shall indicate support for reporting <code>invasive debug mode</code> is active. Bit [2]. Shall indicate support for reporting non-invasive debug mode has been active this Reset cycle. Bit [3]. Shall indicate support for reporting invasive debug mode has been active this Reset cycle. Bit [4]. Shall indicate support for reporting invasive debug mode has been active on this device at least once since exiting manufacturing mode. All other values reserved.

Byte offset	Field	Size (bytes)	Description
12	DeviceModeState	4	Fields with bits set to 1 indicate true for the reported state. <ul style="list-style-type: none"> • Bit [0]. Shall indicate non-invasive debug mode is active. • Bit [1]. Shall indicate invasive debug mode is active. • Bit [2]. Shall indicate non-invasive debug mode has been active this Reset cycle. • Bit [3]. Shall indicate invasive debug mode has been active this Reset cycle. • Bit [4]. Shall indicate invasive debug mode has been active on this device at least once since exiting manufacturing mode. • All other values reserved.

509 10.11.1.3 Manifest format for a measurement block

510 When `DMTFSpecMeasurementValueType[6:0]=0xA`, the response shall be either a manifest or the digest of a manifest. If `DMTFSpecMeasurementValueType[7]=0b`, then the `Measurement` field of the [Measurement block](#) shall contain a digest of the structure described in [Table 56 — Manifest measurement block format](#). If `DMTFSpecMeasurementValueType[7]=1b`, then the `Measurement` field of the [Measurement block](#) shall contain a manifest in the format described in [Table 56 — Manifest measurement block format](#).

511 **Table 56 — Manifest measurement block format**

Byte offset	Field	Size (bytes)	Description
0	SVH	2 + <code>VendorIDLen</code>	Shall be a standards body or vendor-defined header , as described in Table 64 — Standards body or vendor-defined header (SVH) .
2 + <code>VendorIDLen</code>	Manifest	Variable	Shall contain the manifest data, as defined by the registry, standards body, or vendor specified in the <code>ID</code> and <code>VendorID</code> fields.

512 10.11.2 MEASUREMENTS signature generation

513 While a Requester can opt to require a signature in each of the request-response messages, it is advisable that the cost of the signature generation process is minimized by amortizing it over multiple request-response messages where applicable. In this scheme, the Requester issues a number of requests without requiring signatures followed by a final request requiring a signature over the entire set of request-response messages exchanged. The steps to complete this scheme are as follows:

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

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

```
GET_MEASUREMENTS_REQUESTn, MEASUREMENTS_RESPONSEn)
```

515 where:

- `Concatenate` is the standard concatenation function.
- `GET_MEASUREMENTS_REQUEST1` is the entire first `GET_MEASUREMENTS` request message under consideration, where the Requester has not requested a signature on that specific `GET_MEASUREMENTS` request.
- `MEASUREMENTS_RESPONSE1` is the entire `MEASUREMENTS` response message without the signature bytes that the Responder sent in response to `GET_MEASUREMENTS_REQUEST1`.
- `GET_MEASUREMENTS_REQUESTn-1` is the 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` is the entire `MEASUREMENTS` response message without the signature bytes that the Responder sent in response to `GET_MEASUREMENTS_REQUESTn-1`.
- `GET_MEASUREMENTS_REQUESTn` is the 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` is the entire `MEASUREMENTS` response message without the signature bytes that the Responder sent in response to `GET_MEASUREMENTS_REQUESTn`.

516 Any communication between Requester and Responder other than a `GET_MEASUREMENTS` request or response re-initializes L1/L2 computation to `null`. The `GET_MEASUREMENTS` requests and `MEASUREMENTS` responses before the L1/L2 re-initialization will not be covered by the signature in the final `MEASUREMENTS` response. Consequently, it is recommended that the Requester not use the measurements before verifying the signature.

517 An `ERROR` message of `ErrorCode=ResponseNotReady` shall not re-initialize L1/L2 - Requester and Responder shall continue to construct L1/L2 with `GET_MEASUREMENTS` and `MEASUREMENTS`. An error response with any error code other than `ResponseNotReady` shall re-initialize L1/L2 to `null`.

518 2. The Responder shall generate:

```
Signature = SPDMsign(PrivKey, L1, "measurements signing");
```

519 where:

- `SPDMsign` is described in [Signature generation](#).

- `PrivKey` shall be the private key of the Responder associated with the leaf certificate stored in `SlotID` of `SlotIDParam` in `GET_MEASUREMENTS`. If the public key of the Responder was provisioned to the Requester, then `PrivKey` shall be the associated private key.

520 10.11.3 MEASUREMENTS signature verification

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

522 1. The Requester shall perform:

```
result = SPDMsignatureVerify(PubKey, Signature, L2, "measurements signing")
```

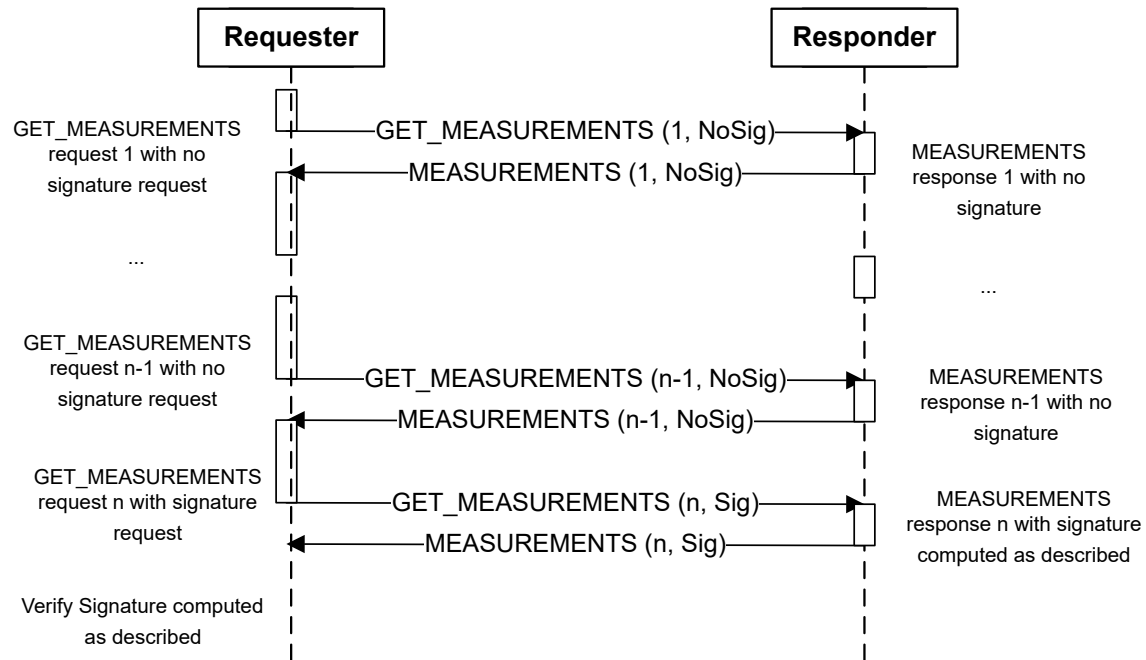
523 where:

- `SPDMsignatureVerify` is described in [Signature verification](#). A successful verification is when `result` is `success`.
- `PubKey` shall be the public key associated with the leaf certificate stored in `SlotID` of `SlotIDParam` in `GET_MEASUREMENTS`. `PubKey` is extracted from the `CERTIFICATE` response. If the public key of the Responder was provisioned to the Requester, then `PubKey` shall be the provisioned public key.

524 [Figure 13 — Measurement signature computation example](#) shows an example of a typical Requester-Responder protocol where the Requester issues 1 to $n-1$ `GET_MEASUREMENTS` requests without a signature, which is followed by a single `GET_MEASUREMENTS` request n with a signature.

525 **Figure 13 — Measurement signature computation example**

526



527 10.12 ERROR response message

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

529 [Table 57 — ERROR response message format](#) shows the `ERROR` response format.

530 [Table 58 — Error code and error data](#) shows the detailed error code, error data, and extended error data.

531 [Table 59 — ResponseNotReady extended error data](#) shows the `ResponseNotReady` extended error data.

532 [Table 60 — Registry or standards body ID](#) shows the registry or standards body ID.

533 [Table 61 — ExtendedErrorData format for vendor or other standards-defined ERROR response message](#) shows the `ExtendedErrorData` format definition for vendor or other standards-defined `ERROR` response messages.

534 Table 57 — ERROR response message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .

Byte offset	Field	Size (bytes)	Description
1	RequestResponseCode	1	Shall be 0x7F = ERROR. See Table 5 — SPDM response codes.
2	Param1	1	Shall be the ErrorCode. See Table 58 — Error code and error data.
3	Param2	1	Shall be the Error data. See Table 58 — Error code and error data.
4	ExtendedErrorData	0-32	Shall be Optional extended data. See Table 58 — Error code and error data.

535 **Table 58 — Error code and error data**

ErrorCode	Value	Description	Error data	ExtendedErrorData
Reserved	0x00	Reserved.	Reserved	Reserved
InvalidRequest	0x01	One or more request fields are invalid	0x00	No extended error data is provided.
Reserved	0x02	Reserved.	Reserved	No extended error data is provided.
Busy	0x03	The Responder received the request message and the Responder decided to ignore the request message, but the Responder might 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 cannot decrypt or verify data during the session.	Reserved	No extended error data is provided.
UnsupportedRequest	0x07	The RequestResponseCode or the SubCode (if applicable) in the request message is unsupported.	RequestResponseCode or the SubCode in the request message.	No extended error data is provided
RequestInFlight	0x08	The Responder has delivered an encapsulated request to which it is still waiting for the response.	Reserved	No extended error data is provided.
InvalidResponseCode	0x09	The Requester delivered an invalid response for an encapsulated response.	Reserved	No extended error data is provided.
SessionLimitExceeded	0x0A	Maximum number of concurrent sessions reached.	Reserved	No extended error data is provided.

ErrorCode	Value	Description	Error data	ExtendedErrorData
SessionRequired	0x0B	The Request message received by the Responder is only allowed within a session.	Reserved	No extended error data is provided.
ResetRequired	0x0C	The device requires a reset to complete the requested operation. This ErrorCode can be sent in response to the GET_CSR or SET_CERTIFICATE message.	Bit[7:3]. Reserved. Bit[2:0]. If sent in response to GET_CSR, the Responder-assigned CSRTackingTag. Otherwise, shall be 0.	No extended error data is provided.
ResponseTooLarge	0x0D	The response is greater than the MaxSPDMMsgSize of the requesting SPDM endpoint.	Reserved	See Table 62 — ExtendedErrorData format for ResponseTooLarge.
RequestTooLarge	0x0E	The request is greater than the MaxSPDMMsgSize of the receiving SPDM endpoint.	Reserved	Reserved
LargeResponse	0x0F	The response is greater than DataTransferSize and less than or equal to MaxSPDMMsgSize of the requesting SPDM endpoint.	Reserved	See Table 63 — ExtendedErrorData format for LargeResponse.
MessageLost	0x10	The SPDM message is lost. For example, this error code can be used to indicate the loss of a Large Request, Large Response, or the request in a ResponseNotReady.	Reserved	Reserved
InvalidPolicy	0x11	The Responder received one or more messages that violated its security policy. For example, if a Responder requires both encryption and MAC capabilities in a secure session, and the Requester only supports encryption, then the Responder would return this error code if the Requester sends KEY_EXCHANGE.	Reserved	Reserved
Reserved	0x12–0x40	Reserved	Reserved	Reserved
VersionMismatch	0x41	Requested SPDM version is not supported or is a different version from the selected version.	0x00	No extended error data is provided.
ResponseNotReady	0x42	See the RESPOND_IF_READY request message format.	0x00	See Table 59 — ResponseNotReady extended error data.
RequestResynch	0x43	Responder is requesting Requester to reissue GET_VERSION to re-synchronize. An example is following a firmware update.	0x00	No extended error data is provided.

ErrorCode	Value	Description	Error data	ExtendedErrorData
OperationFailed	0x44	An internal error occurred upon servicing the request issued by the Requester.	0x00	No extended error data is provided.
NoPendingRequests	0x45	The Responder does not have any pending request for a GET_ENCAPSULATED_REQUEST message.	Reserved	Reserved
Reserved	0x46–0xFE	Reserved.	Reserved	Reserved
Vendor or Standards-Defined	0xFF	Vendor or standards-defined	Shall indicate the registry or standards body using one of the values in the ID column of Table 60 — Registry or standards body ID.	See Table 61 — ExtendedErrorData format for vendor or other standards-defined ERROR response message for format definition.

536 **Table 59 — ResponseNotReady extended error data**

Byte offset	Field	Size (bytes)	Description
0	RDTExponent	1	<p>Shall be the exponent expressed in logarithmic (base-2 scale) to calculate RDT time in μs after which the Responder can provide successful completion response.</p> <p>For example, the raw value 8 indicates that the Responder will be ready in $2^8 = 256 \mu$s.</p> <p>Requester should use RDT to avoid continuous pinging and issue the RESPOND_IF_READY request message, as Table 65 — RESPOND_IF_READY request message format shows, after RDT time.</p> <p>For timing requirement details, see Table 7 — Timing specification for SPDM messages.</p>
1	RequestCode	1	Shall be the request code that triggered this response.
2	Token	1	Shall be the opaque handle that the Requester shall pass in with the RESPOND_IF_READY request message, as Table 65 — RESPOND_IF_READY request message format shows. The Responder can use the value in this field to provide the correct response when the Requester issues a RESPOND_IF_READY request.
3	RDTM	1	<p>Shall be the multiplier used to compute WT_{Max} in μs to indicate that the response might be dropped after this delay.</p> <p>The multiplier shall always be greater than 1.</p> <p>The Responder might also stop processing the initial request if the same Requester issues a different request.</p> <p>For timing requirement details, see Table 7 — Timing specification for SPDM messages.</p>

537 **Table 60 — Registry or standards body ID**

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

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 define any algorithms 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 the vendor ID assigned by USB.
0x3	2	PCI-SIG	Vendor is identified using PCI-SIG Vendor ID .
0x4	4	IANA	The Private Enterprise Number (PEN) assigned by the Internet Assigned Numbers Authority (IANA) identifies the vendor.
0x5	4	HDBaseT	Vendor is identified by using HDBaseT HDCD entity.
0x6	2	MIPI	The Manufacturer ID assigned by MIPI identifies the vendor.
0x7	2	CXL	Vendor is identified by using CXL vendor ID.
0x8	2	JEDEC	Vendor is identified by using JEDEC vendor ID.
0x9	0	VESA	For fields and formats defined by the VESA standards body, there is no Vendor ID registry.
0xA	Variable	IANA CBOR	The CBOR Tag Registry that identifies the format of the element, as assigned by the Internet Assigned Numbers Authority (IANA). The encoding of the CBOR tag indicates the length of the tag. When a CBOR Tag is used with a standards body or vendor-defined header , the <code>VendorIDLen</code> field shall be set to the length of the encoded CBOR tag, followed by the data payload, which starts with an encoded CBOR tag.

539 **Table 61 — ExtendedErrorData format for vendor or other standards-defined ERROR response message**

Byte offset	Field	Size (bytes)	Description
0	Len	1	<p>Shall be the length of the <code>VendorID</code> field.</p> <p>If the vendor defines the error, the value of this field shall equal the "Vendor ID length", as Table 60 — Registry or standards body ID describes, of the corresponding registry or standards body name.</p> <p>If a registry or standards body defines the error, this field shall be zero (0), which also indicates that the <code>VendorID</code> field is not present.</p> <p>The <code>Error Data</code> field in the <code>ERROR</code> message indicates the registry or standards body name (that is, <code>Param2</code>) and is one of the values in the ID column of Table 60 — Registry or standards body ID.</p>

Byte offset	Field	Size (bytes)	Description
1	VendorID	Len	The value of this field shall indicate the Vendor ID as assigned by the registry or standards body. Table 60 — Registry or standards body ID describes the length of this field. Shall be in little-endian format. The name of the registry or standards body in the <code>ERROR</code> is indicated in the <code>Error Data</code> field (that is, <code>Param2</code>) and is one of the values in the ID column of Table 60 — Registry or standards body ID .
1 + Len	OpaqueErrorData	Variable	The vendor or standards body defines this value.

540 **Table 62 — ExtendedErrorData format for ResponseTooLarge**

Byte offset	Field	Size (bytes)	Description
0	ActualSize	4	Shall be the size of the actual response.

541 **Table 63 — ExtendedErrorData format for LargeResponse**

Byte offset	Field	Size (bytes)	Description
0	Handle	1	Shall be a unique value that identifies the Large SPDM Response and shall be the same value for all chunks of the same large SPDM message. The value of this field should either sequentially increase or sequentially decrease with each large SPDM message with the expectation that it will wrap around after reaching the maximum or minimum value, respectively, of this field. See CHUNK_GET request and CHUNK_RESPONSE response message .

542 10.12.1 Standards body or vendor-defined header

543 This specification uses the format that [Table 64 — Standards body or vendor-defined header \(SVH\)](#) describes to help identify the entity that defines the format for a given payload. The clauses in the other parts of this specification indicate to which payload this header applies.

544 **Table 64 — Standards body or vendor-defined header (SVH)**

Byte offset	Field	Size (bytes)	Description
0	ID	1	Shall be one of the values in the <code>ID</code> column of Table 60 — Registry or standards body ID .

Byte offset	Field	Size (bytes)	Description
1	VendorIDLen	1	<p>Shall be the <code>Length</code> in bytes of the <code>VendorID</code> field.</p> <p>If the given payload belongs to a standards body or registry, this field shall be 0.</p> <p>Otherwise, the given payload belongs to the vendor and therefore, this field shall be the length indicated in the "Vendor ID length" column of Table 60 — Registry or standards body ID for the respective <code>ID</code>.</p>
2	VendorID	VendorIDLen	<p>If <code>VendorIDLen</code> is greater than zero, this field shall be the ID of the vendor corresponding to the <code>ID</code> field. Otherwise, this field shall be absent.</p>

545 10.13 RESPOND_IF_READY request message format

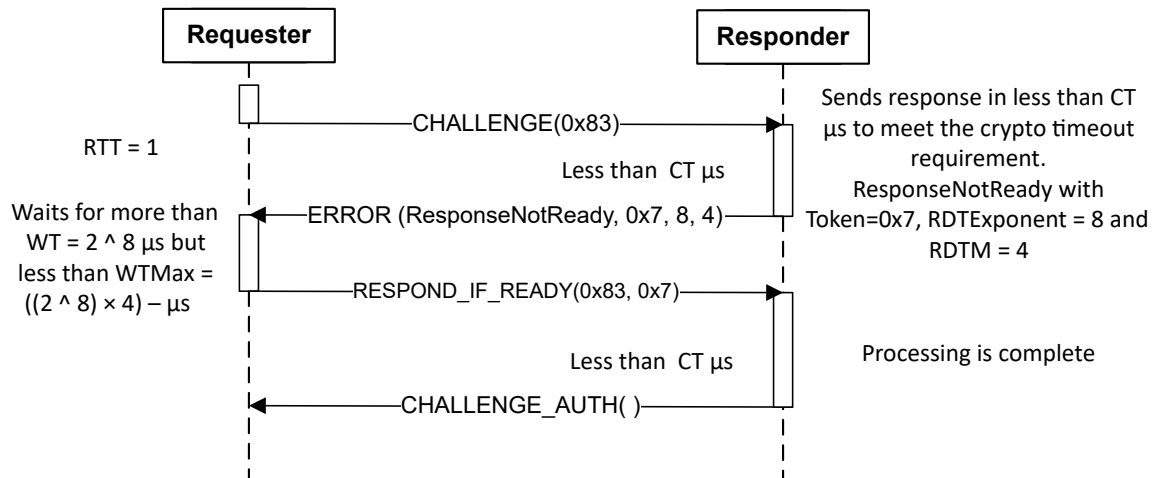
546 This request message shall ask for the response to the original request upon receipt of the `ResponseNotReady` error code. If the response to the original request is ready, the Responder shall return that response message. If the response to the original request is not ready, the Responder shall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

547 The validity of the `RESPOND_IF_READY` request (see the [SPDM Request and Response messages validity table](#)) is defined by the original request that caused the `RESPOND_IF_READY` flow. This means the last request to which the Responder sent an `ERROR` message of `ErrorCode=ResponseNotReady`.

548 [Figure 14 — RESPOND_IF_READY flow leading to completion](#) shows the `RESPOND_IF_READY` flow:

549 **Figure 14 — RESPOND_IF_READY flow leading to completion**

550



551 [Table 55 — RESPOND_IF_READY request message format](#) shows the RESPOND_IF_READY request message format.

552 **Table 65 — RESPOND_IF_READY request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in SPDM version.
1	RequestResponseCode	1	Shall be 0xFF = RESPOND_IF_READY. See Table 4 — SPDM request codes .
2	Param1	1	Shall be 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	Param2	1	Shall be the token that was returned as part of the ResponseNotReady extended error data.

553 10.14 VENDOR_DEFINED_REQUEST request message

554 A Requester intending to define a unique request to meet its needs can use this request message. [Table 66 — VENDOR_DEFINED_REQUEST request message format](#) defines the format.

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

556 If the vendor intends that these messages are to be used before a session has been established, and the vendor wishes to have the requests authenticated, then the vendor shall indicate how the transcript and/or message transcript are changed to add the vendor-defined commands.

557 [Table 66 — VENDOR_DEFINED_REQUEST request message format](#) shows the `VENDOR_DEFINED_REQUEST` request message format.

558 **Table 66 — VENDOR_DEFINED_REQUEST request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0xFE = VENDOR_DEFINED_REQUEST</code> . See Table 4 — SPDM request codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	StandardID	2	Shall indicate the registry or standards body by using one of the values in the ID column of Table 60 — Registry or standards body ID .
6	Len	1	Shall be the length of the <code>Vendor ID</code> field. If the <code>VendorDefinedReqPayload</code> is standards-defined, <code>Len</code> shall be <code>0</code> . If the <code>VendorDefinedReqPayload</code> is vendor-defined, <code>Len</code> shall equal "Vendor ID length", as Table 60 — Registry or standards body ID describes.
7	VendorID	Len	Shall be the Vendor ID as assigned by the registry or standards body. Shall be in little-endian format.
7 + Len	ReqLength	2	Shall be the length of the <code>VendorDefinedReqPayload</code> .
7 + Len + 2	VendorDefinedReqPayload	ReqLength	This field shall be used to send the request payload.

559 Other DMTF specifications may define `VENDOR_DEFINED_REQUEST` with `StandardID` set to 0. See [VendorDefinedReqPayload and VendorDefinedRespPayload defined by DMTF specifications](#) for more information.

560 10.15 VENDOR_DEFINED_RESPONSE response message

561 A Responder can use this response message in response to `VENDOR_DEFINED_REQUEST`. [Table 67 — VENDOR_DEFINED_RESPONSE response message format](#) defines the format.

562 **Table 67 — VENDOR_DEFINED_RESPONSE response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x7E = VENDOR_DEFINED_RESPONSE</code> . See Table 5 — SPDM response codes .

Byte offset	Field	Size (bytes)	Description
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	StandardID	2	Shall indicate the registry or standards body using one of the values in the ID column of Table 60 — Registry or standards body ID .
6	Len	1	Shall be the length of the <code>Vendor ID</code> field. If the <code>VendorDefinedRespPayload</code> is standards-defined, length shall be 0. If the <code>VendorDefinedRespPayload</code> is vendor-defined, length shall equal "Vendor ID length" as Table 60 — Registry or standards body ID 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	Shall be the length of the <code>VendorDefinedRespPayload</code>
7 + Len + 2	VendorDefinedRespPayload	ReqLength	This value shall be used to send the response payload.

563 10.15.1 VendorDefinedReqPayload and VendorDefinedRespPayload defined by DMTF specifications

564 Other DMTF specifications may define `VENDOR_DEFINED_REQUEST` and `VENDOR_DEFINED_RESPONSE` messages with `StandardID` set to 0 ("DMTF", as defined in [Table 50 — Registry or standards body ID](#)) and `Len` set to 0. In this case, `VENDOR_DEFINED_REQUEST` and `VENDOR_DEFINED_RESPONSE` messages shall specify the underlying DMTF specification that defines them. A DMTF specification which defines the data model of `VendorDefinedReqPayload` for `VENDOR_DEFINED_REQUEST` and the data model of `VendorDefinedRespPayload` for `VENDOR_DEFINED_RESPONSE` shall follow [Table 68 — Format of VendorDefinedReqPayload and VendorDefinedRespPayload when StandardID is DMTF](#).

565 **Table 68 — Format of VendorDefinedReqPayload and VendorDefinedRespPayload when StandardID is DMTF**

Byte offset	Field	Size (bytes)	Description
0	DSPNumber	2	Shall be the DMTF specification's DSP number as a 16-bit integer. For example, DSP0287 shall use 0x011F.
2	DSPVersion	2	Shall be the version number of the DMTF specification whose DSP number is populated in the <code>DSPNumber</code> field. The format of the version number shall follow Table 10 — VersionNumberEntry definition .
4	VendorPayload	Variable	Shall be the actual payload data defined by the DMTF specification whose DSP number is populated in the <code>DSPNumber</code> field.

10.16 KEY_EXCHANGE request and KEY_EXCHANGE_RSP response messages

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.

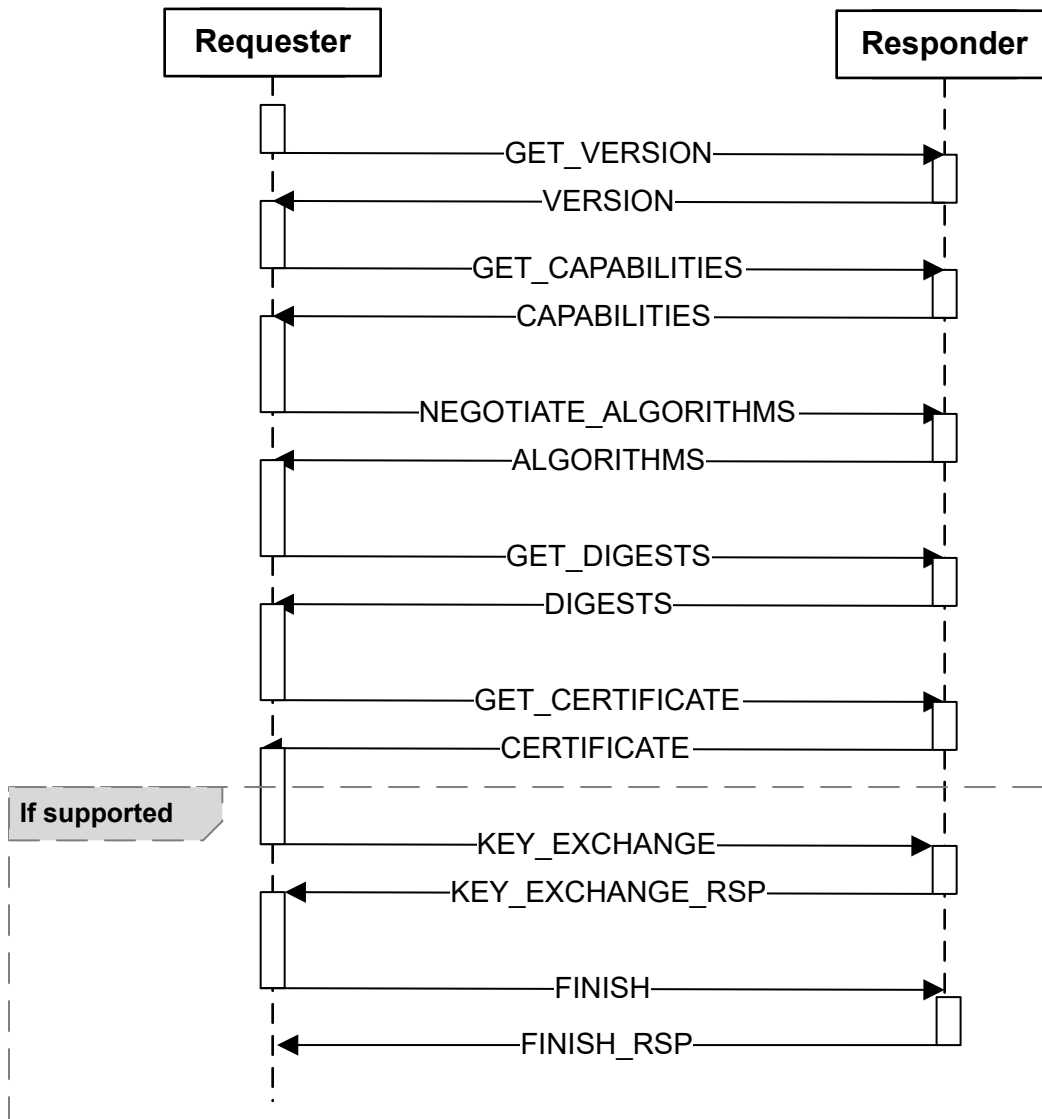
Table 69 — KEY_EXCHANGE request message format shows the KEY_EXCHANGE request message format, and Table 71 — Successful KEY_EXCHANGE_RSP response message format shows the KEY_EXCHANGE_RSP response message format. The handshake is completed by the successful exchange of the FINISH request and FINISH_RSP response messages presented in the next clause. The handshake depends on the tight coupling between these two request/response message pairs.

The Requester-Responder pair can support two modes of handshakes. If HANDSHAKE_IN_THE_CLEAR_CAP is set in both the Requester and the Responder, all SPDM messages exchanged during the Session Handshake Phase are sent in the clear (outside of a secure session). Otherwise both the Requester and the Responder use encryption and/or message authentication during the Session Handshake Phase using the Handshake secret derived at the completion of the KEY_EXCHANGE_RSP message for subsequent message communication until the completion of the FINISH_RSP message.

Figure 15 — Responder authentication key exchange example shows an example of a Responder authentication key exchange:

Figure 15 — Responder authentication key exchange example

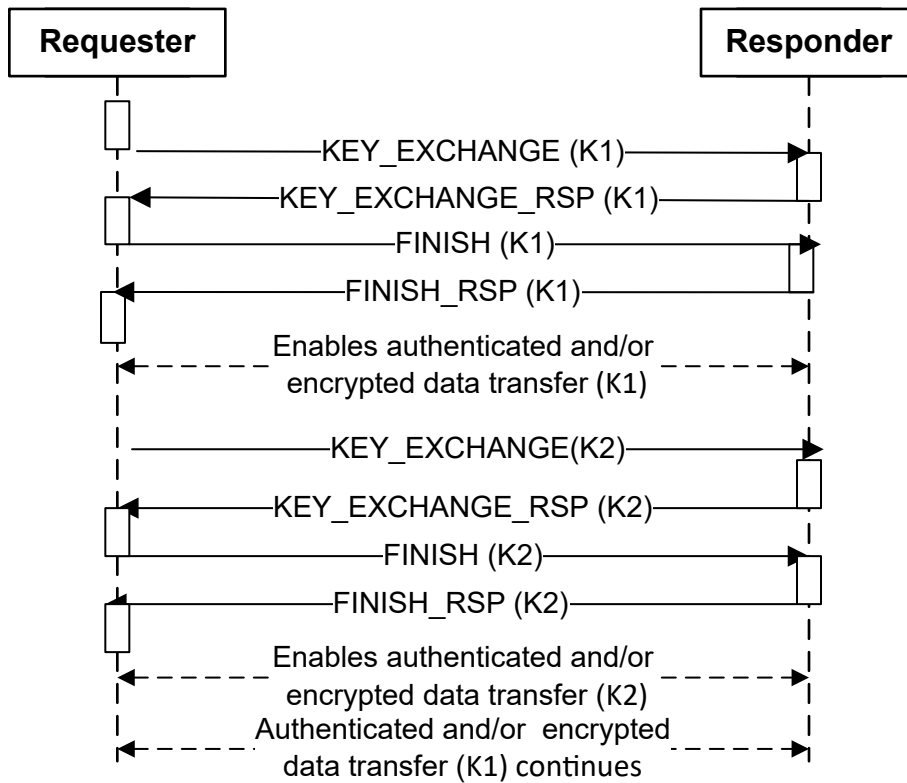
571



572 **Figure 16 — Responder authentication multiple key exchange example** shows an example of multiple sessions using two independent sets of root session keys that coexist at the same time. The specification does not require a specific temporal relationship between the second `KEY_EXCHANGE` request message and the first `FINISH_RSP` response message. However, to simplify implementation, a Responder might respond with an `ERROR` message of `ErrorCode=Busy` to the second `KEY_EXCHANGE` request message until the first `FINISH_RSP` response message is complete.

573 **Figure 16 — Responder authentication multiple key exchange example**

574



575 The handshake includes an ephemeral Diffie-Hellman (DHE) key exchange in which the Requester and Responder each generate an ephemeral (that is, temporary) Diffie-Hellman key pair and exchange the public keys of those key pairs in the `ExchangeData` fields of the `KEY_EXCHANGE` request message and `KEY_EXCHANGE_RSP` response message. The Responder generates a DHE secret by using the private key of the DHE key pair of the Responder and the public key of the DHE key pair of the Requester provided in the `KEY_EXCHANGE` request message. Similarly, the Requester generates a DHE secret by using the private key of the DHE key pair of the Requester and the public key of the DHE key pair of the Responder provided in the `KEY_EXCHANGE_RSP` response message. The DHE secrets are computed as specified in clause 7.4 of [RFC 8446](#). Assuming that the public keys were received correctly, both the Requester and Responder generate identical DHE secrets from which session secrets are generated.

576 Diffie-Hellman group parameters are determined by the DHE group in use, which is selected in the most recent `ALGORITHMS` response. The contents of the `ExchangeData` field are computed as specified in clause 4.2.8 of [RFC 8446](#). Specifically, if the DHE key exchange is based on finite-fields (FFDHE), the `ExchangeData` field in `KEY_EXCHANGE` and `KEY_EXCHANGE_RSP` shall contain the computed public value ($Y = g^X \text{ mod } p$) for the specified group (see [Table 17 — DHE structure](#) for group definitions) encoded as a big-endian integer and padded to the left with zeros to the size of p in bytes. If the key exchange is based on elliptic curves (ECDHE), the `ExchangeData` field in `KEY_EXCHANGE` and `KEY_EXCHANGE_RSP` shall contain the serialization of X and Y , which are the binary representations of the x and y values respectively in network byte order, padded on the left by zeros if necessary. The

size of each number representation occupies as many octets as are implied by the curve parameters selected. Specifically, X is [0: C - 1] and Y is [C : D - 1], where C and D are determined by the group (see [Table 17 — DHE structure](#)).

- 577 For SM2_P256 key exchange, the identifiers ID_A and ID_B that the [GB/T 32918.3-2016](#) specification defines are needed to derive the shared secret. If this algorithm is selected, the ID for the Requester (that is, ID_A) shall be the concatenation of "Requester-KEP-dmtf-spdm-v" and `SPDMversionString`. Likewise, the ID for the Responder (that is, ID_B) shall be the concatenation of "Responder-KEP-dmtf-spdm-v" and `SPDMversionString`.
- 578 A Requester should generate a new DHE key pair for each `KEY_EXCHANGE` request message that the Requester sends. A Responder should generate a new DHE key pair for each `KEY_EXCHANGE_RSP` response message that the Responder sends.

579 **Table 69 — KEY_EXCHANGE request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0xE4</code> = <code>KEY_EXCHANGE</code> . See Table 4 — SPDM request codes .
2	Param1	1	Shall be the type of measurement summary hash requested: <code>0x0</code> : No measurement summary hash requested. <code>0x1</code> : TCB measurements only. <code>0xFF</code> : All measurements. All other values reserved. If a Responder does not support measurements (<code>MEAS_CAP=00b</code> in its <code>CAPABILITIES</code> response), the Requester shall set this value to <code>0x0</code> .
3	Param2	1	Shall be the <code>SlotID</code> . Slot number of the Responder certificate chain that shall be used for authentication. If the public key of the Responder was provisioned to the Requester in a trusted environment, the value in this field shall be <code>0xFF</code> ; otherwise it shall be between 0 and 7 inclusive.
4	ReqSessionID	2	Shall be the two-byte Requester contribution to allow construction of a unique four-byte session ID between a Requester-Responder pair. The final session ID (<code>SessionID</code>) = <code>Concatenate(ReqSessionID, RspSessionID)</code> .
6	SessionPolicy	1	Shall be the session policy as Table 70 — Session policy defines.
7	Reserved	1	Reserved.
8	RandomData	32	Shall be the Requester-provided random data.

Byte offset	Field	Size (bytes)	Description
40	ExchangeData	D	Shall be the DHE public information generated by the Requester. If the DHE group selected in the most recent <code>ALGORITHMS</code> response is finite-field-based (FFDHE), the <code>ExchangeData</code> represents the computed public value. If the selected DHE group is elliptic-curve-based (ECDHE), the <code>ExchangeData</code> 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 group, as described in Table 23 — DHE structure .
40 + D	OpaqueDataLength	2	Shall be the size of the <code>OpaqueData</code> field that follows in bytes. The value should not be greater than 1024 bytes. Shall be 0 if no <code>OpaqueData</code> is provided.
42 + D	OpaqueData	OpaqueDataLength	If present, shall be the <code>OpaqueData</code> sent by the Requester. Used to indicate any parameters that the Requester wishes to pass to the Responder as part of key exchange. If present, this field shall conform to the selected opaque data format in <code>OtherParamsSelection</code> .

580 **Table 70 — Session policy**

Bit offset	Field	Description
0	TerminationPolicy	<p>This field specifies the behavior of the Responder when the Responder completes a runtime code or configuration update that affects the hardware or firmware measurement of the Responder. The Requester selects the value. If not set, the Responder shall terminate the session when the runtime update has taken effect. If set, the Responder shall decide whether to terminate or continue with the session based on its own policy. A policy example is one where the Responder terminates the session whenever an update to configuration or runtime code changes the security version of the firmware that manages SPDM sessions. The policy of the Responder is outside the scope of this specification.</p> <p>To terminate a session, the Responder shall either respond with an <code>ERROR</code> message of <code>ErrorCode=RequestResynch</code> to any SPDM request received within the session or silently discard any request received within the session until a <code>GET_VERSION</code> request is received.</p>
1	EventAllPolicy	<p>If set, the Responder shall subscribe the Requester to all events the Responder supports. Upon successfully entering the application phase of a session, the Responder may immediately send events.</p> <p>If <code>EVENT_CAP</code> is not set in <code>CAPABILITIES</code>, the Responder shall either respond with an <code>ERROR</code> message of <code>ErrorCode=InvalidRequest</code> or silently discard the request.</p>
[7:2]	Reserved	Reserved

581 **Table 71 — Successful KEY_EXCHANGE_RSP response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .

Byte offset	Field	Size (bytes)	Description
1	RequestResponseCode	1	Shall be <code>0x64</code> = <code>KEY_EXCHANGE_RSP</code> . See Table 5 — SPDM response codes .
2	Param1	1	Shall be HeartbeatPeriod. The value of this field shall be zero if Heartbeat is not supported by one of the endpoints. Otherwise, the value shall be in units of seconds. Zero is a legal value if Heartbeat is supported, and this means that a heartbeat is not desired on this session.
3	Param2	1	Reserved.
4	RspSessionID	2	Shall be the two-byte Responder contribution to allow construction of a unique four-byte session ID between a Requester-Responder pair. The final session ID = Concatenate(ReqSessionID, RspSessionID).
6	MutAuthRequested	1	<p>Bit 0. If set, the Responder is requesting to authenticate the Requester (Session-based mutual authentication) without using the encapsulated request flow.</p> <p>Bit 1. If set, Responder is requesting Session-based mutual authentication with the encapsulated request flow.</p> <p>Bit 2. If set, Responder is requesting Session-based mutual authentication with an implicit <code>GET_DIGESTS</code> request. The Responder and Requester shall follow the optimized encapsulated request flow.</p> <p>Bit [7:3]. Reserved.</p> <p>At most one bit of Bit 0, Bit 1, or Bit 2 shall be set.</p> <p>For encapsulated request flow and the optimized encapsulated request flow details, see the GET_ENCAPSULATED_REQUEST request and ENCAPSULATED_REQUEST response messages clause.</p>
7	SlotIDParam	1	<p>Bit [7:4]. Reserved.</p> <p>Bit [3:0]. Shall be the <code>SlotID</code>. Slot number of the Requester certificate chain that shall be used for mutual authentication, if <code>MutAuthRequested</code> Bit 0 is set. If the public key of the Requester was provisioned to the Responder through other means, the value in this field shall be <code>0xF</code>; otherwise it shall be between 0 and 7 inclusive. All other values reserved.</p> <p>For any other value of <code>MutAuthRequested</code>, this field shall be set to <code>0</code> and ignored by the Requester.</p>
8	RandomData	32	Shall be the Responder-provided random data.

Byte offset	Field	Size (bytes)	Description
40	ExchangeData	D	Shall be the DHE public information generated by the Responder. If the DHE group selected in the most recent <code>ALGORITHMS</code> response is finite-field-based (FFDHE), the <code>ExchangeData</code> represents the computed public value. If the selected DHE group is elliptic-curve-based (ECDHE), the <code>ExchangeData</code> represents the X and Y values in network byte order. Specifically, X is <code>[0: C - 1]</code> and Y is <code>[C : D - 1]</code> . In both cases the size of D (and C for ECDHE) is derived from the selected DHE group, as described in Table 23 — DHE structure .
40 + D	MeasurementSummaryHash	MSHLength = H or 0	<p>If the Responder does not support measurements (<code>MEAS_CAP=00b</code> in its <code>CAPABILITIES</code> response) or requested <code>Param1 = 0x0</code>, this field shall be absent.</p> <p>If the requested <code>Param1 = 0x1</code>, this field shall be the combined hash of measurements of all measurable components considered to be in the TCB required to generate this response, computed as <code>hash(Concatenate(MeasurementBlock[0], MeasurementBlock[1], ...))</code>, where <code>MeasurementBlock[x]</code> denotes a measurement of an element in the TCB and <code>hash</code> is the negotiated base hashing algorithm. Measurements are concatenated in ascending order based on their measurement index as Table 53 — Measurement block format describes.</p> <p>If the requested <code>Param1 = 0x1</code> and if there are no measurable components in the TCB required to generate this response, this field shall be <code>0</code>.</p> <p>If requested <code>Param1 = 0xFF</code>, this field shall be computed as <code>hash(Concatenate(MeasurementBlock[0], MeasurementBlock[1], ..., MeasurementBlock[n]))</code> of all supported measurements available in the measurement index range <code>0x01 - 0xFE</code>, concatenated in ascending index order. Indices with no associated measurements shall not be included in the hash calculation. See the Measurement index assignments clause.</p> <p>If the Responder supports both raw bit stream and digest representations for a given measurement index, the Responder shall use the digest form.</p> <p>This field shall be in hash byte order.</p>
40 + D + MSHLength	OpaqueDataLength	2	Shall be the size of the <code>OpaqueData</code> field that follows in bytes. The value should not be greater than 1024 bytes. Shall be <code>0</code> if no <code>OpaqueData</code> is provided.
42 + D + MSHLength	OpaqueData	OpaqueDataLength	If present, shall be the <code>OpaqueData</code> sent by the Responder. Used to indicate any parameters that the Responder wishes to pass to the Requester as part of key exchange. If present, this field shall conform to the selected opaque data format in <code>OtherParamsSelection</code> .

Byte offset	Field	Size (bytes)	Description
42 + D + MSHLength + OpaqueDataLength	Signature	SigLen	Shall be the <code>Signature</code> over the transcript. <code>SigLen</code> is the size of the asymmetric signing algorithm output the Responder selected via the last <code>ALGORITHMS</code> response message to the Requester. The Transcript for KEY_EXCHANGE_RSP signature defines the construction of the transcript.
42 + D + MSHLength + OpaqueDataLength + SigLen	ResponderVerifyData	H or 0	<p>Conditional field.</p> <p>If the Session Handshake Phase is encrypted and/or message authenticated, this field shall be of length H and shall equal the HMAC of the transcript hash, using <code>finished_key</code> as the secret key and using the negotiated hash algorithm as the hash function. The transcript hash shall be the hash of the transcript for <code>KEY_EXCHANGE_RSP</code> HMAC as Transcript for KEY_EXCHANGE_RSP HMAC shows. The <code>finished_key</code> shall be derived from the Response Direction Handshake Secret and is described in Finished_key derivation. HMAC is described in RFC 2104.</p> <p>If both the Requester and Responder set <code>HANDSHAKE_IN_THE_CLEAR_CAP</code> to 1, this field shall be absent.</p>

582 10.16.1 Session-based mutual authentication

583 Mutual authentication for `KEY_EXCHANGE` occurs in the session handshake phase of a session.

584 To perform authentication of a Requester, the Responder sets the appropriate bit in the `MutAuthRequested` field of the `KEY_EXCHANGE_RSP` message. When either Bit 1 or Bit 2 of `MutAuthRequested` are set, the encapsulated request flow or the optimized encapsulated request flow shall be used accordingly to enable the Responder to obtain the certificate chains and certificate chain digests of the Requester. For flow details and illustrations, see [GET_ENCAPSULATED_REQUEST request and ENCAPSULATED_REQUEST response messages](#).

585 When either bit 1 or bit 2 of `MutAuthRequested` are set, the only allowed messages in this phase of the session shall be `GET_DIGESTS`, `DIGESTS`, `GET_CERTIFICATE`, `CERTIFICATE`, and `ERROR`. If the Requester receives other requests during this flow, the Requester can respond with an `ERROR` message of `ErrorCode=UnexpectedRequest` and shall terminate the session.

586 If Bit 0 of `MutAuthRequested` is set, then mutual authentication shall be performed without exchanging any messages between `KEY_EXCHANGE_RSP` and `FINISH` request. This is useful for Responders that have obtained a Requester's certificate chains in a previous interaction.

587 10.16.1.1 Specify Requester certificate for session-based mutual authentication

588 The SPDM key exchange protocol is optimized to perform key exchange with the least number of messages exchanged. For Responder-only authentication and for mutual authentication where the Responder has obtained the certificate chains of the Requester in a previous interaction, key exchange is carried out with two request/response

message pairs (`KEY_EXCHANGE` and `KEY_EXCHANGE_RSP` ; `FINISH` and `FINISH_RSP`). In other cases where mutual authentication is desired, additional [encapsulated messages](#) are exchanged between `KEY_EXCHANGE_RSP` and `FINISH` to enable the Responder to obtain the certificate chains and certificate chain digests of the Requester. However, in all cases the certificate chain (or raw public key) the Requester should authenticate against is specified by the Responder via the `SlotID` field in `KEY_EXCHANGE_RSP` , which precedes the aforementioned encapsulated messages. This means that a Responder has no way of knowing in advance which `SlotID` value to use when authenticating a Requester whose certificates it has not obtained in a previous interaction, other than the default (Slot 0).

589 To address this case, the Responder explicitly designates the certificate chain to be used via the final `ENCAPSULATED_RESPONSE_ACK` request issued inside the encapsulated request flow. Specifically, if either Bit 1 or 2 in `MutAuthRequested` is set to 1, the Responder shall use an `ENCAPSULATED_RESPONSE_ACK` request with `Param2` = 0x02 and a 1-byte-long `Encapsulated Request` field containing the `SlotID` value. The Requester shall use the certificate chain corresponding to the slot specified in the `Encapsulated Request` field.

590 If Bit 0 of `MutAuthRequested` is set, then no encapsulated messages are exchanged after `KEY_EXCHANGE_RSP` and the certificate chain of the Requester is determined by the value of `SlotIDParam` in `KEY_EXCHANGE_RSP` .

591 10.17 FINISH request and FINISH_RSP response messages

592 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_RSP` response messages is to provide key confirmation, bind the identity of each party to the exchanged keys and protect the entire handshake against manipulation by an active attacker. Upon receiving a `FINISH` request, the Responder shall ensure the session and the corresponding session ID were created through a `KEY_EXCHANGE` request and corresponding `KEY_EXCHANGE_RSP` response. [Table 72 — FINISH request message format](#) shows the `FINISH` request message format and [Table 73 — Successful FINISH_RSP response message format](#) shows the `FINISH_RSP` response message format.

593 **Table 72 — FINISH request message format**

Byte offset	Field	Size (bytes)	Description
0	<code>SPDMVersion</code>	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	<code>RequestResponseCode</code>	1	Shall be 0xE5 = <code>FINISH</code> . See Table 4 — SPDM request codes .
2	<code>Param1</code>	1	Bit 0. If set, the Signature field is included. This bit shall be set when Session-based mutual authentication occurs. All other bits reserved.
3	<code>Param2</code>	1	Shall be the <code>SlotID</code> . Only valid if <code>Param1</code> = 0x01 , otherwise reserved. Slot number of the Responder certificate chain that shall be used for authentication. If the public key of the Responder was provisioned to the Requester in a trusted environment, the value in this field shall be 0xFF ; otherwise it shall be between 0 and 7 inclusive.

Byte offset	Field	Size (bytes)	Description
4	Signature	SigLen	Shall be the <code>Signature</code> over the transcript. <code>SigLen</code> is the size of the asymmetric signing algorithm (<code>ReqBaseAsymAlg</code>) output the Responder selected via the last <code>ALGORITHMS</code> response message to the Requester. If <code>Param1 = 0x00</code> , <code>SigLen</code> is zero and this field shall be absent. Transcript for FINISH signature, mutual authentication defines the construction of the transcript, signature generation, and verification.
4 + SigLen	RequesterVerifyData	H	Shall be an HMAC of the transcript hash using the <code>finished_key</code> as the secret key and using the negotiated hash algorithm as the hash function. For mutual authentication, the transcript hash shall be the hash of the transcript for <code>FINISH</code> HMAC, mutual authentication as the transcript for FINISH HMAC, mutual authentication shows. Otherwise, it shall be the hash of the transcript for <code>FINISH</code> HMAC, Responder-only authentication as the transcript for FINISH HMAC, Responder-only authentication shows. The <code>finished_key</code> shall be derived from Request Direction Handshake Secret and is described in Finished_key derivation . HMAC is described in RFC 2104 .

594 If the handshake is performed in the clear (that is, if `HANDSHAKE_IN_THE_CLEAR_CAP = 1` for both Requester and Responder), and if either Bit 1 or Bit 2 in `KEY_EXCHANGE_RSP` . `MutAuthRequested` is set, then upon receiving `FINISH` the Responder shall confirm that the value in `FINISH` . `Param2` matches the value that the Responder specified in the final `ENCAPSULATED_RESPONSE_ACK` . `EncapsulatedRequest` .

595 **Table 73 — Successful FINISH_RSP response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x65 = FINISH_RSP</code> . See Table 5 — SPDM response codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	ResponderVerifyData	H or 0	Conditional field. If the Session Handshake Phase is encrypted and/or message authenticated (that is, if either the Requester or the Responder set <code>HANDSHAKE_IN_THE_CLEAR_CAP</code> to 0), this field shall be absent. If both the Requester and Responder support <code>HANDSHAKE_IN_THE_CLEAR_CAP</code> field, this field shall be of length H and shall equal the HMAC of the transcript hash using <code>finished_key</code> as the secret key and using the negotiated hash algorithm as the hash function. For Session-based mutual authentication, the transcript hash shall be the hash of the transcript for <code>FINISH_RSP</code> HMAC, as the transcript for FINISH_RSP HMAC, mutual authentication shows. Otherwise, the transcript hash shall be the hash of the transcript for <code>FINISH_RSP</code> HMAC, Responder-only authentication as the transcript for FINISH_RSP HMAC, Responder-only authentication shows. The <code>finished_key</code> shall be derived from Response Direction Handshake Secret and is described in Finished_key derivation . HMAC is described in RFC 2104 .

596 10.17.1 Transcript and transcript hash calculation rules for KEY_EXCHANGE

597 [Transcript for KEY_EXCHANGE_RSP signature](#) shows the transcript for the `KEY_EXCHANGE_RSP` signature:

598 Transcript for KEY_EXCHANGE_RSP signature

1. `VCA`
2. `[DIGESTS].*` (if issued and `MULTI_KEY_CONN_RSP` is true).
3. Hash of the specified certificate chain in DER format (that is, `Param2` of `KEY_EXCHANGE`) or hash of the public key in its provisioned format, if a certificate is not used.
4. `[KEY_EXCHANGE].*`
5. `[KEY_EXCHANGE_RSP].*` except the `Signature` and `ResponderVerifyData` fields.

599 The Responder shall generate the `KEY_EXCHANGE_RSP` signature from:

```
SPDMsign(PrivKey, transcript, "key_exchange_rsp signing");
```

600 where

- `SPDMsign` is described by the [Signature generation](#) clause.
- `PrivKey` shall be the private key of the Responder associated with the leaf certificate stored in `SlotID` in `KEY_EXCHANGE`. If the public key of the Responder was provisioned to the Requester, then `PrivKey` shall be the associated private key.
- `transcript` shall be the concatenation of the messages for a `KEY_EXCHANGE_RSP` signature.

601 The leaf certificate of the Responder shall be the one indicated by `SlotID` in `Param2` of `KEY_EXCHANGE` request.

602 Likewise, the Requester shall verify the `KEY_EXCHANGE_RSP` signature using `SPDMsignatureVerify(PubKey, signature, transcript, "key_exchange_rsp signing")`, where `transcript` is the concatenation of the messages for a `KEY_EXCHANGE_RSP` signature and `PubKey` is the public key of the leaf certificate of the Responder. The leaf certificate of the Responder shall be the one indicated by `SlotID` in `Param2` of `KEY_EXCHANGE` request. `SPDMsignatureVerify` is described in [Signature verification](#). A successful verification shall be when `SPDMsignatureVerify` returns `success`.

603 [Transcript for KEY_EXCHANGE_RSP HMAC](#) shows the transcript for `KEY_EXCHANGE_RSP` HMAC:

604 Transcript for KEY_EXCHANGE_RSP HMAC

1. `VCA`
2. `[DIGESTS].*` (if issued and `MULTI_KEY_CONN_RSP` is true).
3. Hash of the specified certificate chain in DER format (that is, `Param2` of `KEY_EXCHANGE`) or hash of the

public key in its provisioned format, if a certificate is not used.

4. [KEY_EXCHANGE] . *
5. [KEY_EXCHANGE_RSP] . * except the ResponderVerifyData field.

605 [Transcript for FINISH signature, mutual authentication](#) shows the transcript for the FINISH signature with mutual authentication:

606 **Transcript for FINISH signature, mutual authentication**

1. VCA
2. [DIGESTS] . * (if issued and MULTI_KEY_CONN_RSP is true).
3. Hash of the specified certificate chain in DER format (that is, Param2 of KEY_EXCHANGE) or hash of the public key in its provisioned format, if a certificate is not used.
4. [KEY_EXCHANGE] . *
5. [KEY_EXCHANGE_RSP] . *
6. [DIGESTS] . * (if encapsulated DIGEST is issued and MULTI_KEY_CONN_REQ is true).
7. Hash of the specified certificate chain in DER format (that is, Param2 of FINISH) or hash of the public key in its provisioned format, if a certificate is not used.
8. [FINISH] . SPDM Header Fields

607 The Requester shall generate the FINISH signature from SPDMsign(PrivKey, transcript, "finish signing"), where transcript is the concatenation of the messages for FINISH signature and the PrivKey is the private key of the leaf certificate of the Requester. The leaf certificate of the Requester shall be the one indicated in SlotID in Param2 of FINISH request. SPDMsign is described in [Signature generation](#).

608 Likewise, the Responder shall verify the FINISH signature using SPDMsignatureVerify(PubKey, signature, transcript, "finish signing"), where transcript is the concatenation of the messages for a FINISH signature and the PubKey is the public key of the leaf certificate of the Requester. The leaf certificate of the Requester shall be the one indicated in SlotID in Param2 of the FINISH request. SPDMsignatureVerify is described in [Signature verification](#). A successful verification is when SPDMsignatureVerify returns success .

609 [Transcript for FINISH HMAC, Responder-only authentication](#) shows the transcript for FINISH HMAC with Responder-only authentication:

610 **Transcript for FINISH HMAC, Responder-only authentication**

1. VCA
2. [DIGESTS] . * (if issued and MULTI_KEY_CONN_RSP is true).
3. Hash of the specified certificate chain in DER format (that is, Param2 of KEY_EXCHANGE) or hash of the public key in its provisioned format, if a certificate is not used.
4. [KEY_EXCHANGE] . *
5. [KEY_EXCHANGE_RSP] . *

6. [FINISH] . SPDM Header Fields

611 [Transcript for FINISH HMAC, mutual authentication](#) shows the transcript for [FINISH] HMAC with mutual authentication:

612 **Transcript for [FINISH] HMAC, mutual authentication**

1. VCA
2. [DIGESTS] . * (if issued and MULTI_KEY_CONN_RSP is true).
3. Hash of the specified certificate chain in DER format (that is, Param2 of KEY_EXCHANGE) or hash of the public key in its provisioned format, if a certificate is not used.
4. [KEY_EXCHANGE] . *
5. [KEY_EXCHANGE_RSP] . *
6. [DIGESTS] . * (if encapsulated DIGEST is issued and MULTI_KEY_CONN_REQ is true).
7. Hash of the specified certificate chain in DER format (that is, Param2 of FINISH) or hash of the public key in its provisioned format, if a certificate is not used.
8. [FINISH] . SPDM Header Fields
9. [FINISH] . Signature

613 [Transcript for FINISH_RSP HMAC, Responder-only authentication](#) shows the transcript for [FINISH_RSP] HMAC with Responder-only authentication:

614 **Transcript for [FINISH_RSP] HMAC, Responder-only authentication**

1. VCA
2. [DIGESTS] . * (if issued and MULTI_KEY_CONN_RSP is true).
3. Hash of the specified certificate chain in DER format (that is, Param2 of KEY_EXCHANGE) or hash of the public key in its provisioned format, if a certificate is not used.
4. [KEY_EXCHANGE] . *
5. [KEY_EXCHANGE_RSP] . *
6. [FINISH] . *
7. [FINISH_RSP] . SPDM Header fields

615 [Transcript for FINISH_RSP HMAC, mutual authentication](#) shows the transcript for [FINISH_RSP] HMAC with mutual authentication:

616 **Transcript for [FINISH_RSP] HMAC, mutual authentication**

1. VCA
2. [DIGESTS] . * (if issued and MULTI_KEY_CONN_RSP is true).
3. Hash of the specified certificate chain in DER format (that is, Param2 of KEY_EXCHANGE) or hash of the public key in its provisioned format, if a certificate is not used.

4. [KEY_EXCHANGE] . *
5. [KEY_EXCHANGE_RSP] . *
6. [DIGESTS] . * (if encapsulated DIGEST is issued and MULTI_KEY_CONN_REQ is true).
7. Hash of the specified certificate chain in DER format (that is, Param2 of FINISH) or hash of the public key in its provisioned format, if a certificate is not used.
8. [FINISH] . *
9. [FINISH_RSP] . SPDM Header fields

617 When multiple session keys are being established between the same Requester-Responder pair, the Signature over the transcript during FINISH request is computed using only the corresponding KEY_EXCHANGE , KEY_EXCHANGE_RSP , and FINISH request parameters.

618 For additional rules, see [general ordering rules](#).

619 10.18 PSK_EXCHANGE request and PSK_EXCHANGE_RSP response messages

620 The Pre-Shared Key (PSK) key exchange scheme provides an option for a Requester and a Responder to perform 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 session key establishment even if asymmetric-key cryptography is supported.

621 This option requires the Requester and Responder to have prior knowledge of a common PSK before the handshake. Essentially, the PSK serves as a mutual authentication credential and as the base of session key establishment. As such, only the two endpoints and potentially a trusted third party that provisions the PSK to the two endpoints know the value of the PSK. For these same reasons, the HANDSHAKE_IN_THE_CLEAR_CAP is not applicable in a PSK key exchange. Thus, for PSK-based session establishment, both the Responder and the Requester shall ignore the HANDSHAKE_IN_THE_CLEAR_CAP bit.

622 A Requester can pair with multiple Responders. Likewise, a Responder can pair with multiple Requesters. A Requester-Responder pair can be provisioned with one or more PSKs. An endpoint can act as a Requester to one device and simultaneously a Responder to another device. If both endpoints can act as Requester or Responder, then the endpoints shall use different PSKs for each role. 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.

623 The PSK can be provisioned in a trusted environment, for example, during the secure manufacturing process. In an untrusted environment, the PSK can be agreed upon between the two endpoints using a secure protocol. The mechanism for PSK provisioning is outside the scope of this specification. The size of the provisioned PSK is determined by the security strength requirements of the application, but it should be at least 128 bits. It is recommended to be at least 256 bits in order to resist dictionary attacks, particularly when the Requester and Responder cannot both contribute sufficient entropy during the exchange.

Two message pairs are defined for this option:

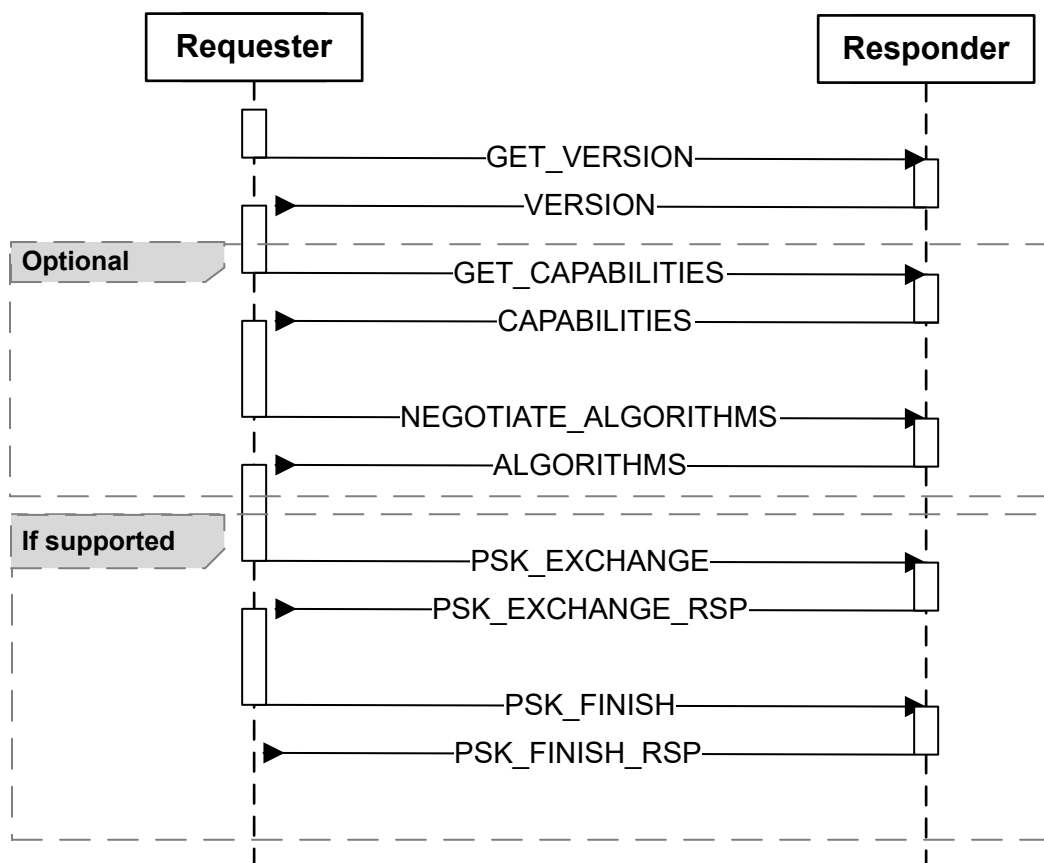
- PSK_EXCHANGE / PSK_EXCHANGE_RSP
- PSK_FINISH / PSK_FINISH_RSP

The PSK_EXCHANGE message carries three responsibilities:

1. Prompts the Responder to retrieve the specific PSK.
2. Exchanges contextual information 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.

Figure 17 — PSK_EXCHANGE: Example shows an example of the PSK_EXCHANGE message:

Figure 17 — PSK_EXCHANGE: Example



629 **Table 74 — PSK_EXCHANGE request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0xE6</code> = <code>PSK_EXCHANGE</code> . See Table 4 — SPDM request codes .
2	Param1	1	Shall be the type of measurement summary hash requested: <code>0x0</code> : No measurement summary hash requested. <code>0x1</code> : TCB measurements only. <code>0xFF</code> : All measurements. All other values reserved. If a Responder does not support measurements (<code>MEAS_CAP=00b</code> in its <code>CAPABILITIES</code> response), the Requester shall set this value to <code>0x0</code> .
3	Param2	1	Shall be the session policy. See Table 70 — Session policy .
4	ReqSessionID	2	Shall be the two-byte Requester contribution to allow construction of a unique four-byte session ID between a Requester-Responder pair. The final session ID = Concatenate(ReqSessionID, RspSessionID).
6	P	2	Shall be the length of <code>PSKHint</code> in bytes.
8	R	2	Shall be the length of <code>RequesterContext</code> in bytes.
10	OpaqueDataLength	2	Shall be the size of the <code>OpaqueData</code> field that follows in bytes. The value should not be greater than 1024 bytes. Shall be <code>0</code> if no <code>OpaqueData</code> is provided.
12	PSKHint	P	Shall be the information required by the Responder to retrieve the PSK. Optional.
12 + P	RequesterContext	R	Shall be the context of the Requester. Shall include a nonce or non-repeating counter of at least 32 bytes and, optionally, relevant information contributed by the Requester.
12 + P + R	OpaqueData	OpaqueDataLength	Optional. If present, the <code>OpaqueData</code> sent by the Requester is used to indicate any parameters that the Requester wishes to pass to the Responder as part of PSK-based key exchange. If present, this field shall conform to the selected opaque data format in OtherParamsSelection .

630 The field `PSKHint` is optional. It is 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 `PSKHint` as an identifier to specify which PSK will be used in this particular session.
- The Responder does not store the actual value of the PSK but can derive the PSK using `PSKHint`. For example, if the Responder has an immutable UDS (Unique Device Secret) in fuses, then during provisioning a PSK can be

derived from the UDS (or a derivative value) and a non-secret salt known by the Requester. During session key establishment, the salt value is sent to the Responder in `PSKHint` of `PSK_EXCHANGE`. This mechanism allows the Responder to support any number of PSKs without consuming secure storage.

631 The `RequesterContext` is the contribution of the Requester to session key derivation. It shall contain a nonce or non-repeating counter to ensure that the derived session keys are ephemeral to mitigate against replay attacks. If a non-repeating counter is used, the counter shall not be reset for the lifetime of the device. The `RequesterContext` can also contain other information from the Requester.

632 Upon receiving a `PSK_EXCHANGE` request, the Responder:

1. Generates PSK from `PSKHint`, if necessary.
2. Generates `ResponderContext`, if supported.
3. Derives the `finished_key` of the Responder by following the [key schedule](#).
4. Constructs the `PSK_EXCHANGE_RSP` response message and sends it to the Requester.

633 **Table 75 — PSK_EXCHANGE_RSP response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x66 = PSK_EXCHANGE_RSP</code> . See Table 5 — SPDM response codes .
2	Param1	1	Shall be HeartbeatPeriod. The value of this field shall be zero if Heartbeat is not supported by one of the endpoints. Otherwise, the value shall be in units of seconds. Zero is a legal value if Heartbeat is supported, and this means that a heartbeat is not desired on this session.
3	Param2	1	Reserved.
4	RspSessionID	2	Shall be the two-byte Responder contribution to allow construction of a unique four-byte session ID between a Requester-Responder pair. The final session ID (<code>SessionID</code>) = <code>Concatenate(ReqSessionID, RspSessionID)</code> .
6	Reserved	2	Reserved.
8	Q	2	Shall be the length of <code>ResponderContext</code> in bytes.
10	OpaqueDataLength	2	Shall be the size of the <code>OpaqueData</code> field that follows in bytes. The value should not be greater than 1024 bytes. Shall be <code>0</code> if no <code>OpaqueData</code> is provided.

Byte offset	Field	Size (bytes)	Description
12	MeasurementSummaryHash	MSHLength = H or 0	<p>If the Responder does not support measurements (<code>MEAS_CAP=00b</code> in its <code>CAPABILITIES</code> response) or requested <code>Param1 = 0x0</code>, this field shall be absent.</p> <p>If the requested <code>Param1 = 0x1</code>, this field shall be the combined hash of measurements of all measurable components considered to be in the TCB required to generate this response, computed as <code>hash(Concatenate(MeasurementBlock[0], MeasurementBlock[1], ...))</code>, where <code>MeasurementBlock[x]</code> denotes a measurement of an element in the TCB and <code>hash</code> is the negotiated base hashing algorithm. Measurements are concatenated in ascending order based on their measurement index as Table 53 — Measurement block format describes.</p> <p>If the requested <code>Param1 = 0x1</code> and if there are no measurable components in the TCB required to generate this response, this field shall be <code>0</code>.</p> <p>If requested <code>Param1 = 0xFF</code>, this field shall be computed as <code>hash(Concatenate(MeasurementBlock[0], MeasurementBlock[1], ..., MeasurementBlock[n]))</code> of all supported measurements available in the measurement index range <code>0x01 - 0xFE</code>, concatenated in ascending index order. Indices with no associated measurements shall not be included in the hash calculation. See the Measurement index assignments clause.</p> <p>If the Responder supports both raw bit stream and digest representations for a given measurement index, the Responder shall use the digest form.</p> <p>This field shall be in hash byte order.</p>
12 + MSHLength	ResponderContext	Q	Shall be the context of the Responder. Optional. If present, shall include a nonce and/or information contributed by the Responder.
12 + MSHLength + Q	OpaqueData	OpaqueDataLength	Optional. If present, the <code>OpaqueData</code> sent by the Responder is used to indicate any parameters that the Responder wishes to pass to the Requester as part of PSK-based key exchange. If present, this field shall conform to the selected opaque data format in <code>OtherParamsSelection</code> .
12 + MSHLength + Q + OpaqueDataLength	ResponderVerifyData	H	Shall be the data to be verified by the Requester using the <code>finished_key</code> of the Responder.

634 The `ResponderContext` is the contribution of the Responder to session key derivation. It should contain a nonce or non-repeating counter and other information from the Responder. If a non-repeating counter is used, the counter shall not be reset for the lifetime of the device. Because the Responder can be a constrained device that cannot

generate a nonce, `ResponderContext` is optional. However, the Responder is required to use `ResponderContext` if it can generate a nonce.

635 Note that the nonce in `ResponderContext` is critical for anti-replay. If a nonce is not present in `ResponderContext`, then the Responder is not challenging the Requester for real-time knowledge of the PSK. Such a session is subject to replay attacks—that is, a person-in-the-middle attacker could record and replay prior `PSK_EXCHANGE` and `PSK_FINISH` messages and set up a session with the Responder. But the bogus session would not leak secrets, so long as the PSK and session keys of the prior replayed session are not compromised.

636 If `ResponderContext` is absent, such as when `PSK_CAP` in the `CAPABILITIES` of the Responder is `01b`, the Requester shall not send `PSK_FINISH`, because the session keys are solely determined by the Requester and the Session immediately enters the Application Phase. If and only if the `ResponderContext` is present in the response, such as when `PSK_CAP` in the `CAPABILITIES` of the Responder is `10b`, the Requester shall send `PSK_FINISH` with `RequesterVerifyData` to prove that it has derived correct session keys.

637 To calculate `ResponderVerifyData`, the Responder calculates an HMAC. The HMAC key is the `finished_key` of the Responder. The data is the hash of the concatenation of all messages sent up to this point between the Requester and the Responder. For messages that are encrypted, the plaintext messages are used in calculating the hash.

1. `[GET_VERSION].*`
2. `[VERSION].*`
3. `[GET_CAPABILITIES].*` (if issued)
4. `[CAPABILITIES].*` (if issued)
5. `[NEGOTIATE_ALGORITHMS].*` (if issued)
6. `[ALGORITHMS].*` (if issued)
7. `[PSK_EXCHANGE].*`
8. `[PSK_EXCHANGE_RSP].*` except the `ResponderVerifyData` field

638 Note that, even if `CERTIFICATE` and Responder-signed response messages (such as `CHALLENGE_AUTH`) were issued, these messages would not be included in the data for calculating `ResponderVerifyData`. In other words, the identity of the signer of the response messages is not bound to the identity of the sender of `PSK_EXCHANGE_RSP`. Therefore, to mitigate Responder identity impersonation, if the Requester has received a response with a signature and if there is no cryptographic binding between the signer of the Responder-signed response and the sender of `PSK_EXCHANGE_RSP`, then the Requester should not issue `PSK_EXCHANGE`. The method of cryptographic binding between the signer of the Responder-signed response and the sender of `PSK_EXCHANGE_RSP` is outside the scope of this specification.

639 Upon receiving `PSK_EXCHANGE_RSP`, the Requester:

1. Derives the `finished_key` of the Responder by following the [key schedule](#).
2. Verifies `ResponderVerifyData` by calculating the HMAC in the same manner as the Responder. If verification fails, the Requester terminates the session.
3. If the Responder contributes to session key derivation, such as when the `ResponderContext` field is present in the `PSK_EXCHANGE_RSP` response, it constructs the `PSK_FINISH` request and sends it to the

Responder.

640 If a successful `PSK_EXCHANGE_RSP` has been received by the Requester, and the `PSK_CAP` of the Responder is `10b`, and the `ResponderContext` field is present in the `PSK_EXCHANGE_RSP` response then, for the session ID created by the `PSK_EXCHANGE` and `PSK_EXCHANGE_RSP` messages, the next request shall be `PSK_FINISH`.

641 10.19 PSK_FINISH request and PSK_FINISH_RSP response messages

642 These messages shall complete the mutually-authenticated handshake between Requester and Responder initiated by a `PSK_EXCHANGE` request. The `PSK_FINISH` request proves to the Responder that the Requester knows the PSK and has derived the correct session keys. This is achieved by an HMAC value calculated with the `finished_key` of the Requester and messages of this session. The Requester shall send `PSK_FINISH` only if `ResponderContext` is present in `PSK_EXCHANGE_RSP`. Upon receiving a `PSK_FINISH` request, the Responder shall ensure the session and the corresponding session ID were created through a `PSK_EXCHANGE` request and corresponding `PSK_EXCHANGE_RSP` response.

643 [Table 76 — PSK_FINISH request message format](#) describes the `PSK_FINISH` request message format:

644 **Table 76 — PSK_FINISH request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0xE7</code> = <code>PSK_FINISH</code> . See Table 4 — SPDM request codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	RequesterVerifyData	H	Shall be the data to be verified by the Responder using the <code>finished_key</code> of the Requester.

645 To calculate `RequesterVerifyData`, the Requester calculates an HMAC. The key is the `finished_key` of the Requester, as described in the [Key schedule](#) clause. The data is the hash of the concatenation of all messages sent so far between the Requester and the Responder. For messages that are encrypted, the plaintext messages are used in calculating the hash.

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

- 7. [PSK_EXCHANGE].*
- 8. [PSK_EXCHANGE_RSP].*
- 9. [PSK_FINISH].* except the RequesterVerifyData field

646 For additional rules, see [general ordering rules](#).

647 Upon receiving the PSK_FINISH request, the Responder derives the finished_key of the Requester and calculates the HMAC independently in the same manner and verifies that the result matches RequesterVerifyData. If verification is successful, the Responder constructs the PSK_FINISH_RSP response and sends it to the Requester. Otherwise, the Responder sends the Requester an ERROR message of ErrorCode=InvalidRequest.

648 [Table 77 — Successful PSK_FINISH_RSP response message format](#) describes the successful PSK_FINISH_RSP response message format:

649 **Table 77 — Successful PSK_FINISH_RSP response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in SPDM version .
1	RequestResponseCode	1	Shall be 0x67 = PSK_FINISH_RSP. See Table 5 — SPDM response codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.

650 10.20 HEARTBEAT request and HEARTBEAT_ACK response messages

651 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 the KEY_EXCHANGE_RSP or PSK_EXCHANGE_RSP response messages. The Responder shall terminate the session if session traffic is not received for two successive HeartbeatPeriod s. Likewise, the Requester shall terminate the session if session traffic, including ERROR responses, is not received for two successive HeartbeatPeriod s. Session traffic includes encrypted data at the transport layer. How an SPDM endpoint is informed of encrypted data at the transport layer is outside the scope of this specification. The Requester can retry HEARTBEAT requests.

652 The timer for the Heartbeat period shall start at either the transmission (for Responders) or the reception (for Requesters) of either the FINISH_RSP or the PSK_FINISH_RSP response messages. When determining the value of HeartbeatPeriod, the Responder should ensure this value is sufficiently greater than T1.

653 For session termination details, see [session termination phase](#).

654 [Table 78 — HEARTBEAT request message format](#) describes the message format.

Table 78 — HEARTBEAT request message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0xE8</code> = HEARTBEAT request. See Table 4 — SPDM request codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.

[Table 79 — HEARTBEAT_ACK response message format](#) describes the format for the Heartbeat Response.

Table 79 — HEARTBEAT_ACK response message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x68</code> = HEARTBEAT_ACK response. See Table 5 — SPDM response codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.

10.20.1 Heartbeat additional information

The transport layer might allow the HEARTBEAT request to be sent from the Responder to the Requester. This is recommended for transports capable of asynchronous bidirectional communication.

10.21 KEY_UPDATE request and KEY_UPDATE_ACK response messages

This request shall be used to update session keys. There are many reasons for doing this, but an important one is when the per-record nonce will soon reach its maximum value and roll over. The KEY_UPDATE request can also be issued by the Responder using the GET_ENCAPSULATED_REQUEST mechanism. A KEY_UPDATE request shall perform the operation given in Param1 and defined in [Table 82 — KEY_UPDATE operations](#). 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 at a time. Until the session key update synchronization successfully completes, subsequent KEY_UPDATE requests for the same direction shall be considered a retry of the original KEY_UPDATE request.

[Table 80 — KEY_UPDATE request message format](#) describes the KEY_UPDATE request message format:

Table 80 — KEY_UPDATE request message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0xE9</code> = <code>KEY_UPDATE</code> Request. See Table 4 — SPDM request codes .
2	Param1	1	Shall indicate the key operation. See Table 82 — KEY_UPDATE operations .
3	Param2	1	Shall be the requesting SPDM endpoint assigned tag. This field shall contain a unique number to aid the responding SPDM endpoint in differentiating between the original and any retry requests. A retry request shall contain the same tag number as the original.

664 [Table 81 — KEY_UPDATE_ACK response message format](#) describes the `KEY_UPDATE_ACK` response message format:

665 **Table 81 — KEY_UPDATE_ACK response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x69</code> = <code>KEY_UPDATE_ACK</code> response. See Table 5 — SPDM response codes .
2	Param1	1	Shall indicate the key operation. This field shall reflect the Key Operation field of the request. See Table 82 — KEY_UPDATE operations
3	Param2	1	Shall be the tag. This field shall reflect the Tag number (<code>Param2</code>) from the <code>KEY_UPDATE</code> request.

666 [Table 82 — KEY_UPDATE operations](#) describes the `KEY_UPDATE` operations:

667 **Table 82 — KEY_UPDATE operations**

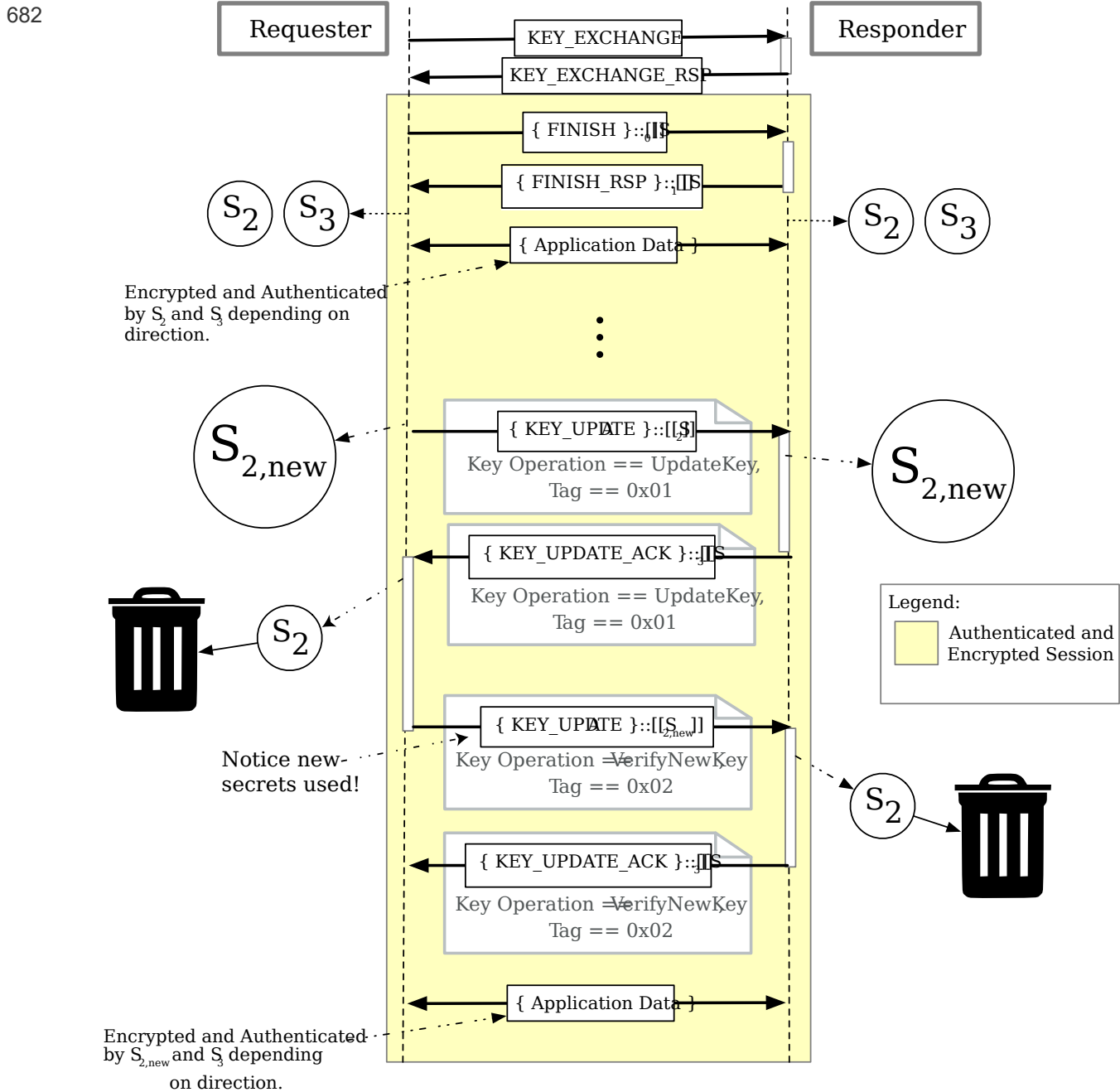
Value	Operation	Description
0	Reserved	Reserved.
1	UpdateKey	Shall update only the single-direction key associated with the direction of the request.
2	UpdateAllKeys	Shall update the keys for both directions.
3	VerifyNewKey	Shall ensure that the key update is successful and that old keys can be safely discarded.
4 - 255	Reserved	Reserved.

668 10.21.1 Session key update synchronization

669 In the key update process, to clarify, the term "sender" means the SPDM endpoint that issued the `KEY_UPDATE` request, and the term "receiver" means the SPDM endpoint that received the `KEY_UPDATE` request. To ensure the key

update process is seamless while still allowing the transmission and reception of records, both sender and receiver shall follow the prescribed method described in this clause.

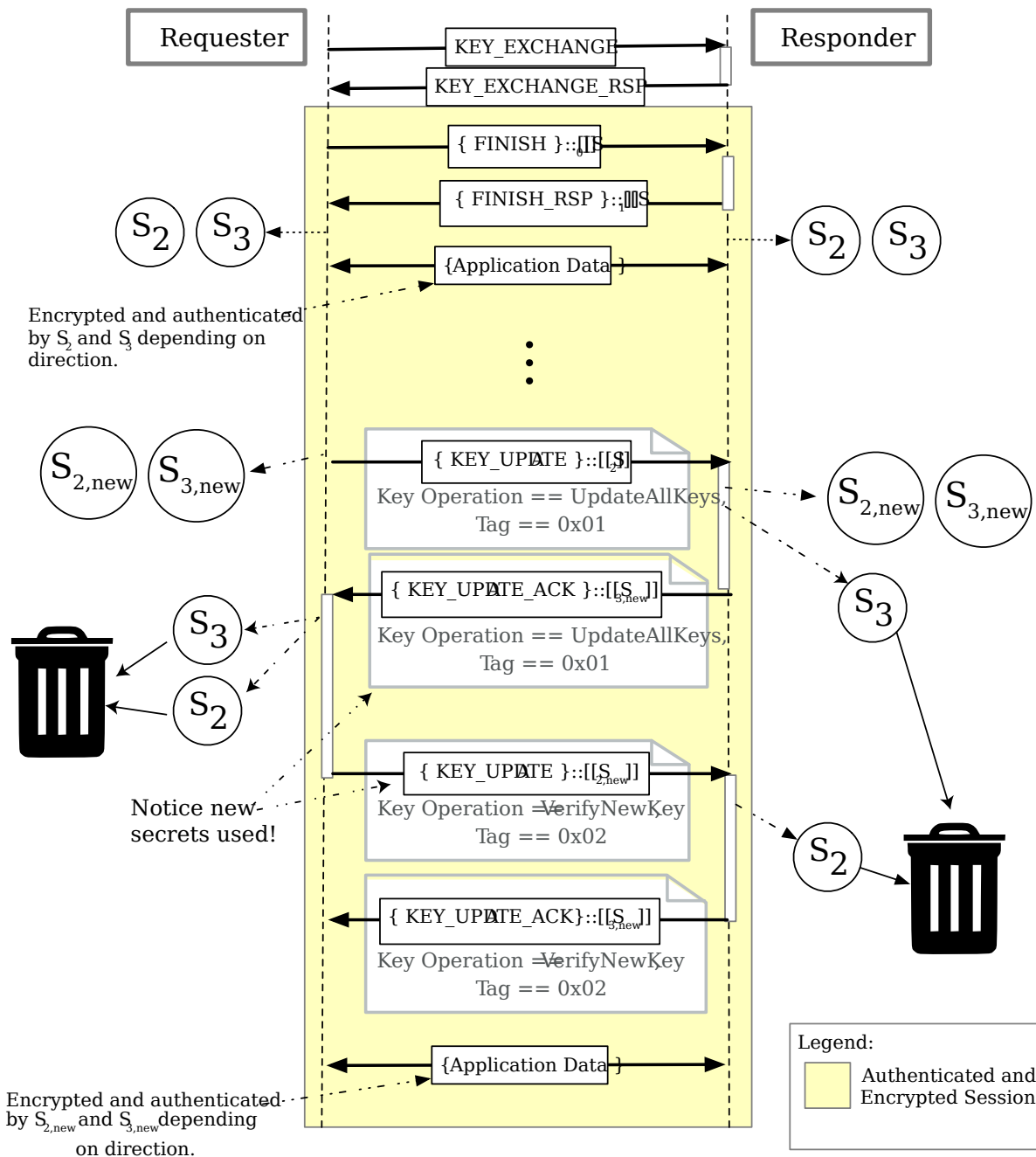
- 670 The data transport layer shall ensure that data transfer during key updates is managed in such a way that the correct keys are used before, during, and after the key update operation. How this is accomplished by the data transport layer is outside the scope of this specification.
- 671 Both the sender and the receiver shall derive the new keys as detailed in [Major secrets update](#).
- 672 The sender shall not use the new transmit key until after reception of the `KEY_UPDATE_ACK` response.
- 673 The sender and receiver shall use the new key on the `KEY_UPDATE` request with the `VerifyNewKey` command and all subsequent commands until another key update is performed.
- 674 In the case of a `KEY_UPDATE` request with `UpdateAllKeys`, the receiver shall use the new transmit key for the `KEY_UPDATE_ACK` response. The `KEY_UPDATE` request with `UpdateAllKeys` should only be used with physical transports that are single master to ensure that simultaneous `UpdateAllKeys` requests do not occur.
- 675 If the sender has not received `KEY_UPDATE_ACK`, the sender can retry or end the session. The sender shall not proceed to the next step until successfully receiving the corresponding `KEY_UPDATE_ACK`.
- 676 Upon the successful reception of the `KEY_UPDATE_ACK`, the sender shall transmit a `KEY_UPDATE` request with the `VerifyNewKey` operation using the new session keys. The sender can retry until the corresponding `KEY_UPDATE_ACK` response is received. However, the sender shall be prohibited, at this point, from restarting this process or going back to a previous step. Its only recourse in error handling is either to retry the same request or to terminate the session.
- 677 For `UpdateKey`, upon successful reception and verification of the `KEY_UPDATE` with the `VerifyNewKey` operation, the receiver can discard the old session keys. For `UpdateAllKeys`, upon successful reception and verification of the `KEY_UPDATE_ACK` with the `UpdateAllKeys` operation, the sender can discard the old session keys that protect receiver-sent messages. Upon successful reception and verification of the `KEY_UPDATE` with the `VerifyNewKey` operation, the receiver can discard the old session keys that protect sender-sent messages.
- 678 In certain scenarios, the receiver might need additional time to process the `KEY_UPDATE` request such as when processing data already in its buffer. Thus, the receiver can reply with an `ERROR` message of `ErrorCode=Busy`. The sender should retry the request after a reasonable period of time and with a reasonable number of retries to prevent premature session termination.
- 679 Finally, it bears repeating that a key update in one direction can happen simultaneously with a key update in the opposite direction. In this case, the aforementioned synchronization process occurs independently but simultaneously for each direction.
- 680 [Figure 18 — KEY_UPDATE protocol example flow](#) illustrates a typical key update initiated by the Requester to update its secret. In this example, the Responder and Requester are both capable of message authentication and encryption.
- 681 **Figure 18 — KEY_UPDATE protocol example flow**



683 **Figure 19 — KEY_UPDATE protocol change all keys example flow** illustrates a typical key update initiated by the Requester to update all secrets. In this example, the Responder and Requester are both capable of message authentication and encryption.

684 **Figure 19 — KEY_UPDATE protocol change all keys example flow**

685



686 10.21.2 KEY_UPDATE transport allowances

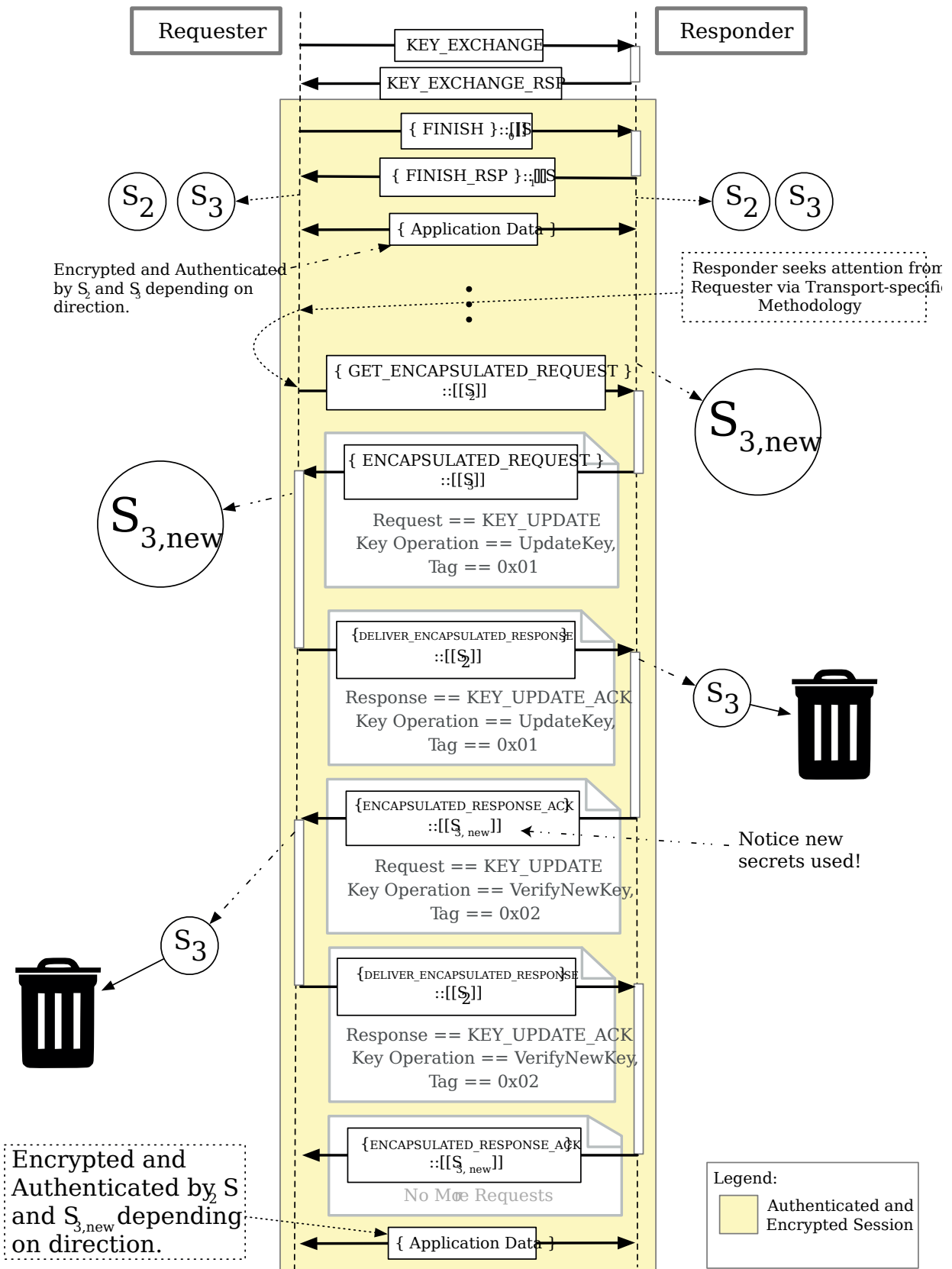
687 On some transports, bidirectional communication can occur asynchronously. On such transports, the transport can allow or disallow the **KEY_UPDATE** to be sent asynchronously without using the **GET_ENCAPSULATED_REQUEST**

mechanism. The transport should define the actual method to use. Such a definition is outside the scope of this specification.

688 [Figure 20 — KEY_UPDATE protocol example flow 2](#) illustrates a key update over a physical transport that has a limitation whereby only a single device (often called the "primary") is allowed to initiate all transactions on that bus. This physical transport specifies that a Responder shall alert the Requester through a side-band mechanism and to utilize the `GET_ENCAPSULATED_REQUEST` mechanism to fulfill SPDM-related requirements. Note also in this example that the Requester and Responder are both capable of encryption and message authentication.

689 **Figure 20 — KEY_UPDATE protocol example flow 2**

690



10.22 GET_ENCAPSULATED_REQUEST request and ENCAPSULATED_REQUEST response messages

In certain use cases, such as mutual authentication, the Responder needs the ability to issue its own SPDM request messages to the Requester. Certain transports prohibit the Responder from asynchronously sending out data on that transport. Cases like these are addressed through message encapsulation, which preserves the roles of Requester and Responder as far as the transport is concerned but enables the Responder to issue its own requests to the Requester. Message encapsulation is only allowed in certain scenarios, as described in various clauses in other parts of this specification. For example, [Figure 21 — Session-based mutual authentication example](#) and [Figure 22 — Optimized session-based mutual authentication example](#) illustrate the use of this scheme.

A Requester issues a `GET_ENCAPSULATED_REQUEST` request message to retrieve an encapsulated SPDM request message from the Responder. The response to this message is an `ENCAPSULATED_REQUEST` that encapsulates the SPDM request message as if the Responder were acting as a Requester. [Table 83 — GET_ENCAPSULATED_REQUEST request message format](#) describes the request message format. The Responder shall use the same SPDM version the Requester used.

10.22.1 Encapsulated request flow

The encapsulated request flow starts with the Requester sending a `GET_ENCAPSULATED_REQUEST` message and ends with an `ENCAPSULATED_RESPONSE_ACK` that carries no more encapsulated requests. The `GET_ENCAPSULATED_REQUEST` shall only be issued once, with the exception of retries. This is also illustrated in [Figure 21 — Session-based mutual authentication example](#).

When the Requester issues a `GET_ENCAPSULATED_REQUEST`, the encapsulated request flow shall start. Upon the successful reception of the `ENCAPSULATED_REQUEST` and when the encapsulated response is ready, the Requester shall continue the flow by issuing the `DELIVER_ENCAPSULATED_RESPONSE`. During this period, the Requester shall not issue any other message, with the exception of `GET_VERSION`, `RESPOND_IF_READY`, or `DELIVER_ENCAPSULATED_RESPONSE`. If a Responder receives a request other than `DELIVER_ENCAPSULATED_RESPONSE`, `RESPOND_IF_READY`, or `GET_VERSION`, the Responder should respond with an `ERROR` message of `ErrorCode=RequestInFlight`.

10.22.2 Optimized encapsulated request flow

The optimized encapsulated request flow is similar to the encapsulated request flow but without the need of a `GET_ENCAPSULATED_REQUEST`. This is because the encapsulated request accompanies one of the Session-Secrets-Exchange responses; thereby removing the obligation on the Requester to issue a `GET_ENCAPSULATED_REQUEST`. When the Responder includes an encapsulated request with a Session-Secrets-Exchange response, the optimized encapsulated request flow shall start. See [Figure 22 — Optimized session-based mutual authentication example](#).

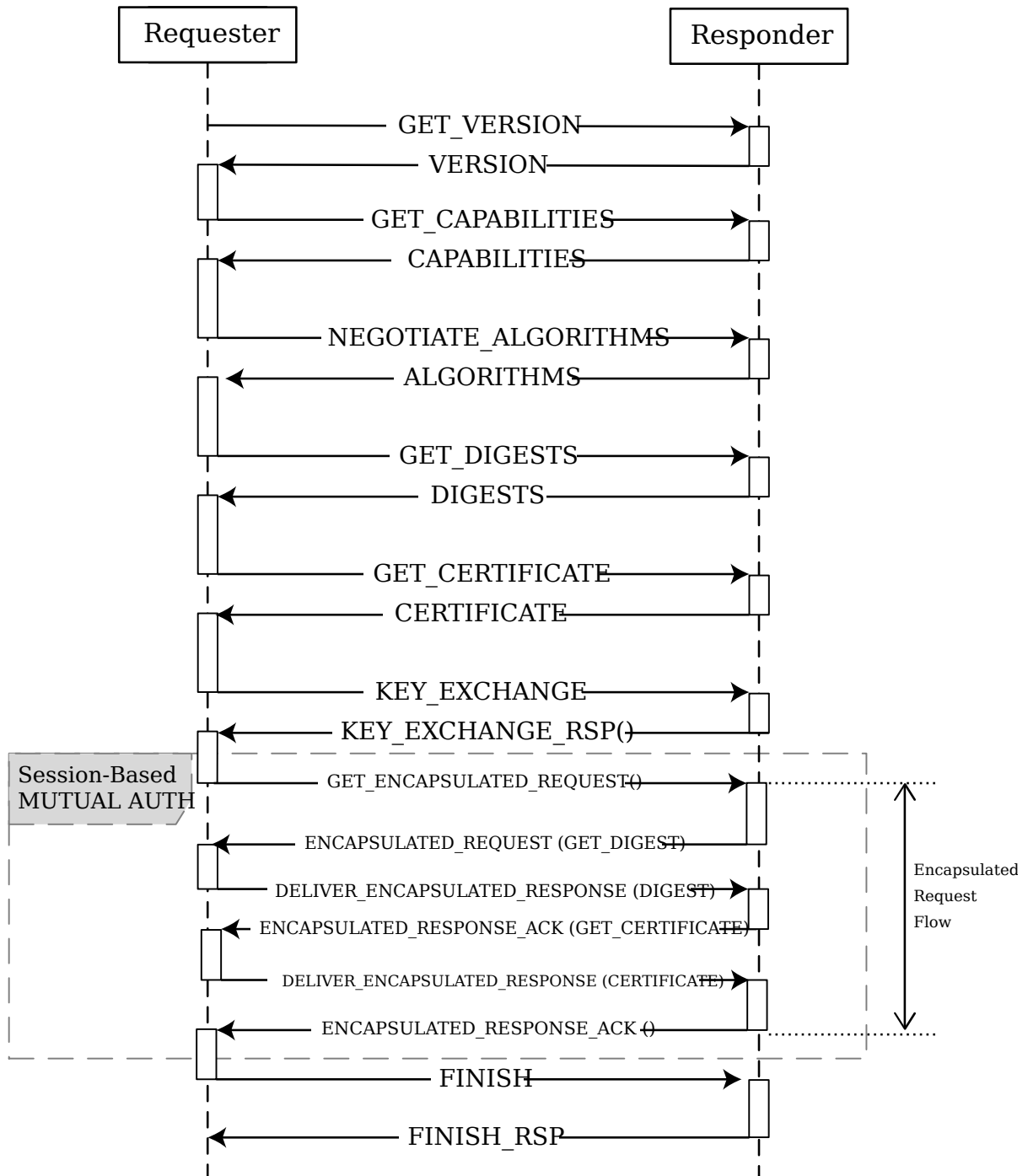
When the Requester successfully receives a Session-Secrets-Exchange response with an included encapsulated

request, the Requester shall send a `DELIVER_ENCAPSULATED_RESPONSE` after processing the encapsulated request. The Requester shall not issue any other requests except for `DELIVER_ENCAPSULATED_RESPONSE`, `RESPOND_IF_READY`, and `GET_VERSION`. If a Responder receives a request other than `DELIVER_ENCAPSULATED_RESPONSE`, `RESPOND_IF_READY`, or `GET_VERSION`, the Responder should respond with an `ERROR` message of `ErrorCode=RequestInFlight`.

700 [Figure 21 — Session-based mutual authentication example](#) shows an example of session-based mutual authentication:

701 **Figure 21 — Session-based mutual authentication example**

702



703 **Figure 22 — Optimized session-based mutual authentication example** shows an example of optimized session-based mutual authentication:

Figure 22 — Optimized session-based mutual authentication example

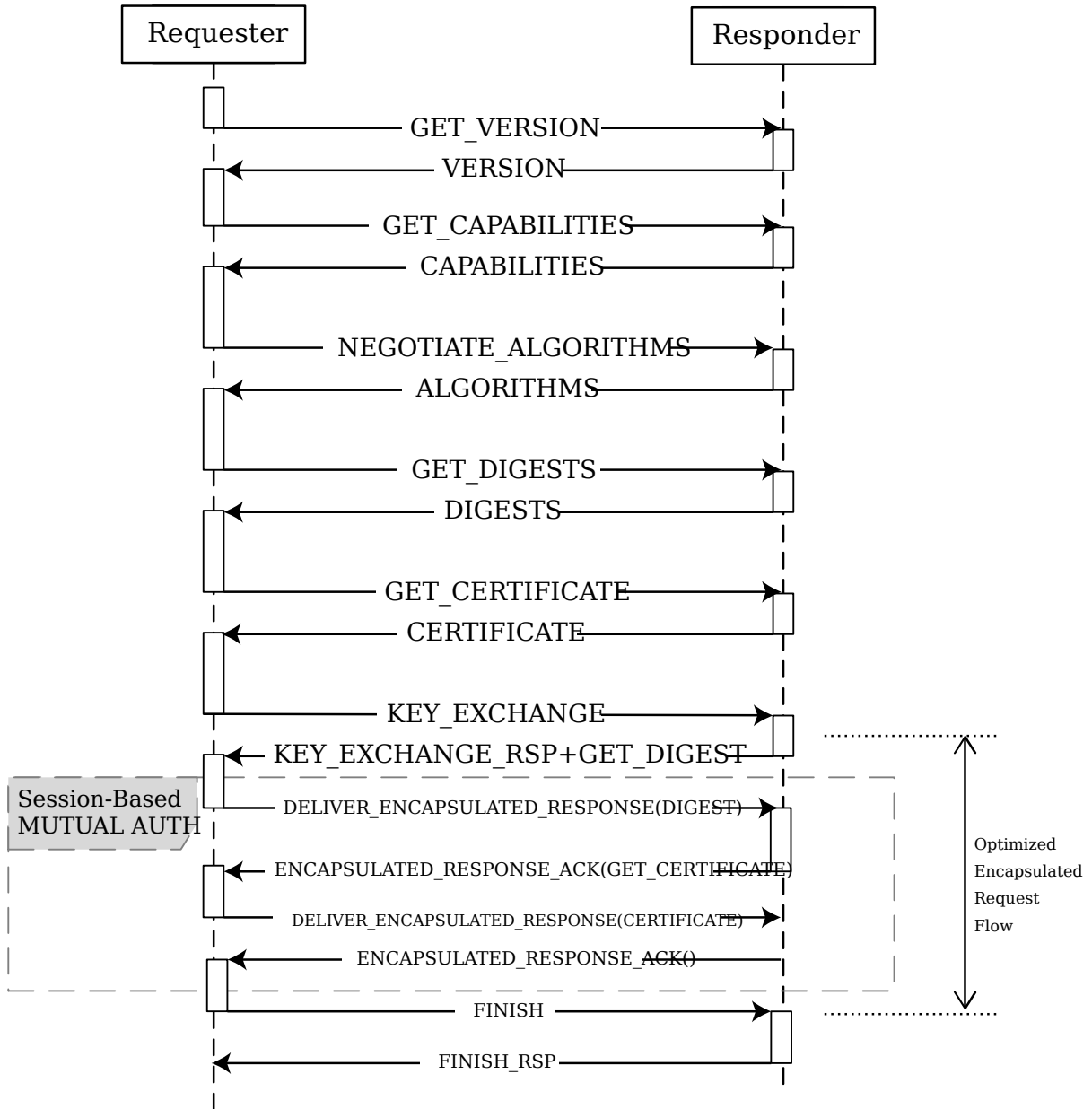


Table 83 — GET_ENCAPSULATED_REQUEST request message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0xEA</code> = <code>GET_ENCAPSULATED_REQUEST</code> . See Table 4 — SPDM request codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.

707 [Table 84 — ENCAPSULATED_REQUEST response message format](#) describes the format of this response.

708 **Table 84 — ENCAPSULATED_REQUEST response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x6A</code> = <code>ENCAPSULATED_REQUEST</code> response. See Table 5 — SPDM response codes .
2	Param1	1	Shall be the Responder-allocated Request ID. This field should be unique to help the Responder match response to request.
3	Param2	1	Reserved.
4	EncapsulatedRequest	Variable	Shall be the 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 <code>SPDMVersion</code> field. The <code>SPDMVersion</code> field of the <code>Encapsulated Request</code> shall be the same as the <code>SPDMVersion</code> of the <code>ENCAPSULATED_REQUEST</code> response. Both <code>GET_ENCAPSULATED_REQUEST</code> and <code>DELIVER_ENCAPSULATED_RESPONSE</code> shall be invalid requests, and the Requester should respond with an <code>ERROR</code> message of <code>ErrorCode=UnexpectedRequest</code> if these requests are encapsulated.

709 10.22.3 Triggering `GET_ENCAPSULATED_REQUEST`

710 Once a session has been established, the Responder might wish to send a request asynchronously, such as a `KEY_UPDATE` request, but cannot due to the limitations of the physical bus or transport protocol. In such a scenario, the transport and/or physical layer is responsible for defining an alerting mechanism for the Requester. Upon receiving the alert, the Requester shall issue a `GET_ENCAPSULATED_REQUEST` to the Responder.

711 If the physical transport cannot define an alerting mechanism to the Requester, the Requester can still use the encapsulated request flow as a polling mechanism by periodically sending the `GET_ENCAPSULATED_REQUEST` message. If the Responder receives a `GET_ENCAPSULATED_REQUEST` and has no request pending, the Responder should respond with an `ERROR` message of `ErrorCode=NoPendingRequests`.

712 10.22.4 Additional constraints

713 The `GET_ENCAPSULATED_REQUEST` and `ENCAPSULATED_REQUEST` messages shall only be allowed to encapsulate certain requests in certain scenarios. For details about these constraints, see the [Session, Basic mutual authentication](#), and [KEY_UPDATE request and KEY_UPDATE_ACK response messages](#) clauses.

714 10.23 DELIVER_ENCAPSULATED_RESPONSE request and ENCAPSULATED_RESPONSE_ACK response messages

715 As a Requester processes an encapsulated request, it needs a mechanism to deliver back the corresponding response. That mechanism shall be the `DELIVER_ENCAPSULATED_RESPONSE` and `ENCAPSULATED_RESPONSE_ACK` messages. The `DELIVER_ENCAPSULATED_RESPONSE`, which is an SPDM request, encapsulates the response and delivers it to the Responder. The `ENCAPSULATED_RESPONSE_ACK`, which is an SPDM response, acknowledges the reception of the encapsulated response.

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

717 In an encapsulated request flow, the Requester shall not send any other requests after the successful reception of the first encapsulated request, with the exception of `DELIVER_ENCAPSULATED_RESPONSE`, `RESPOND_IF_READY`, or `GET_VERSION`. If a Responder receives a request other than `DELIVER_ENCAPSULATED_RESPONSE`, `RESPOND_IF_READY`, or `GET_VERSION` after the successful reception of the first `DELIVER_ENCAPSULATED_RESPONSE`, the Responder should respond with an `ERROR` message of `ErrorCode=RequestInFlight`.

718 If `Param2` of `ENCAPSULATED_RESPONSE_ACK` is set to `0x00` or `0x02`, then this shall be the final encapsulated flow message that the Responder shall issue and the encapsulated flow shall be completed.

719 The timing parameters for the response shall depend on the encapsulated request. This enables the Responder to process the response before delivering the next request. See [Additional information](#).

720 [Table 85 — DELIVER_ENCAPSULATED_RESPONSE request message format](#) describes the request message format.

721 **Table 85 — DELIVER_ENCAPSULATED_RESPONSE request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0xEB</code> = <code>DELIVER_ENCAPSULATED_RESPONSE</code> Request. See Table 4 — SPDM request codes .

Byte offset	Field	Size (bytes)	Description
2	Param1	1	<p>Shall be the Request ID.</p> <p>The Requester shall use the same Request ID (that is, Param1) that was provided by the Responder in the corresponding ENCAPSULATED_REQUEST or ENCAPSULATED_RESPONSE_ACK.</p> <p>If the value was not provided by the Responder (for example, in the first message of an optimized encapsulated request flow), Request ID shall be 0.</p>
3	Param2	1	Reserved.
4	EncapsulatedResponse	Variable	<p>Shall be the SPDM Response Message.</p> <p>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 SPDMVersion field. The SPDMVersion field of the Encapsulated Response shall be the same as the SPDMVersion of the DELIVER_ENCAPSULATED_RESPONSE request. Both ENCAPSULATED_REQUEST and ENCAPSULATED_RESPONSE_ACK shall be invalid responses, and the Responder should respond with an ERROR message of ErrorCode=InvalidResponseCode if these responses are encapsulated.</p>

722 [Table 86 — ENCAPSULATED_RESPONSE_ACK response message format](#) describes the response message format.

723 **Table 86 — ENCAPSULATED_RESPONSE_ACK response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in SPDM version .
1	RequestResponseCode	1	Shall be 0x6B = ENCAPSULATED_RESPONSE_ACK. See Table 5 — SPDM response codes .
2	Param1	1	<p>Shall be the Request ID.</p> <p>If EncapsulatedRequest is present and if Param2 = 0x01, this field should contain a unique non-zero number to help the Responder match response to request. Otherwise, this field shall be 0x00.</p>
3	Param2	1	<p>Shall indicate the payload Type.</p> <p>If set to 0x00, no request message is encapsulated and the EncapsulatedRequest field is absent.</p> <p>If set to 0x01, the EncapsulatedRequest field follows.</p> <p>If set to 0x02, a 1-byte EncapsulatedRequest field follows containing the SlotID of the Requester's certificate chain used for mutual authentication. The value in this field shall be between 0 and 7 inclusive.</p> <p>All other values reserved.</p>

Byte offset	Field	Size (bytes)	Description
4	AckRequestID	1	Shall be the same as Param1 of the DELIVER_ENCAPSULATED_RESPONSE request message. The purpose of this field is to help the Requester distinguish between new requests and retries.
5	Reserved	3	Reserved.
8	EncapsulatedRequest	Variable	<p>If Param2 = 0x01, 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 SPDMVersion field. The SPDMVersion field of the EncapsulatedRequest shall be the same as the SPDMVersion of the ENCAPSULATED_REQUEST response. Both GET_ENCAPSULATED_REQUEST and DELIVER_ENCAPSULATED_RESPONSE shall be invalid requests, and the Requester shall respond with an ERROR message of ErrorCode=UnexpectedRequest if these requests are encapsulated.</p> <p>If Param2 = 0x02, the value of this field shall contain the SlotID corresponding to the certificate chain the Requester shall use for mutual authentication. The field size shall be 1 byte.</p> <p>If Param2 = 0x00, this field shall be absent.</p>

724 10.23.1 Additional information

725 Using unique Request ID s is highly recommended to aid the Responder in differentiating between retries and new DELIVER_ENCAPSULATED_RESPONSE messages. For example, if the Responder sent an ENCAPSULATED_RESPONSE_ACK message with a new encapsulated request and the message failed in transmission over the wire, the Requester would send a retry but that retry would still contain the response to the previous encapsulated request. Without a different Request ID, the Responder might mistake the retried DELIVER_ENCAPSULATED_RESPONSE for a new request. This mistake might cause further mistakes to occur.

726 The response timing for ENCAPSULATED_RESPONSE_ACK shall have the same timing constraints as the encapsulated request. For example, if the encapsulated request is CHALLENGE_AUTH, the Responder, too, would adhere to CT timing rules when it has a subsequent request. If necessary, the Requester can return an ERROR message of ErrorCode=ResponseNotReady.

727 The DELIVER_ENCAPSULATED_RESPONSE and ENCAPSULATED_RESPONSE_ACK messages shall only be allowed to encapsulate certain requests in certain scenarios. For details about these constraints, see the [Session, Basic mutual authentication](#), and [KEY_UPDATE request and KEY_UPDATE_ACK response messages](#) clauses.

728 10.23.2 Allowance for encapsulated requests

729 Only certain requests can be encapsulated in any encapsulated request flow. Their corresponding responses, including ERROR, can also be encapsulated. Additionally, these requests are only allowed in certain flows as described in various parts of this specification. This consolidated list shall be the requests that are allowed to be encapsulated:

- CHALLENGE

- GET_CERTIFICATE
- GET_DIGESTS
- KEY_UPDATE
- SUBSCRIBE_EVENT_TYPES
- SEND_EVENT
- GET_SUPPORTED_EVENT_TYPES
- GET_ENDPOINT_INFO

730 If a request is not in this list, the request and its corresponding response shall be prohibited from being encapsulated.

731 10.23.3 Certain error handling in encapsulated flows

732 These clauses describe special error scenarios and their handling requirements.

733 10.23.3.1 Response not ready

734 In an encapsulated request flow, a Responder can issue an encapsulated request that can take up to CT time to fulfill. When the Requester delivers an ERROR message of ErrorCode=ResponseNotReady, the Responder shall not encapsulate another request by setting Param2 in ENCAPSULATED_RESPONSE_ACK to a value of zero. This effectively and naturally terminates the encapsulated request flow.

735 The Responder should wait the amount of time indicated in the ERROR message for the particular error code.

736 When the timeout is near expiration, the Responder should perform the following:

1. Trigger its transport-defined alert mechanism to initiate the [Encapsulated request flow](#).
2. When the Requester issues a GET_ENCAPSULATED_REQUEST, the Responder should encapsulate the RESPOND_IF_READY request populated with the information from the previous ERROR with ResponseNotReady message.
 - If the Responder does not do this, the Requester can drop the original response.

737 10.23.3.2 Timeouts

738 If the Responder is not receiving a response to its encapsulated request, the Responder can trigger its transport-defined alert mechanism. When this occurs, if the Requester is in the middle of an existing encapsulated request flow with the same Responder, then the existing flow shall terminate and the Requester shall restart the encapsulated request flow.

739 Both Responder and Requester should comply with the timing requirements prescribed in [Timing requirements](#).

10.24 END_SESSION request and END_SESSION_ACK response messages

This request shall terminate a session. See the [Session termination phase](#) clause.

[Table 87 — END_SESSION request message format](#) and [Table 88 — End session request attributes](#) describe this format.

Table 87 — END_SESSION request message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0xEC = END_SESSION</code> . See Table 4 — SPDM request codes .
2	Param1	1	See Table 88 — End session request attributes .
3	Param2	1	Reserved.

Table 88 — End session request attributes

Bit offset	Value	Field	Description
0	0	Negotiated State Preservation Indicator	If the Responder supports Negotiated State caching (<code>CACHE_CAP=1</code>), the Responder shall preserve the Negotiated State. Otherwise, this field shall be ignored.
0	1	Negotiated State Preservation Indicator	If the Responder supports Negotiated State caching (<code>CACHE_CAP=1</code>), the Responder shall also clear the Negotiated State as part of session termination. If there is no Negotiated State to be cleared due to a previous <code>END_SESSION</code> request message with this field set to 1, this field shall be ignored. If the Responder does not support Negotiated State caching (<code>CACHE_CAP=0</code>), this field shall be ignored.
[7:1]	Reserved	Reserved	Reserved.

[Table 89 — END_SESSION_ACK response message format](#) describes the response message.

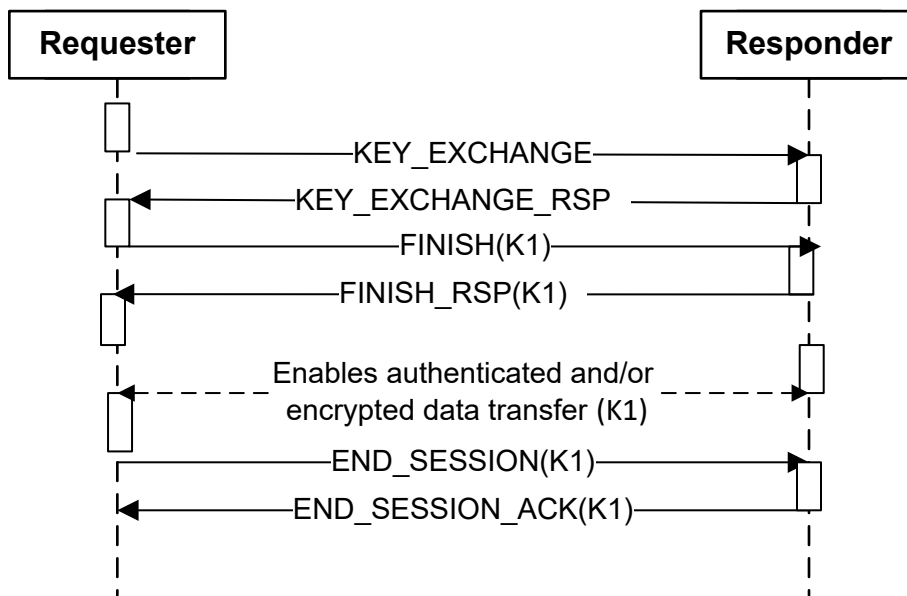
Table 89 — END_SESSION_ACK response message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x6C = END_SESSION_ACK</code> . See Table 5 — SPDM response codes .
2	Param1	1	Reserved.

Byte offset	Field	Size (bytes)	Description
3	Param2	1	Reserved.

Figure 23 — END_SESSION protocol flow shows the END_SESSION protocol flow:

Figure 23 — END_SESSION protocol flow



10.25 Certificate provisioning

These clauses describe the request and response messages used for provisioning a device with certificate chains. Provisioning of Slot 0 should only be done in a trusted environment (such as a secure manufacturing environment).

10.25.1 GET_CSR request and CSR response messages

The **GET_CSR** request message shall retrieve a Certificate Signing Request (CSR) from the Responder.

A Responder shall only process a **GET_CSR** request if it already possesses an appropriate asymmetric key pair for the signature suite (that is, the algorithms and associated parameters) required by the request. If more than one signature suite are supported, selection of the appropriate signature suite (and, thus, the key pair) shall be determined via the most recent **ALGORITHMS** response. Upon receiving a **GET_CSR** request, a Responder shall

generate and sign a CSR for the corresponding public key. The CSR shall be populated with a combination of attributes provided by the Requester via the `RequesterInfo` field and other attributes contributed by the Responder itself. The `RequesterInfo` format shall comply with the PKCS #10 specification in [RFC 2986](#), specifically the `CertificationRequestInfo` format. OEM extensions (that is, OEM OIDs) shall be encoded using the `Attributes` type. The Responder shall return an `ERROR` message of `ErrorCode=InvalidRequest` if it cannot support all the fields included in the `RequesterInfo`. If the Responder receives a `GET_CSR` request while another `GET_CSR` request is outstanding and if `Overwrite` is not specified (that is, Bit 7 of `Param1` is set to `0b`), the Responder can either overwrite the existing request and process the new `GET_CSR` request or respond with an `ERROR` message of `ErrorCode=Busy`. If the Responder receives a `GET_CSR` request while another `GET_CSR` request is outstanding and if `Overwrite` is specified (that is, Bit 7 of `Param1` is set to `1b`), the Responder shall overwrite the existing request and process the new `GET_CSR` request.

755 If the device requires a reset to complete the `GET_CSR` request, the device shall respond with an `ERROR` message of `ErrorCode=ResetRequired` with `Bit[2:0]` of the `Error Data` field set to a Responder-assigned `CSRTrackingTag` in the range of 1 to 7, inclusive. If the Responder requires a reset to process a `GET_CSR` request, but does not have any available `CSRTrackingTag`s, it shall respond with an `ERROR` message of `ErrorCode=Busy`. After the Responder has processed the reset, the Requester sends a `GET_CSR` request with `Bit[5:3]` in `Param2` set to the `CSRTrackingTag` that the Responder provided in the corresponding `ERROR` response, which signals to the Responder to send the `CSR` response associated with the previous request. After a Requester has retrieved a `CSR` response from a previous `GET_CSR` request, the Responder can discard any associated CSR data and reuse the `CSRTrackingTag`. If the Requester sends a `GET_CSR` request with a `CSRTrackingTag` that the Responder did not generate, the Responder shall either respond with an `ERROR` message of `ErrorCode=UnexpectedRequest` or drop the request.

756 The attributes of the resulting CSR and their values shall comply with the clauses presented in [SPDM certificate requirements and recommendations](#). If the `GET_CSR` request conforms to the `DeviceCert` model, the resulting CSR shall be for a Device Certificate. If the `GET_CSR` request conforms to the `AliasCert` model, the resulting CSR shall be for a Device Certificate CA. If the `GET_CSR` request conforms to the `GenericCert` model, the resulting CSR shall be for a Generic Leaf Certificate. See [Identity provisioning](#) for more details.

757 [Table 90 — GET_CSR request message format](#) shows the `GET_CSR` request message format.

758 [Table 92 — CSR response message format](#) shows the `CSR` response message format.

759 The `CSRdata` contained in a successful `CSR` response should be signed by an appropriate Certificate Authority. The details of the Public Key Infrastructure used to verify and sign the CSR and make the final certificate available for provisioning are outside the scope of this specification.

760 **Table 90 — GET_CSR request message format**

Byte offset	Field	Size (bytes)	Description
0	<code>SPDMVersion</code>	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	<code>RequestResponseCode</code>	1	Shall be <code>0xED</code> = <code>GET_CSR</code> . See Table 4 — SPDM request codes .

Byte offset	Field	Size (bytes)	Description
2	Param1	1	KeyPairID. The value of this field shall be the key pair ID identifying the desired asymmetric key pair to use in generating the CSR. If <code>MULTI_KEY_CONN_RSP</code> is false, the value shall be zero; otherwise, the value shall be non-zero.
3	Param2	1	Request Attributes. Shall be the format as Get CSR request attributes defines.
4	RequesterInfoLength	2	Shall be the length of the <code>RequesterInfo</code> field in bytes provided by the Requester. This field can be 0.
6	OpaqueDataLength	2	Shall be the size of the <code>OpaqueData</code> field that follows in bytes. The value should not be greater than 1024 bytes. Shall be 0 if no <code>OpaqueData</code> is provided.
8	RequesterInfo	<code>RequesterInfoLength</code>	Shall be the optional information provided by the Requester. This field shall be DER-encoded.
8 + <code>RequesterInfoLength</code>	OpaqueData	<code>OpaqueDataLength</code>	The Requester can include vendor-specific information for the Responder to generate the CSR. This field is optional. If present, this field shall conform to the selected opaque data format in <code>OtherParamsSelection</code> .

761 **Table 91 — Get CSR request attributes**

Bit offset	Field	Description
[2:0]	CSRCertModel	This field indicates the desired certificate model of the CSR. The value and format of this field shall be the same as <code>CertModel</code> in Certificate info .
[5:3]	CSRTrackingTag	If the Requester is requesting a previously requested <code>GET_CSR</code> after a reset has completed, this field shall contain the <code>CSRTrackingTag</code> of the associated <code>GET_CSR</code> request.
6	Reserved	Reserved.
7	Overwrite	If set, the Responder shall stop processing any existing <code>GET_CSR</code> request and overwrite it with this request, and the Responder shall discard all previously generated <code>CSRTrackingTag</code> s.

762 The `CSRCertModel` field in [Get CSR request attributes](#) helps the Responder determine the content of the CSR. For example, if the `CSRCertModel` indicates a device certificate model, the Responder may add additional OIDs such as those OIDs defined in this specification. If the `CSRCertModel` indicates an alias certificate model, the Responder sets the `CA` constraint to `TRUE` in the CSR.

763 **Table 92 — CSR response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x6D</code> = CSR. See Table 5 — SPDM response codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	CSRLength	2	Shall be the length of the <code>CSRdata</code> in bytes.
6	Reserved	2	Reserved.
8	CSRdata	<code>CSRLength</code>	Shall be the requested contents of the CSR. This field shall be DER-encoded.

764 The `CSRdata` format shall comply with the PKCS #10 specification in [RFC 2986](#), specifically the `CertificationRequest` format. When the Responder supports multiple asymmetric keys (`MULTI_KEY_CONN_RSP` is true) in the SPDM connection, the `SubjectPublicKeyInfo` as defined in [RFC 5280](#) shall contain values consistent with the requested asymmetric key pair (`KeyPairID`) in the corresponding request.

765 10.25.2 SET_CERTIFICATE request and SET_CERTIFICATE_RSP response messages

766 For Slot 0 provisioning, the Requester should issue `SET_CERTIFICATE` only in a trusted environment (such as a secure manufacturing environment). For slots 1-7, if the provisioning happens in a trusted environment, the Requester should issue `SET_CERTIFICATE` inside a secure session. If the provisioning for slots 1-7 is done outside of a trusted environment, the Requester shall issue `SET_CERTIFICATE` inside a secure session. Mutual authentication and/or other means for checking the authorization of the Requester that issues the `SET_CERTIFICATE` request should be performed. Requester authorization is outside the scope of this specification. The device might require a reset to complete the `SET_CERTIFICATE` request, potentially so that the device can generate `AliasCert` certificates using lower firmware layers. If the device requires a reset to complete the `SET_CERTIFICATE` request, then the device shall respond with an `ERROR` message of `ErrorCode=ResetRequired`. If the device temporarily cannot write to a slot, including in the case when it receives overlapping `SET_CERTIFICATE` requests from different Requesters, then the device shall respond with an `ERROR` message of `ErrorCode=Busy`.

767 If `Bit 7` of `SET_CERTIFICATE` . `Param1` is set to `1`, the Responder shall erase the certificate chain present in the slot identified by bits [3:0] of `SET_CERTIFICATE` . `Param1` and report it as unpopulated until it is re-provisioned. If the operation completes successfully, the Responder shall respond with a `SET_CERTIFICATE_RSP` response message with bits [3:0] of `Param1` identifying the `SlotID` of the slot that was erased. If the operation failed, the Responder shall respond with an `ERROR` message of `ErrorCode=OperationFailed`.

768 [Table 93 — SET_CERTIFICATE request message format](#) shows the `SET_CERTIFICATE` request message format.

769 [Table 95 — Successful SET_CERTIFICATE_RSP response message format](#) shows the `SET_CERTIFICATE_RSP` response message format.

770 **Table 93 — SET_CERTIFICATE request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0xEE</code> = <code>SET_CERTIFICATE</code> . See Table 4 — SPDM request codes .
2	Param1	1	Request attributes. Shall be the format that the set certificate request attributes table defines.
3	Param2	1	KeyPairID. The value of this field shall be the unique key pair number identifying the desired asymmetric key pair to associate with <code>SlotID</code> . If support for multiple asymmetric keys (<code>MULTI_KEY_CONN_RSP</code>) is false, the value of this field shall be zero.
4	CertChain	Variable	Shall be the contents of the target certificate chain, as specified in Certificates and certificate chains , with the additional requirement that it include the root certificate. If the Responder uses the <code>AliasCert</code> model (<code>ALIAS_CERT_CAP=1b</code> in its <code>CAPABILITIES</code> response) and <code>SetCertModel</code> is set to <code>AliasCert</code> , this field shall contain a partial certificate chain from the root CA to the Device Certificate CA. If the <code>Request attributes . Erase</code> bit is set, this field shall be absent.

771 **Table 94 — Set certificate request attributes**

Bit offset	Field	Description
[3:0]	SlotID	The certificate slot where the new certificate is written. The value in this field shall be between 0 and 7 inclusive.
[6:4]	SetCertModel	This field indicates the certificate model of the certificate chain. The value and format of this field shall be the same as <code>CertModel</code> in Certificate info . The value in this field shall match the value in the <code>CSRCertModel</code> field from the corresponding <code>GET_CSR</code> request.
7	Erase	If set, the certificate chain in the certificate slot identified by bits [3:0] shall be deleted. Additionally, if this bit is set, the <code>CertChain</code> field shall be absent and the value of <code>SetCertModel</code> shall be zero.

772 The Responder should verify that contents of the certificate chain meet the requirements in this specification for the requested certificate model and key pair.

773 **Table 95 — Successful SET_CERTIFICATE_RSP response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x6E</code> = <code>SET_CERTIFICATE_RSP</code> . See Table 5 — SPDM response codes .

Byte offset	Field	Size (bytes)	Description
			Bit [7:4]. Reserved.
2	Param1	1	Bit [3:0]. Shall be the SlotID where the new certificate is written. If the Erase bit is set in the Request attributes field, this field shall contain the SlotID of the slot that was erased. The value in this field shall be between 0 and 7 inclusive.
3	Param2	1	Reserved.

774 10.26 Large SPDM message transfer mechanism

775 A large SPDM message is an SPDM message whose size is either greater than the `DataTransferSize` of the receiving SPDM endpoint or greater than the transmit buffer size of the sending SPDM endpoint. These clauses provide a transport-agnostic mechanism to transfer large SPDM messages. This mechanism will be used only if the size of an SPDM message exceeds either the `DataTransferSize` of the receiving SPDM endpoint or the transmit buffer size of the sending SPDM endpoint. Additionally, the transport may provide an alternative method to transfer large SPDM messages. For SPDM messages that are less than or equal to both the `DataTransferSize` of the receiving SPDM endpoint and the transmit buffer size of the sending SPDM endpoint, the sending SPDM endpoint shall not utilize this transfer mechanism.

776 This transfer mechanism divides a large SPDM message into smaller fragments called chunks. The chunks shall be numbered and shall be transferred in sequence. The chunks and their sequence of transfer are described thus:

- The first chunk shall be assigned a numeric value of 0, the second chunk shall be assigned a numeric value of 1, the third chunk shall be assigned a numeric value of 2, and this pattern shall continue up to and including the last chunk. Each of these numeric values is called a chunk sequence number.
- The first chunk shall contain the first set of bytes of the large SPDM message, the second chunk shall contain the second set of bytes, the third chunk shall contain the third set of bytes, and this pattern shall continue up to and including the last chunk.
- All chunks shall represent all bytes of the large SPDM message without altering the message in any way.
- The sequence of transfer shall start with chunk sequence number 0 and shall continue with sequentially increasing chunk sequence numbers up to and including the last chunk.
- `CHUNK_SEND`, `CHUNK_GET`, and their corresponding Responses shall be used to transfer these chunks.

777 The `ChunkSeqNo` fields indicate the chunk sequence number for a given chunk.

778 The requests and responses, which these clauses define, handle the transfer of each chunk.

779 10.26.1 `CHUNK_SEND` request and `CHUNK_SEND_ACK` response message

780 The `CHUNK_SEND` request and the `CHUNK_SEND_ACK` response shall be used to send a request to an SPDM endpoint

when the size of the request is greater than either the `DataTransferSize` of the receiving SPDM endpoint or the transmit buffer size of the sending SPDM endpoint.

781 [Table 96 — CHUNK_SEND request format](#) describes the format for the request.

782 [Table 97 — Chunk sender attributes](#) describes the chunk sender attributes.

783 **Table 96 — CHUNK_SEND request format table**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x85</code> = <code>CHUNK_SEND</code> request. See Table 4 — SPDM request codes .
2	Param1	1	Shall be the Request Attributes. See Table 97 — Chunk sender attributes .
3	Param2	1	Shall be the handle. This field should uniquely identify the transfer of a large SPDM message. The value of this field shall be the same for all chunks of the same large SPDM message. The value of this field should either sequentially increase or sequentially decrease with each large SPDM message and with the expectation that it will wrap around after reaching the maximum or minimum value, respectively, of this field.
4	ChunkSeqNo	2	Shall identify the chunk sequence number associated with <code>SPDMchunk</code> .
6	Reserved	2	Reserved.
8	ChunkSize	4	Shall indicate the size of <code>SPDMchunk</code> . See Additional chunk transfer requirements .
12	LargeMessageSize	<code>L0</code> = 0 or 4	Shall indicate the size of the large SPDM message being transferred. This field shall only be present when <code>ChunkSeqNo</code> is zero and shall have a non-zero value. The value of this field shall be greater than the <code>DataTransferSize</code> of the receiving SPDM endpoint.
12 + <code>L0</code>	SPDMchunk	Variable	Shall contain the chunk of the large SPDM request message associated with <code>ChunkSeqNo</code> .

784 **Table 97 — Chunk sender attributes**

Bit offset	Field	Description
0	LastChunk	If set, the chunk indicated by <code>ChunkSeqNo</code> shall represent the last chunk of the large SPDM message.
[7:1]	Reserved	Reserved.

785 [Table 98 — CHUNK_SEND_ACK response message format](#) describes the format for the response.

786 **Table 98 — CHUNK_SEND_ACK response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	<code>0x05</code> = <code>CHUNK_SEND_ACK</code> response. See Table 5 — SPDM response codes .
2	Param1	1	Shall be the Response attributes. See Table 99 — Chunk receiver attributes .
3	Param2	1	Shall contain the handle from the corresponding <code>CHUNK_SEND</code> request. This field should uniquely identify the transfer of a large SPDM message. The value of this field shall be the same for all chunks of the same SPDM message.
4	ChunkSeqNo	2	Shall be the same as <code>ChunkSeqNo</code> in the corresponding request.
6	ResponseToLargeRequest	Variable	Shall be present on the last chunk (that is, when <code>LastChunk</code> is set), or when the <code>EarlyErrorDetected</code> bit in <code>Param1</code> is set. This field shall contain the response to the large SPDM request message. When the <code>EarlyErrorDetected</code> bit in <code>Param1</code> is set, this field shall contain an <code>ERROR</code> message.

787 [Table 99 — Chunk receiver attributes](#) describes the chunk receiver attributes:

788 **Table 99 — Chunk receiver attributes**

Bit offset	Field	Description
0	EarlyErrorDetected	If set, the receiver of a large SPDM request message detected an error in the Request before the last chunk was received. If set, the sender of the large SPDM request message shall terminate the transfer of any remaining chunks. After addressing the issue, the sender of the failed large SPDM request message can transfer the fixed large SPDM request message as a new transfer.
[7:1]	Reserved	Reserved.

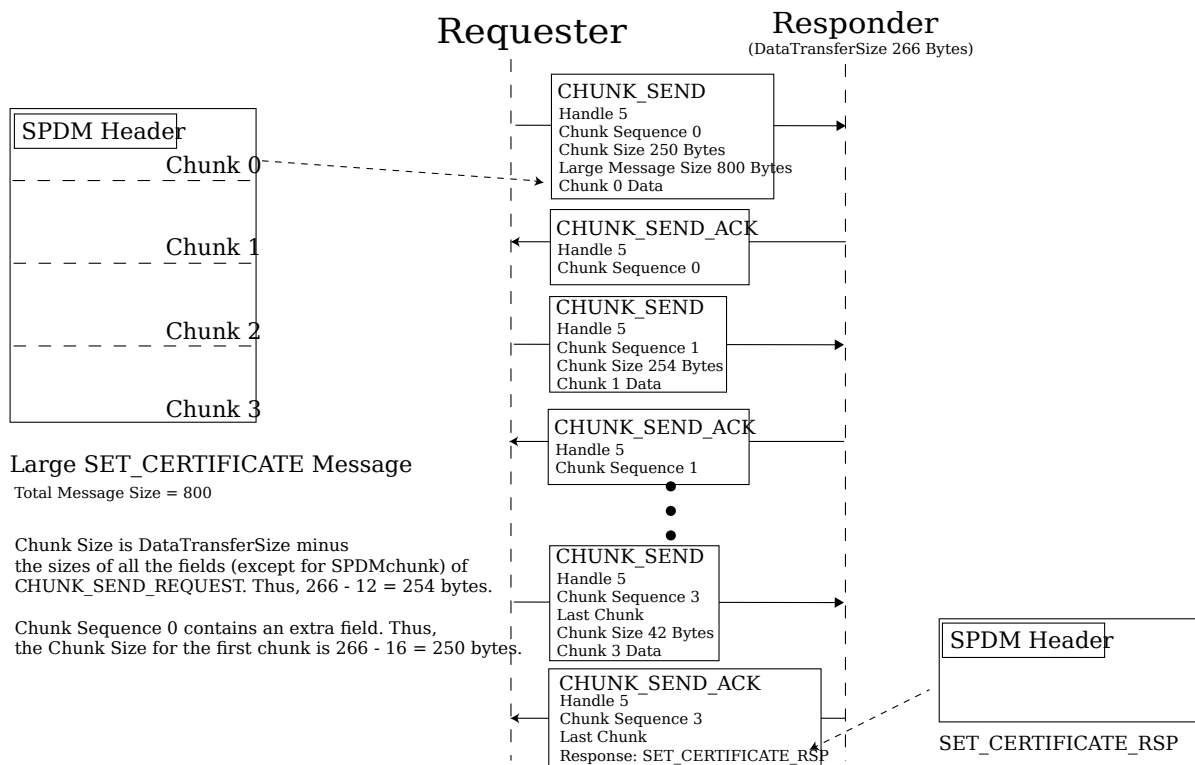
789 [Table 98 — CHUNK_SEND_ACK response message format](#) describes the format for the response.

790 Upon reception of the last chunk, the receiving SPDM endpoint shall respond with the response corresponding to the large SPDM request message in `ResponseToLargeRequest`. If placing the response in `ResponseToLargeRequest` causes the size of the `CHUNK_SEND_ACK` to exceed the `DataTransferSize`, the receiving end point shall, instead, respond to `CHUNK_SEND` with an `ERROR` message of `ErrorCode=LargeResponse`. An `ERROR` message of `ErrorCode=LargeResponse` shall not be allowed in `ResponseToLargeRequest`. `ERROR` messages with other error codes can be placed in `ResponseToLargeRequest` to distinguish between an `ERROR` message to the `CHUNK_SEND` request and an `ERROR` message that is a response to the large SPDM request message.

791 [Figure 24 — Large SET_CERTIFICATE example](#) illustrates the sending of a large SPDM request message to a Responder.

792 **Figure 24 — Large SET_CERTIFICATE example**

793



794 10.26.2 CHUNK_GET request and CHUNK_RESPONSE response message

795 **CHUNK_GET** request and **CHUNK_RESPONSE** response shall be used to retrieve a Large SPDM Response from an SPDM endpoint when the size of the Response is greater than the **DataTransferSize** of the SPDM endpoint receiving the Response.

796 When responding to a Request of any size, if the corresponding response will be a Large SPDM Response, the responding SPDM endpoint shall respond with an **ERROR** message of **ErrorCode=LargeResponse**. This **ERROR** message contains a handle to uniquely identify the given Large SPDM Response. The handle shall be used for all **CHUNK_GET** Requests retrieving the same large SPDM message. The value of the handle is indicated in the **Handle** field of this **ERROR** message.

797 [Table 100 — CHUNK_GET request format](#) describes the format for the request.

798 **Table 100 — CHUNK_GET request format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in SPDM version .
1	RequestResponseCode	1	Shall be 0x86 = CHUNK_GET request. See Table 4 — SPDM request codes .

Byte offset	Field	Size (bytes)	Description
2	Param1	1	Reserved.
3	Param2	1	Shall contain a handle. This field shall be the same value as given in the <code>Handle</code> field of the <code>ERROR</code> message of <code>ErrorCode=LargeResponse</code> .
4	ChunkSeqNo	2	Shall indicate the desired chunk sequence number of the Large SPDM Response to retrieve.

799 [Table 101 — CHUNK_RESPONSE response format](#) describes the format for the response.

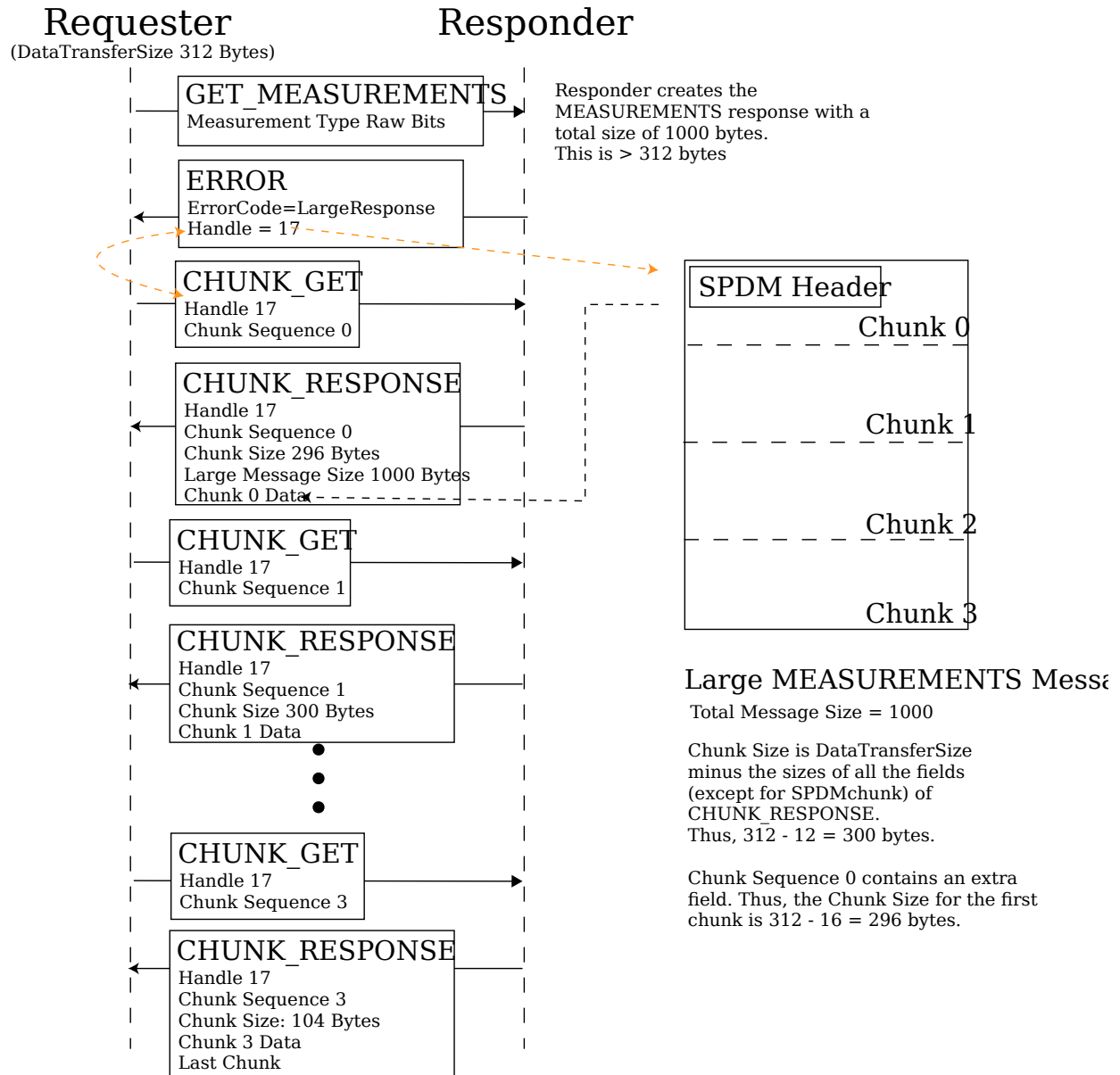
800 **Table 101 — CHUNK_RESPONSE response format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	Shall be <code>0x06 = CHUNK_RESPONSE</code> response. See Table 5 — SPDM response codes .
2	Param1	1	Shall be the Response attributes. See Table 97 — Chunk sender attributes .
3	Param2	1	Shall be the handle. This field shall be the same for all chunks of the same Large SPDM Response. The value of this field shall be the same value as in <code>Param2</code> field of <code>CHUNK_GET</code> .
4	ChunkSeqNo	2	Shall identify the chunk sequence number associated with <code>SPDMchunk</code> . The value of this field shall be the same value as <code>ChunkSeqNo</code> in the <code>CHUNK_GET</code> .
6	Reserved	2	Reserved.
8	ChunkSize	4	Shall indicate the size of <code>SPDMchunk</code> . See Additional chunk transfer requirements .
12	LargeMessageSize	$L_0 = 0$ or 4	Shall indicate the size of the large SPDM message being transferred. Shall only be present when <code>ChunkSeqNo</code> is zero and shall have a non-zero value. The value of this field should be greater than the <code>DataTransferSize</code> of the receiving SPDM endpoint.
12 + L_0	SPDMchunk	Variable	Shall contain the chunk of the large SPDM request message associated with <code>ChunkSeqNo</code> .

801 [Figure 25 — Large MEASUREMENT example](#) illustrates the sending and retrieval of a Large SPDM Response that was the result of a Requester issuing a `GET_MEASUREMENTS` request.

802 **Figure 25 — Large MEASUREMENT example**

803



804 10.26.3 Additional chunk transfer requirements

805 When transferring a large SPDM message, an SPDM endpoint shall be prohibited from transferring a chunk sequence number (that is, a `ChunkSeqNo`) less than the current chunk sequence number. In other words, an SPDM endpoint cannot go backwards in the transfer or re-send or re-retrieve a chunk sequence number less than the current one in the transfer. However, due to retries, an SPDM endpoint might re-send or re-retrieve the current chunk number in the transfer. Additionally, if the receiving SPDM endpoint receives an out-of-order chunk sequence number, the receiving

SPDM endpoint shall either silently discard the request or respond with an `ERROR` message of `ErrorCode=InvalidRequest`.

806 The value of `ChunkSize` fields shall be one that ensures the total size of `CHUNK_SEND` or `CHUNK_RESPONSE` does not exceed the `DataTransferSize` of the receiving SPDM endpoint. For all chunks that are not the last chunk, `ChunkSize` shall be a value where the total size of `CHUNK_SEND` or `CHUNK_RESPONSE` shall be from `MinDataTransferSize` to the `DataTransferSize` of the receiving SPDM endpoint. For the last chunk, `ChunkSize` shall be a value where the total size of `CHUNK_SEND` or `CHUNK_RESPONSE` shall be equal to or less than the `DataTransferSize` of the receiving SPDM endpoint.

807 While this transfer mechanism can carry any Request or Response, this transfer mechanism shall prohibit `CHUNK_SEND`, `CHUNK_GET`, and their corresponding responses to be transferred as chunks themselves. Additionally to ensure the general interoperability and reliability of this transfer mechanism, these messages shall be prohibited from being transferred in chunks using this transfer mechanism:

- `GET_VERSION`
- `VERSION`
- `GET_CAPABILITIES`
- `CAPABILITIES` with `Param1` in the `GET_CAPABILITIES` request set to 0.
- `ERROR`
 - An `ERROR` message with an `ErrorCode` other than `LargeResponse` can be placed in the `ResponseToLargeRequest` of a `CHUNK_SEND_ACK` response.

808 This transfer mechanism can carry Requests and Responses that are involved in signature generation or verification and other cryptographic computations. However, this transfer mechanism is not part of any signature generation or verification or cryptographic computation. In other words, `CHUNK_SEND`, `CHUNK_GET`, and their corresponding responses shall not become part of any data or bit stream, such as message transcript, transcript, and so on, that are used to verify or generate a signature or other cryptographic information. Signature generation, signature verification, and other cryptographic computations operate on the large SPDM messages, themselves, which other parts of this specification define.

809 The `ERROR` message of `ErrorCode=ResponseNotReady` shall not be used to directly respond to `CHUNK_SEND` and `CHUNK_GET` requests. However, the `ResponseToLargeRequest` can contain an `ERROR` message of `ErrorCode=ResponseNotReady`.

810 While a large SPDM message is being transferred in chunks, this large SPDM message is not considered a complete SPDM message until the last chunk is received. Therefore, as soon as the `CHUNK_SEND` request begins transmission, this large SPDM request message is considered to be outstanding.

811 10.27 Key configuration

812 Key configuration is the ability to retrieve or configure various parameters pertaining to asymmetric keys for a given SPDM endpoint. These clauses describe the requests and responses that provide key-configuration capabilities.

813 SPDM endpoints can contain key pair ID(s) (`KeyPairID`) that are fixed and already provisioned, key pair IDs that are configurable, or an assortment of both types. For configurable key pair IDs, one or more parameters related to the key pair are configurable. The requests and responses in these clauses provide the details for each `KeyPairID` . An SPDM endpoint shall contain `KeyPairID` s starting from 1 to `TotalKeyPairs` inclusive and without gaps.

814 The Responder should authorize the Requester before allowing it to retrieve or change information related to a key pair. The method of authorization is outside the scope of this specification.

815 In general, if a key pair ID is configurable, the high-level flow for provisioning and configuring a key pair ID to a usable state should follow these steps:

1. Use the `GET_KEY_PAIR_INFO` request and its corresponding response to retrieve information about one or more key pair ID(s).
2. Use the `SET_KEY_PAIR_INFO` request and its corresponding response to configure the key pair ID.
 - Ensure the key pair ID is associated with one or more certificate slots.
3. Use the `GET_CSR` and/or `SET_CERTIFICATE` requests and their corresponding responses to provision a certificate chain to one or more of the certificate slots the key pair ID is associated with.

816 To return a key pair ID to its initial or default values, follow these steps:

1. Use the `GET_KEY_PAIR_INFO` request and its corresponding response to retrieve information about the desired key pair ID.
 - In particular, note all the certificate slots the key pair ID is associated with.
2. Use the `SET_CERTIFICATE` request and its corresponding response to erase all certificate chains associated with the key pair ID.
3. Use the `SET_KEY_PAIR_INFO` request and its corresponding response to erase the key pair ID.

817 Outside of a session, the Requester and Responder should only issue `GET_KEY_PAIR_INFO` , `SET_KEY_PAIR_INFO` , and their corresponding responses while in a trusted environment.

818 10.27.1 `GET_KEY_PAIR_INFO` request and `KEY_PAIR_INFO` response

819 The `GET_KEY_PAIR_INFO` request shall retrieve key pair information from the Responder. This request and its response shall report information for all key pairs on the Responder independent of any negotiated parameters of the current SPDM connection. This allows the Requester to inquire about key pair information for all key pair IDs without restarting the SPDM connection.

820 [Table 102 — `GET_KEY_PAIR_INFO` request message format](#) shows the `GET_KEY_PAIR_INFO` request message format.

821 **Table 102 — `GET_KEY_PAIR_INFO` request message format**

Byte offset	Field	Size (bytes)	Description
0	<code>SPDMVersion</code>	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .

Byte offset	Field	Size (bytes)	Description
1	RequestResponseCode	1	GET_KEY_PAIR_INFO=0xFC . See Table 4 — SPDM request codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	KeyPairID	1	The value of this field shall indicate which key pair ID's information to retrieve.

822 The corresponding successful response shall be the `KEY_PAIR_INFO` response as [Table 103 — KEY_PAIR_INFO response message format](#) describes.

823 **Table 103 — KEY_PAIR_INFO response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	KEY_PAIR_INFO = 0x7C . See Table 4 — SPDM request codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	TotalKeyPairs	1	The value of this field shall indicate the total number of key pairs on the Responder.
5	KeyPairID	1	The value of this field shall be the same value as the <code>KeyPairID</code> field in the corresponding request. The remaining fields in this response shall pertain to the requested key pair ID in the corresponding Request.
6	Capabilities	2	This field indicates the capabilities of the requested key pair (<code>KeyPairID</code>). The format of this field shall be as Table 104 — Key pair capabilities format defines.
8	KeyUsageCapabilities	2	This field shall indicate the key usages the Responder allows. The format of this field shall be as Key usage bit mask defines. At least one bit shall be set. The Responder shall indicate support for one or more key usages by setting the corresponding bits.
10	CurrentKeyUsage	2	This field shall indicate the currently configured key usage for the requested key pair ID. The format of this field shall be as Key usage bit mask defines. If no bits are set, this field shall indicate that the key usage for this key pair ID has not yet been configured. More than one bit can be set. If a bit is set, the Responder shall support cryptographic operations (such as signature generation) for the corresponding key usage.
12	AsymAlgoCapabilities	4	This field shall indicate the asymmetric algorithms the Responder supports for this key pair ID. The format of this field shall be as Table 105 — Asymmetric algorithm capabilities format defines. The Responder shall indicate support for one or more asymmetric algorithms by setting the corresponding bits. At least one bit shall be set.

Byte offset	Field	Size (bytes)	Description
16	CurrentAsymAlgo	4	This field shall indicate the currently configured asymmetric algorithm for this key pair ID. The format of this field shall be as Table 105 — Asymmetric algorithm capabilities format defines. No more than one bit shall be set. If no bits are set, this field shall indicate that the asymmetric algorithm for this key pair has not yet been selected. The set bit shall indicate that the corresponding asymmetric algorithm is currently configured.
20	PublicKeyInfoLen	2	This field shall indicate the size in bytes of the <code>PublicKeyInfo</code> field in this request. A value of zero shall indicate that the actual key pair is absent or has yet to be generated. Otherwise, the value of this field shall be non-zero.
22	AssocCertSlotMask	1	This field is a bit mask representing the currently associated certificate slots. A set bit at position X shall indicate an association between certificate slot X and the requested <code>KeyPairID</code> . If <code>ShareableCap</code> is not set, no more than one bit shall be set.
23	PublicKeyInfo	Variable	The field shall contain the public key information for the requested key pair ID. The format of this field shall be the DER encoding of the <code>AlgorithmIdentifier</code> structure in an X.509 v3 certificate. See the "4.1.2.7. Subject Public Key Info" clauses in RFC 5280 for additional details. Within the <code>AlgorithmIdentifier</code> structure, the <code>parameters</code> member shall be present and contain values consistent with the information pertaining to the requested key pair ID.

824 [Table 104 — Key pair capabilities format](#) defines the format for capabilities associated with a key pair ID.

825 **Table 104 — Key pair capabilities format**

Bit offset	Field	Description
0	GenKeyCap	If set, this key pair identified by the given <code>KeyPairID</code> can be generated or regenerated.
1	ErasableCap	If set, this key pair identified by the given <code>KeyPairID</code> can be erased.
2	CertAssocCap	If set, the Responder allows a Requester to change the association between the given <code>KeyPairID</code> and <code>CertSlot</code> .
3	KeyUsageCap	If set, the Responder allows a Requester to change the key usage for the given <code>KeyPairID</code> .
4	AsymAlgoCap	If set, the Responder allows a Requester to change the asymmetric algorithm for the given <code>KeyPairID</code> .
5	ShareableCap	If set, the Responder allows a Requester to associate the given <code>KeyPairID</code> with more than one <code>CertSlot</code> . This bit shall not be set if <code>CertAssocCap</code> is not set.
All other bits	Reserved	Reserved.

826 [Table 105 — Asymmetric algorithm capabilities format](#) defines the bit mapping for asymmetric algorithms support. See [Table 136 — SPDM Asymmetric Signature Reference Information](#) for references for the asymmetric algorithms.

827 **Table 105 — Asymmetric algorithm capabilities format**

Bit offset	Asymmetric Algorithm
0	RSA 2048
1	RSA 3072
2	RSA 4096
3	ECC NIST P256
4	ECC NIST P384
5	ECC NIST P521
6	SM2 P256
7	Ed25519
8	Ed448
All other bits	Reserved.

828 10.27.2 SET_KEY_PAIR_INFO request and SET_KEY_PAIR_INFO_ACK response

829 The SET_KEY_PAIR_INFO request and the corresponding successful SET_KEY_PAIR_INFO_ACK response shall configure one or more parameters for one key pair ID (KeyPairID).

830 [Table 106 — SET_KEY_PAIR_INFO request message format](#) defines the format for the SET_KEY_PAIR_INFO request.

831 Table 106 — SET_KEY_PAIR_INFO request message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in SPDM version .
1	RequestResponseCode	1	SET_KEY_PAIR_INFO = 0xFD . See Table 4 — SPDM request codes .
2	Param1	1	Operation. This field shall indicate the desired operation. The format of this field shall be the format as Table 107 — Key pair operations defines. If the operation is KeyPairErase , all fields after KeyPairID field in this request shall be absent.
3	Param2	1	Reserved.
4	KeyPairID	1	The value of this field shall indicate the key pair ID's information to change.
5	Reserved	1	Reserved.

Byte offset	Field	Size (bytes)	Description
6	DesiredKeyUsage	2	This field shall indicate the desired key usage (KEY_PAIR_INFO . CurrentKeyUsage) for the requested key pair ID (KeyPairID). The format of this field shall be as Key usage bit mask defines. If no bits are set, the Responder shall not change the current key usage. More than one bit can be set. The Requester shall only select from bits that are set in the KeyUsageCapabilities field of the KEY_PAIR_INFO response for the requested KeyPairID . If KeyUsageCap is not set for the requested KeyPairID , this field shall be zero.
8	DesiredAsymAlgo	4	This field shall indicate the desired asymmetric algorithm (KEY_PAIR_INFO . CurrentAsymAlgo) for the requested key pair ID. The format of this field shall be as Table 105 — Asymmetric algorithm capabilities format defines. If no bits are set, the Responder shall not change the current configuration for the asymmetric algorithm. No more than one bit shall be set. The Requester shall only select from bits that are set in the AsymAlgoCapabilities field of the KEY_PAIR_INFO response for the requested KeyPairID . If AsymAlgoCap is not set for the requested KeyPairID , this field shall be zero.
12	DesiredAssocCertSlotMask	1	This field is a bit mask representing the desired certificate slot association. A set bit at position X shall indicate an association between certificate slot X and the requested KeyPairID . An unset bit at position X shall indicate no association between certificate slot X and the requested KeyPairID . The Responder shall either remove an association or create an association between the corresponding certificate slot and the requested KeyPairID , depending on the value of each bit in this field. If ShareableCap is not set, no more than one bit shall be set.

832 [Table 107 — Key pair operations](#) defines a numeric mapping to an operation.

833 **Table 107 — Key pair operations**

Value	Operation Name	Description
0	ParameterChange	Shall indicate an operation that modifies one or more key-related parameters. The DesiredKeyUsage , DesiredAsymAlgo , and DesiredAssocCertSlotMask fields shall be present.
1	KeyPairErase	Shall indicate an operation that erases all information relating to a KeyPairID . The DesiredKeyUsage , DesiredAsymAlgo , and DesiredAssocCertSlotMask fields shall be absent.
2	GenerateKeyPair	Shall indicate an operation that generates a new key pair for this KeyPairID . The DesiredKeyUsage , DesiredAsymAlgo , and DesiredAssocCertSlotMask fields shall be present.

834 [Table 108 — SET_KEY_PAIR_INFO_ACK response message format](#) defines the format for SET_KEY_PAIR_INFO_ACK response.

835 **Table 108 — SET_KEY_PAIR_INFO_ACK response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in SPDM version .

Byte offset	Field	Size (bytes)	Description
1	RequestResponseCode	1	SET_KEY_PAIR_INFO_ACK = 0x7D . See Table 4 — SPDM request codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.

836 10.27.3 Key pair ID modification error handling

837 These clauses describe some basic configuration error scenarios an SPDM endpoint should handle.

838 The first error scenario is a request for key generation (`GenerateKeyPair`) when no asymmetric algorithm has been selected yet. A Responder should respond with an `ERROR` message of `ErrorCode=OperationFailed` .

839 Key usage for a key pair ID does not need to be specified until the key pair ID is associated with a certificate slot, so this information is not needed for a `GenerateKeyPair` operation. The Responder should decide when it needs to know the key usage information for a configurable key usage.

840 For a `KeyPairErase` or `GenerateKeyPair` operation request, the Responder shall ensure that the requested `KeyPairID` has no association with any certificate slot. Otherwise, the Responder should respond with an `ERROR` message of `ErrorCode=OperationFailed` .

841 10.28 Event mechanism

842 An SPDM endpoint may want to be notified of changes from another SPDM endpoint. These change notifications are called events. The SPDM event mechanism provides a framework for the asynchronous notification of events over a secure session. An Event Notifier is an SPDM endpoint sending an event, and an Event Recipient is an SPDM endpoint receiving an event. An SPDM endpoint can be both an Event Notifier and an Event Recipient in the same secure session. See [Session](#) for details on secure sessions. There can be multiple sessions between the same Responder and same Requester. The event mechanism applies to each session individually.

843 An event is identified by its event group, event type, and an event instance ID. An event group is a group of all event types a given standards body or vendor defines. An event type classifies the event by indicating its type. The event instance ID is a unique numeric value that represents that occurrence of the event.

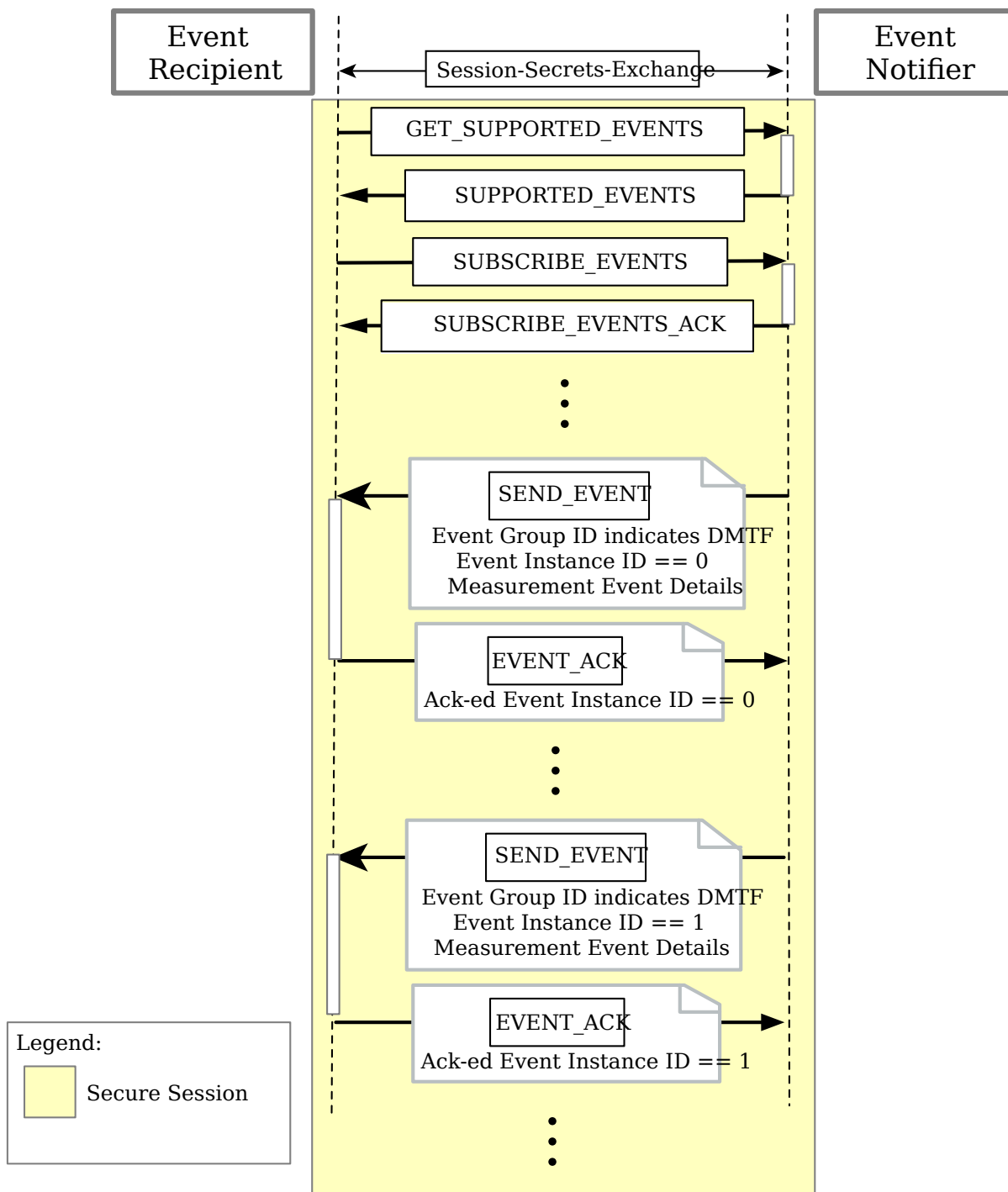
844 An Event Recipient can select the event types that it wants to receive. An event subscription is a list of event types an Event Recipient wants to receive. The Event Notifier manages the event subscription. An Event Notifier shall only send events of event types that match the event types in the event subscription. See [DMTF Event Types](#) for DMTF-defined event types.

845 An Event Notifier shall not send any events in a session until an Event Recipient subscribes to one or more events.

846 The [Event Flow diagram](#) illustrates a typical event flow for event subscription and event delivery over a transport
capable of asynchronous bidirectional communication.

847 **Figure 26 — Event flow diagram**

848



849 For transports that prohibit a Responder from asynchronously sending out data, the Event Notifier and Event

Recipient can use the [encapsulated request flow](#) to deliver or receive events. The encapsulated request flow allows for a polling methodology as [Triggering GET_ENCAPSULATED_REQUEST](#) describes.

850 When `EVENT_CAP` is set, an Event Notifier shall support `SUBSCRIBE_EVENT_TYPES` , `GET_SUPPORTED_EVENT_TYPES` , `SEND_EVENT` , and their corresponding response messages.

851 10.28.1 GET_SUPPORTED_EVENT_TYPES request and SUPPORTED_EVENT_TYPES response message

852 These request and response messages retrieve the list of all event types supported by the Event Notifier. Each event type belongs in an event group. An event group contains all event types belonging to the standards body or vendor that defines them. The [SVH](#) identifies the event group. Within an event group, an event type ID identifies the event type uniquely within the event group. Both the SVH and the event type ID ensure uniqueness for all event types in this specification.

853 Usually, the Event Notifier does not need to support all event types within an event group or within all event groups. However, the standards body or vendor defines the requirements for the event types they define.

854 [Table 109 — GET_SUPPORTED_EVENT_TYPES request message format](#) describes the message format.

855 **Table 109 — GET_SUPPORTED_EVENT_TYPES request message format**

Byte Offset	Field	Size (bytes)	Description
0	<code>SPDMVersion</code>	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	<code>RequestResponseCode</code>	1	<code>0xE2</code> = <code>GET_SUPPORTED_EVENT_TYPES</code>
2	<code>Param1</code>	1	Reserved.
3	<code>Param2</code>	1	Reserved.

856 [Table 110 — SUPPORTED_EVENT_TYPES response message format](#) describes the message format for this response.

857 **Table 110 — SUPPORTED_EVENT_TYPES response message format**

Byte Offset	Field	Size (bytes)	Description
0	<code>SPDMVersion</code>	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	<code>RequestResponseCode</code>	1	<code>0x62</code> = <code>SUPPORTED_EVENT_TYPES</code> Response
2	<code>Param1</code>	1	<code>EventGroupCount</code> . Shall be the number of event groups listed in <code>SupportedEventGroupsList</code> .
3	<code>Param2</code>	1	Reserved.

Byte Offset	Field	Size (bytes)	Description
4	SupportedEventGroupsListLen	4	The value of this field shall be the size in bytes of the <code>SupportedEventGroupsList</code> and shall be greater than zero.
8	SupportedEventGroupsList	<code>SupportedEventGroupsListLen</code>	<p>Shall be a list of all event types grouped by event group supported by the Event Notifier. The format of this field shall be a list of Event group. In this format, each event group contains a list of event types the Event Notifier supports. If an event group is present, it shall be present exactly once to avoid duplicates and to minimize the size of this response. The size of this field shall be the value in <code>SupportedEventGroupsListLen</code>.</p> <p>See Event group format additional information for additional details.</p>

858 [Table 111 — Event group format](#) defines the format for listing event types in a single event group.

859 **Table 111 — Event group format**

Byte Offset	Field	Size (bytes)	Description
0	EventGroupId	2 + <code>VendorIDLen</code>	Shall indicate the event group the event type belongs to. The format of this field shall be the SVH format. The size of this field shall be the size of the SVH.
2 + <code>VendorIDLen</code>	EventTypeCount	2	Shall be the total number of event types listed in the <code>EventTypeList</code> field and belonging to <code>EventGroupId</code> . The value of this field shall be greater than zero.
4 + <code>VendorIDLen</code>	EventGroupVer	2	Shall be the standards body or vendor-assigned version number that indicates the version of the event types belonging to <code>EventGroupId</code> .
6 + <code>VendorIDLen</code>	Attributes	4	Attributes. The format of this field shall be defined by the messages using this Event groups format. For the <code>SUPPORTED_EVENT_TYPES</code> response message, see Event group format additional information . For the <code>SUBSCRIBE_EVENT_TYPES</code> request message, see Additional subscription list information .
10 + <code>VendorIDLen</code>	EventTypeList	Variable	<p>Shall be a list of event types in this Event Group (<code>EventGroupId</code>). The value in <code>EventTypeCount</code> field shall indicate the number of event types in this list. The format of this field shall be a list of Event Type Information.</p> <p>If an event type is present, it shall be present exactly once.</p>

860 [Table 112 — Event type information format](#) defines the format for a single event type.

861 **Table 112 — Event type information format**

Byte Offset	Field	Size (bytes)	Description
0	EventTypeId	2	Shall be a numeric value that uniquely identifies this event type within the corresponding event group.
2	Reserved	2	Reserved.

862 The `EventGroupVer` field allows for updates to the event type list such as a new event type. An Event Notifier should add new event types to the end of the list.

863 10.28.1.1 Event group format additional information

864 This clause describes further information for various fields in the [Event groups format table](#). This format is present in more than one SPDM message.

865 Many fields in the [Event group format table](#) have different definitions depending on which SPDM message uses this table. For `SUBSCRIBE_EVENT_TYPES`, see [Additional subscription list information](#) for requirements on the Event group format.

866 The following requirements shall apply to the Event group format table contained in `SUPPORTED_EVENT_TYPES`.

- The value of `EventTypeCount` field shall be greater than zero.
- The presence of an event type in the `EventTypeList` field shall indicate that the Event Notifier can send events of this type.
- The value of `Attributes` shall be reserved.

867 10.28.2 SUBSCRIBE_EVENT_TYPES request and SUBSCRIBE_EVENT_TYPES_ACK response message

868 The `SUBSCRIBE_EVENT_TYPES` request and `SUBSCRIBE_EVENT_TYPES_ACK` response messages allow an Event Recipient to communicate the list of SPDM event types it is interested in receiving. This request replaces the current subscription list.

869 An event subscription is a list of all event types to which an Event Recipient subscribes. Thus, an Event Notifier shall send events when they occur to an Event Recipient if at least one event type is present in the event subscription of the corresponding Event Recipient.

870 To subscribe or unsubscribe to an event group, an Event Recipient shall send the `SUBSCRIBE_EVENT_TYPES` request message with a complete list of all event types to which the Event Recipient subscribes. An Event Notifier shall replace the current event subscription with the new subscription from the latest `SUBSCRIBE_EVENT_TYPES` message. If the new subscription contains an unsupported or invalid event type, the Responder should respond with an `ERROR` message of `ErrorCode=InvalidRequest`. If an Event Notifier supports multiple Event Recipients, the Event Notifier

shall support a unique event subscription list per session for each subscribed Event Recipient. The [SUBSCRIBE_EVENT_TYPES request message format](#) describes the message format.

871 **Table 113 — SUBSCRIBE_EVENT_TYPES request message format**

Byte Offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	<code>0xEF</code> = SUBSCRIBE_EVENT_TYPES
			SubscribeEventGroupCount. Shall be the number of event groups in <code>SubscribeList</code> .
2	Param1	1	A value of zero shall indicate that the Event Recipient no longer subscribes to any events. This is the equivalent of an empty event subscription or the removal of all event types in an event subscription. If the value of this field is zero, <code>SubscribeListLen</code> and <code>SubscribeList</code> fields shall be absent.
3	Param2	1	Reserved.
4	SubscribeListLen	4	The value of this field shall be the size in bytes of <code>SubscribeList</code> . The value of this field shall be greater than zero.
8	SubscribeList	<code>SubscribeListLen</code>	Shall be a list of event types grouped by event group that the Event Notifier supports and to which the Event Recipient is subscribing. The format of this field shall be a list of Event group . In this format, each event group contains a list of event types to which the Event Recipient subscribes. If an event group is present, it shall be present exactly once. The size of this field shall be the value in <code>SubscribeListLen</code> field. See Additional subscription list information for additional requirements.

872 **Table 114 — SUBSCRIBE_EVENT_TYPES_ACK response message format** describes the response format for the SUBSCRIBE_EVENT_TYPES request.

873 **Table 114 — SUBSCRIBE_EVENT_TYPES_ACK response message format**

Byte Offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	<code>0x6F</code> = SUBSCRIBE_EVENT_TYPES_ACK Response
2	Param1	1	Reserved.
3	Param2	1	Reserved.

874 For event types defined by this specification, see [DMTF event types](#).

875 10.28.2.1 Additional subscription list information

876 These clauses describe further information for various fields in `SubscribeList` whose format is the [Event group format](#).

877 The value of the `EventTypeCount` field shall be greater than or equal to zero. If `EventTypeCount` is zero, then `AllEventTypes` shall also be set.

878 The presence of an event type in the `EventTypeList` field shall subscribe the Event Recipient to that event type. Likewise, the absence of an event type in the `EventTypeList` field shall indicate that the Event Recipient does not or no longer subscribes to this event type. Additionally, the absence of an event group in the `SubscribeList` shall indicate that the Event Recipient does not or no longer subscribes to any event types in this event group.

879 The format of the `Attributes` field shall be as the [SUBSCRIBE_EVENT_TYPES request attributes format table](#) defines.

880 **Table 115 — SUBSCRIBE_EVENT_TYPES request attributes format**

Byte Offset	Bit Offset	Field	Description
0	0	<code>AllEventTypes</code>	If set, the Event Notifier shall subscribe the Event Recipient to all event types supported by the Event Notifier in the corresponding Event Group and the value of <code>EventTypeCount</code> shall be zero.
0	[7:1]	Reserved	Reserved
1	[7:0]	Reserved	Reserved
2	[7:0]	Reserved	Reserved
3	[7:0]	Reserved	Reserved

881 If an Event Recipient sets `AllEventTypes`, it can receive events of event types it does not understand. In this scenario, the Event Recipient shall respond with an `EVENT_ACK` message as [SEND_EVENT request and EVENT_ACK response message](#) describes and stop processing the unknown event type.

882 10.28.3 SEND_EVENT request and EVENT_ACK response message

883 To deliver subscribed events to an Event Recipient, the Event Notifier shall use the `SEND_EVENT` request message. This request can contain more than one event.

884 [Table 116 — SEND_EVENT request message format](#) describes this request.

885 **Table 116 — SEND_EVENT request message format**

Byte Offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in SPDM version .
1	RequestResponseCode	1	0xF0 = SEND_EVENT
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	EventCount	4	Shall be the number of elements in EventsList .
8	EventsList	Variable	Shall be a list of Event Data . The list should be sorted in numerically increasing event instance ID order. The size of this field shall be the size of this list.

886 [Table 117 — Event data table](#) describes the format for details of each event.

887 **Table 117 — Event data table**

Byte Offset	Field	Size (bytes)	Description
0	EventInstanceId	4	Shall be the event instance id for the event.
4	Reserved	4	Reserved.
8	EventGroupId	2 + VendorIDLen	Shall indicate the event group the event type belongs to. The format of this field shall be SVH format.
10 + VendorIDLen	EventTypeId	2	Shall be the numeric value identifying the event type of this event in EventGroupId .
12 + VendorIDLen	EventDetailLen	2	Shall be the length of EventDetail .
14 + VendorIDLen	EventDetail	Variable	Shall be the event-specific details of the event indicated by EventInstanceId , EventGroupId and EventTypeId . The format and further definition of this field is specific to the event type indicated by EventTypeId in the event group indicated by EventGroupId . For the DMTF event group, see Event type details for further information. The size of this field shall be the size of the event-specific details for this event.

888 [Table 118 — EVENT_ACK response message format](#) describes the format for the response.

889 **Table 118 — EVENT_ACK response message format**

Byte Offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in SPDM version .
1	RequestResponseCode	1	0x70 = EVENT_ACK Response

Byte Offset	Field	Size (bytes)	Description
2	Param1	1	Reserved.
3	Param2	1	Reserved.

890 The Event Notifier shall only send unacknowledged event instance IDs.

891 The size of `SEND_EVENT` data can exceed the `DataTransferSize` of the Event Recipient, especially if multiple events happen concurrently. While it is possible to use the [Large SPDM message transfer mechanism](#), the Event Notifier should try to divide the events into multiple `SEND_EVENT` requests to ensure efficient delivery of the events instead of combining all events into a single `SEND_EVENT` request.

892 An Event Notifier shall send a `SEND_EVENT` request with only the Event Lost event (`EventTypeId = EventLost`) as an indication that the original event was too big in size under any of these conditions:

- The Event Notifier does not support the [Large SPDM message transfer mechanism](#) and the `SEND_EVENT` request with only one event exceeds the `DataTransferSize` of the Event Recipient.
- The size of a `SEND_EVENT` request with only one event is greater than the `MaxSPDMmsgSize` of the Event Recipient.

893 The Event Notifier shall follow the requirements in [Timing requirements](#) as a Requester for `SEND_EVENT` . Likewise, the Event Recipient shall follow the timing requirements as a Responder when receiving a `SEND_EVENT` request.

894 10.28.4 Event Instance ID

895 Event Instance ID typically reflects the order of events in the Event Notifier from a chronological perspective. The event instance ID shall start at zero for each secure session and sequentially increase with each occurrence of an event. This method also allows the Event Recipient to determine if an event was lost.

896 When the event instance ID reaches the maximum value, the Event Notifier shall terminate the session after sending a `SEND_EVENT` request containing an event with the maximum value and receiving the corresponding response. An Event Recipient can also terminate the session.

897 10.29 GET_ENDPOINT_INFO request and ENDPOINT_INFO response messages

898 The `GET_ENDPOINT_INFO` request message shall retrieve general information from an endpoint. The `SubCode` parameter is used to differentiate between operations, and a request message shall specify only one `SubCode` . If the Responder does not support the specified `SubCode` , the responder shall return an `ERROR` message of `ErrorCode=UnsupportedRequest` .

899 [Table 119 — GET_ENDPOINT_INFO request format](#) shows the format of the `GET_ENDPOINT_INFO` request message.

Table 122 — **ENDPOINT_INFO** response format shows the format of the **ENDPOINT_INFO** response message.

Table 119 — **GET_ENDPOINT_INFO** request format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in SPDM version .
1	RequestResponseCode	1	0x87 = GET_ENDPOINT_INFO . See Table 4 — SPDM request codes .
2	Param1	1	Shall be the GET_ENDPOINT_INFO SubCode. See GET_ENDPOINT_INFO SubCodes for the list of valid values.
3	Param2	1	Bit [7:4]. Reserved. Bit [3:0]. SlotID that identifies the certificate chain whose leaf certificate is used to sign the response. If a signature is not requested (Bit[0] of the RequestAttributes field is 0), this field shall be ignored. If the Responder's public key was provisioned to the Requester previously, this field shall be 0xF.
4	RequestAttributes	1	Request attributes. See GET_ENDPOINT_INFO request attributes .
5	Reserved	3	Reserved.
8	Nonce	NL = 32 or 0	The Requester should choose a random value. This field shall only be present if a signature is requested (SignatureRequested =1b).

Table 120 — **GET_ENDPOINT_INFO** SubCodes

SubCode	Value	Description
Reserved	0x00	Reserved.
DeviceClassIdentifier	0x01	The DeviceClassIdentifier response returns information that can be used to identify the class of device for the Responder in question. See ENDPOINT_INFO device class identifier list format for the definition of the response data.
Reserved	All other values	SPDM implementations compatible with this version shall not use the reserved SubCode s.

Table 121 — **GET_ENDPOINT_INFO** request attributes

Bit offset	Field	Description
0	SignatureRequested	<p>If the Responder can generate a signature (<code>EP_INFO_CAP=10b</code> in its <code>CAPABILITIES</code> response and either <code>BaseAsymSel</code> or <code>ExtAsymSelCount</code> is non-zero), a value of 1 indicates that a signature on the response is required. When this bit is set to 1, the Requester shall include the <code>Nonce</code> field in the request, and the Responder shall generate a signature and send the signature in the response.</p> <p>A value of 0 indicates that the Requester does not require a signature. The Responder shall not generate a signature in the response. The <code>Nonce</code> field shall be absent in the request and response.</p> <p>For Responders that cannot generate a signature (<code>EP_INFO_CAP=01b</code> in their <code>CAPABILITIES</code> response or both <code>BaseAsymSel</code> and <code>ExtAsymSelCount</code> are zero), the Requester shall always set this bit to 0.</p>
[7:1]	Reserved	Reserved.

904 **Table 122 — ENDPOINT_INFO response format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	<code>0x07</code> = <code>ENDPOINT_INFO</code> . See Table 5 — SPDM response codes .
2	Param1	1	Reserved.
3	Param2	1	<p>Bit [7:4]. Reserved.</p> <p>Bit [3:0]. <code>SlotID</code> that identifies the certificate chain whose leaf certificate is used to sign the response. If a signature is not requested (<code>SignatureRequested=0b</code>), this field shall be 0. If the Responder's public key was provisioned to the Requester previously, this field shall be <code>0xF</code>.</p>
4	Reserved	4	Reserved.
8	Nonce	NL = 32 or 0	The Responder should choose a random value. This field shall only be present if Bit[0] of the <code>RequestAttributes</code> field is 1.
8 + NL	EPInfoLen	4	Shall contain the length of the <code>EPInfo</code> field.
12 + NL	EPInfo	EPInfoLen	Shall contain endpoint information, as described in the endpoint information format for the specified <code>SubCode</code> . The size of this field shall be the size of the returned endpoint information.
12 + NL + EPInfoLen	Signature	SigLen	<p><code>Signature</code> of the endpoint information, excluding the <code>Signature</code> field and signed using the private key associated with the leaf certificate. The Responder shall use the asymmetric signing algorithm it selected during the last <code>ALGORITHMS</code> response message to the Requester, and <code>SigLen</code> is the output size for that asymmetric signing algorithm. This field is conditional and only present in the <code>ENDPOINT_INFO</code> response corresponding to a <code>GET_ENDPOINT_INFO</code> request with the <code>SignatureRequested</code> bit set to 1 in the <code>RequestAttributes</code> field. See ENDPOINT_INFO signature generation and ENDPOINT_INFO signature verification for more details.</p>

The Device Class Identifier format is an extended form of the [standards body or vendor-defined header](#). For a Device Class Identifier list response, `EPInfoLen` shall have a size of $4 + \text{IDElemSize}$. The `IDElemSize` shall be the sum of the sizes of the `NumIdentifiers` of the Device Class Identifier elements. Each Device Class Identifier shall have a size of $4 + \text{VendorIDLen} +$ the sum of the sizes of the subordinate Device Class Identifiers. Each of the subordinate Device Class Identifiers shall have a size of $1 + \text{SubIDLen}$, where `SubIDLen` may be different for each element.

Table 123 — ENDPOINT_INFO device class identifier list format

Byte offset	Field	Size (bytes)	Description
0	NumIdentifiers	1	Shall be the number of Device Class Identifier elements in this response message. Each identifier shall be unique.
1	Reserved	3	Reserved.
4	IdentifierElements	IDElemSize	Shall contain Device Class Identifier elements, as defined in ENDPOINT_INFO device class identifier element format .

Table 124 — ENDPOINT_INFO device class identifier element format

Byte offset	Field	Size (bytes)	Description
0	IDElemLength	1	Shall be the size of this ID element. The value of <code>IDElemLength</code> shall be the number of bytes from the <code>SVH.ID</code> field through the last <code>SubordinateID</code> , inclusive.
1	SVH	$2 + \text{VendorIDLen}$	Shall be a standards body or vendor-defined header , as described in Table 64 — Standards body or vendor-defined header (SVH) .
$3 + \text{VendorIDLen}$	NumSubIDs	1	Shall be the number of subordinate Device Class Identifiers.
$4 + \text{VendorIDLen}$	SubordinateID	NumSubIDs entries of $1 + \text{SubIDLen}$ for a given entry	Shall contain <code>NumSubIDs</code> of subordinate Device Class Identifiers, of the format described in Device class identifier subordinate identifier format . If <code>NumSubIDs</code> is 0, this field shall be absent.

If present, one or more subordinate identifier fields contain identifiers that further identify the device. These identifiers shall be valid in the namespace defined by the standards body specified in the `ID` field and by the vendor ID specified in the `VendorID` field.

Table 125 — Device class identifier subordinate identifier format

Byte offset	Field	Size (bytes)	Description
0	SubIDLen	1	Shall contain the length in bytes of this subordinate identifier.

Byte offset	Field	Size (bytes)	Description
1	SubIdentifier	SubIDLen	Shall contain one subordinate device identifier that is valid in the namespace of the vendor identified in the <code>VendorID</code> field. This field shall be size <code>SubIDLen</code> .

910 10.29.1 ENDPOINT_INFO signature generation

911 The signature for an `ENDPOINT_INFO` response is generated per request and response pair. To complete the `ENDPOINT_INFO` signature generation process, the Responder shall complete these steps:

- 912 1. The Responder shall construct an information log IL1, and the Requester shall construct an information log IL2 over their observed messages:

```
IL1/IL2 = Concatenate(VCA, GET_ENDPOINT_INFO, ENDPOINT_INFO)
```

913 where:

- `Concatenate` is the standard concatenation function.
- `GET_ENDPOINT_INFO` is the entire `GET_ENDPOINT_INFO` request message under consideration where the Requester has set the `SignatureRequested` bit in the `RequestAttributes` field.
- `ENDPOINT_INFO` is the entire `ENDPOINT_INFO` response message under consideration, except for the signature field.

- 914 2. The Responder shall generate:

```
Signature = SPDMSign(PrivKey, IL1, "endpoint_info signing")
```

915 where:

- `SPDMSign` is described in [Signature generation](#).
- `PrivKey` shall be the private key of the Responder associated with the leaf certificate stored in `SlotID` of `Param2` in `GET_ENDPOINT_INFO`. If the public key of the Responder was provisioned to the Requester, then `PrivKey` shall be the associated private key.

916 10.29.2 ENDPOINT_INFO signature verification

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

- 918 1. The Requester shall perform:

```
result = SPDMsignatureVerify(PubKey, Signature, IL2, "endpoint_info signing")
```

919 where:

- `SPDMsignatureVerify` is described in [Signature verification](#). A successful verification is when `result` is `success`.
- `PubKey` shall be the public key associated with the leaf certificate stored in `SlotID` of `Param2` in `GET_ENDPOINT_INFO`, and it is extracted from the `CERTIFICATE` response. If the public key of the Responder was provisioned to the Requester, then `PubKey` shall be the provisioned public key.

920 10.30 Measurement extension log mechanism

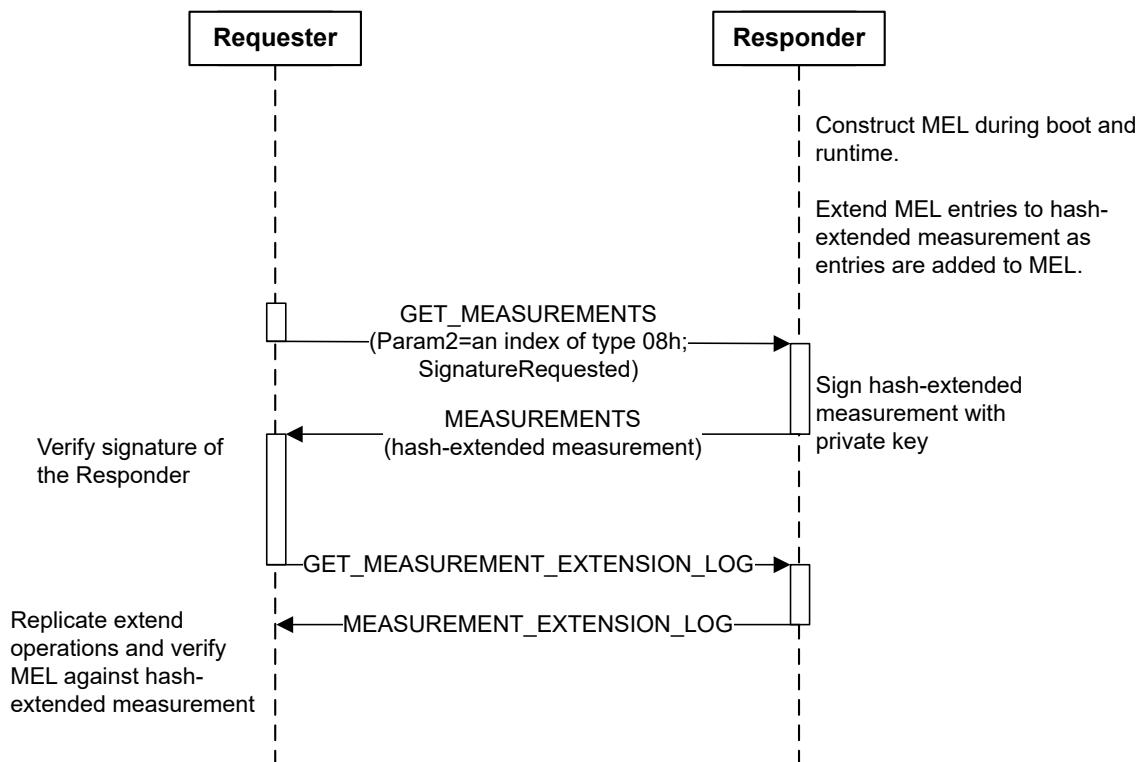
921 A Responder device may create and maintain a Measurement Extension Log (MEL) to record device information such as measurements of firmware and/or software modules loaded during the boot, firmware and/or software updates, configurations, status of the system, and so on. To construct the MEL, when certain events occur, the Responder appends data associated with the events to the end of the MEL. The events that cause the MEL update are specific to and are determined by individual Responder implementations. For example, the Responder may append the digest and version number of a firmware module to the end of the MEL when the firmware module is loaded. The MEL grows as entries are added. At reset, the Responder may reset the MEL or preserve the MEL. If the Responder preserves the MEL across resets, the reset events themselves may be added as new entries to the MEL. Accordingly, the corresponding HEM should also be preserved across resets. The Responder should ensure that the MEL will not overrun memory or wrap under normal uses.

922 If the `MEL_CAP` bit in `CAPABILITIES` is set, the Requester may acquire the MEL of the Responder by issuing a `GET_MEASUREMENT_EXTENSION_LOG` request message. The Responder shall respond with the `MEASUREMENT_EXTENSION_LOG` response message. If a Requester acquires the [hash-extend measurements](#) outside of a secure session, the Requester should set `SignatureRequested=1` in the `GET_MEASUREMENTS` request or secure the response using other means outside of this specification.

923 The [Hash-extend measurements](#) clause introduces a method of constructing a hash value (type `0x8` of `DMTFSpecMeasurementValueType[6:0]`) by extending measurements. The resulting hash guarantees the integrity of the data participating in the extend operations. Leveraging this mechanism can ensure the integrity of the MEL. To do this, an entry of the MEL serves as the `DataToExtend` in calculating `HEM`. After all entries of the MEL are processed, the resulting `HEM` is the hash-extend measurement.

924 To avoid circular dependencies and race conditions, the `DataToExtend` for calculating HEM shall not include the `GET_MEASUREMENTS` request, `MEASUREMENTS` response, `GET_MEASUREMENT_EXTENSION_LOG` request, or `MEASUREMENT_EXTENSION_LOG` response messages.

925 [Figure 27 — Flow for acquiring Hash-Extend Measurement and Measurement Extension Log](#) demonstrates an example flow for the Requester to obtain hash-extend measurement and the MEL from the Responder.

Figure 27 — Flow for acquiring Hash-Extend Measurement and Measurement Extension Log

As the example flow shows, a Responder that supports MEL would construct the MEL at runtime independently of the Requester. The Requester would first issue `GET_MEASUREMENTS` to obtain the hash-extend measurement and verify the signature of the Responder, and then it would issue `GET_MEASUREMENT_EXTENSION_LOG` to obtain the MEL from the Responder. With both hash-extend measurement and MEL, the Requester replicates the extend operations with entries of the MEL in ascending MEL index order for the corresponding HEM received in the `MEASUREMENT_EXTENSION_LOG`. If the result of extend operations does not match the hash-extend measurement, then it indicates that the verification of HEM has failed.

10.30.1 `GET_MEASUREMENT_EXTENSION_LOG` request and `MEASUREMENT_EXTENSION_LOG` response messages

Table 126 — `GET_MEASUREMENT_EXTENSION_LOG` message format shows the `GET_MEASUREMENT_EXTENSION_LOG` request message format.

Table 127 — Successful `MEASUREMENT_EXTENSION_LOG` message format shows the `MEASUREMENT_EXTENSION_LOG` response message format.

932 **Table 126 — GET_MEASUREMENT_EXTENSION_LOG message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	<code>0xEF</code> = GET_MEASUREMENT_EXTENSION_LOG . See Table 4 — SPDM request codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	Offset	4	Shall be the offset in bytes from the start of the MEL to where the read request message begins. The Responder shall send the MEL starting from this offset. Offset 0 shall be the first byte of the MEL.
8	Length	4	Shall be the length of the MEL, in bytes, to be returned in the corresponding response.

933 Note that the [large SPDM message transfer mechanism](#) can be used for the MEASUREMENT_EXTENSION_LOG message.

934 **Table 127 — Successful MEASUREMENT_EXTENSION_LOG response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in SPDM version .
1	RequestResponseCode	1	<code>0x6F</code> = MEASUREMENT_EXTENSION_LOG . See Table 5 — SPDM response codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	PortionLength	4	Shall be the number of bytes of this portion of the MEL. This shall be less than or equal to the <code>Length</code> received as part of the request. For example, the Responder might set this field to a value less than the <code>Length</code> received as part of the request due to limitations on the transmit buffer of the Responder.
8	RemainderLength	4	Shall be the number of bytes remaining in the MEL from the requested offset + <code>PortionLength</code> . A value of 0 shall indicate there are no more bytes beyond the requested offset + <code>PortionLength</code> .
12	MEL	<code>PortionLength</code>	Requested contents of the MEL. This field shall follow the format negotiated in the most recent ALGORITHM message.

935 10.30.2 DMTF Measurement Extension Log Format

936 This clause specifies the format of MEL in the `MEASUREMENT_EXTENSION_LOG` response when the MEL specification (`MELSpecificationSel`) is "DMTFmelSpec" and the measurement specification (`MeasurementSpecificationSel`) is

"DMTFmeasSpec" in the most recent `ALGORITHMS` message (see [Table 21 — Successful ALGORITHMS response message format](#)). The MEL format shown in [Table 128 — DMTF Measurement Extension Log format](#) leverages the [DMTF measurement specification format](#) for its entries.

937 **Table 128 — DMTF Measurement Extension Log Format**

Byte offset	Field	Size (bytes)	Description
0	NumberOfEntries	4	Shall be the number of entries in the MEL.
4	MELEntriesLength	4	Shall be the total number of bytes in all entries of the MEL.
8	Reserved	8	Reserved.
16	MELEntries	MELEntriesLength	Shall be the concatenation of all entries of the MEL. The size of this field shall be equal to MELEntriesLength .

938 The MELEntries field of the DMTF Measurement Extension Log consists of all entries of the MEL. Each MEL entry shall follow the format that [Table 129 — DMTF Measurement Extension Log Entry Format](#) defines. In the calculation of [hash-extend measurement](#), `DataToExtend` shall be one MEL entry at a time.

939 **Table 129 — DMTF Measurement Extension Log Entry Format**

Byte offset	Field	Size (bytes)	Description
0	MELIndex	4	Shall be the index of this entry in the MEL. This field shall be a non-negative integer. The MELIndex shall be in increasing order.
4	MeasIndex	1	Shall be the index of the hash-extend measurement which this entry extends, that is, the <code>Index</code> of Table 53 — Measurement block format for this hash-extend measurement (<code>DMTFSpecMeasurementValueType[6:0] = 0x8</code>) in the <code>MEASUREMENTS</code> response. MeasIndex values of MEL entries can interleave. For example, it is legitimate that a MELIndex of 2 has a MeasIndex of 0x04, but a MELIndex of 1 and a MELIndex of 3 both have a MeasIndex of 0x05. If this entry does not extend to any index, then the Responder shall set this field to 0x00 . In this case, the entry shall not be used in the extend operation for calculating HEM. Some indices are reserved for specific purpose (see Table 51 — Measurement index assigned range).
5	Reserved	3	Reserved.
8	Entry	DMTFSpecMeasurementValueSize + 3	Shall be the entry data of the DMTF measurement specification format .

10.30.3 Example: Verifying Measurement Extension Log Against Hash-Extend Measurement

Figure 28 — Example for Measurement Extension Log illustrates an example of an MEL with 11 entries and two corresponding hash-extend measurements at MEASUREMENTS response indices 1 and 2 to which the log entries extend. The MEL in this example is constructed by the Responder during boot. The Responder implements a simple ROM–firmware secure boot architecture.

Figure 28 — Measurement Extension Log Example

SPDM MEASUREMENTS response			DMTF Measurement Extension Log MELEntries Field			
Index	DMTFMeasurementValue Type	DMTFMeasurementValue	MEL Index	Meas Index	reserved	“Entry” following Table “DMTF Measurement SpecificationFormat”
1	08h (HEM)	<digest-1>	1	0	0	89h: raw bits; informational 3 “ROM”
2	08h (HEM)	<digest-2>	2	1	0	00h: digest; ROM 48 <digest of ROM>
			3	1	0	82h: raw bits; hardware config 128 <hardware config data of ROM>
			4	0	0	89h: raw bits; informational 7 “Boot FW”
			5	2	0	87h: raw bits; security version 8 0x0000000000000002
			6	2	0	86h: raw bits; version 4 0x0100030A
			7	2	0	01h: digest; firmware 48 <digest of boot firmware>
			8	0	0	89h: raw bits; informational 9 “ROM patch”
			9	1	0	00h: digest; ROM 48 <digest of ROM patch>
			10	0	0	89h: raw bits; informational 14 “Application FW”
			11	2	0	01h: digest; firmware 48 <digest of application firmware>

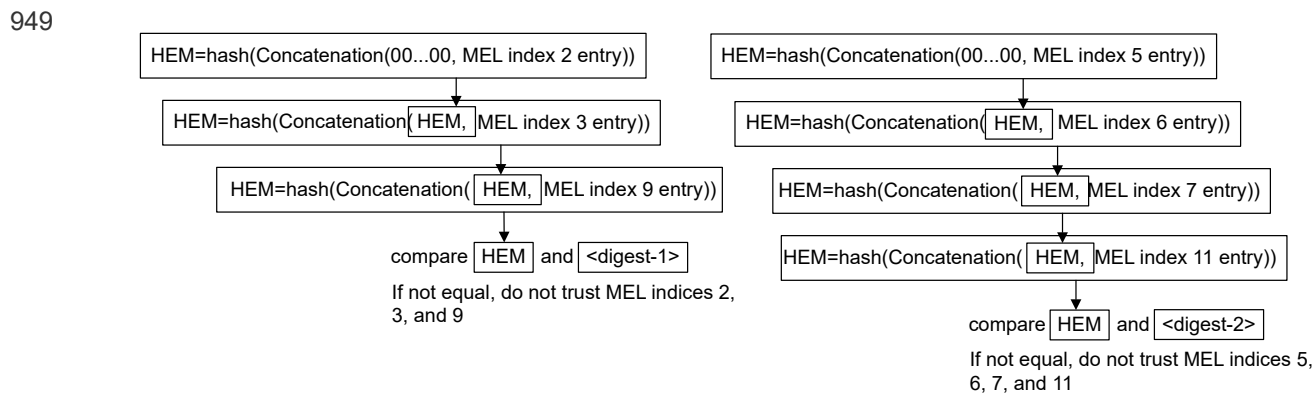
The MEL entries of indices 1, 4, 8, and 10 have a value type of 0x9 (informational). Since these are informational and do not apply to any measurement index, they are ignored in calculating HEM.

The hash-extend measurement at MEASUREMENTS index 1 is used for recording digests of ROM, patch, and hardware configuration. The MEL entries with MEL indices 2, 3, and 9 fit in this category and they extend to MEASUREMENTS index 1. Note that an extend operation shall consume the entire entry, including MELIndex, MeasIndex, Reserved, and Entry.

The hash-extend measurement at MEASUREMENTS index 2 is used for recording the digest of the firmware, firmware configuration, and version information. The MEL entries with MEL indices 5, 6, 7, and 11 fit in this category, and they extend to MEASUREMENTS index 2.

947 The Requester verifies the MEL entries by performing the checks illustrated in [Figure 29 — Example for Verifying Measurement Extension Log Entries](#).

948 **Figure 29 — Example for Verifying Measurement Extension Log Entries**



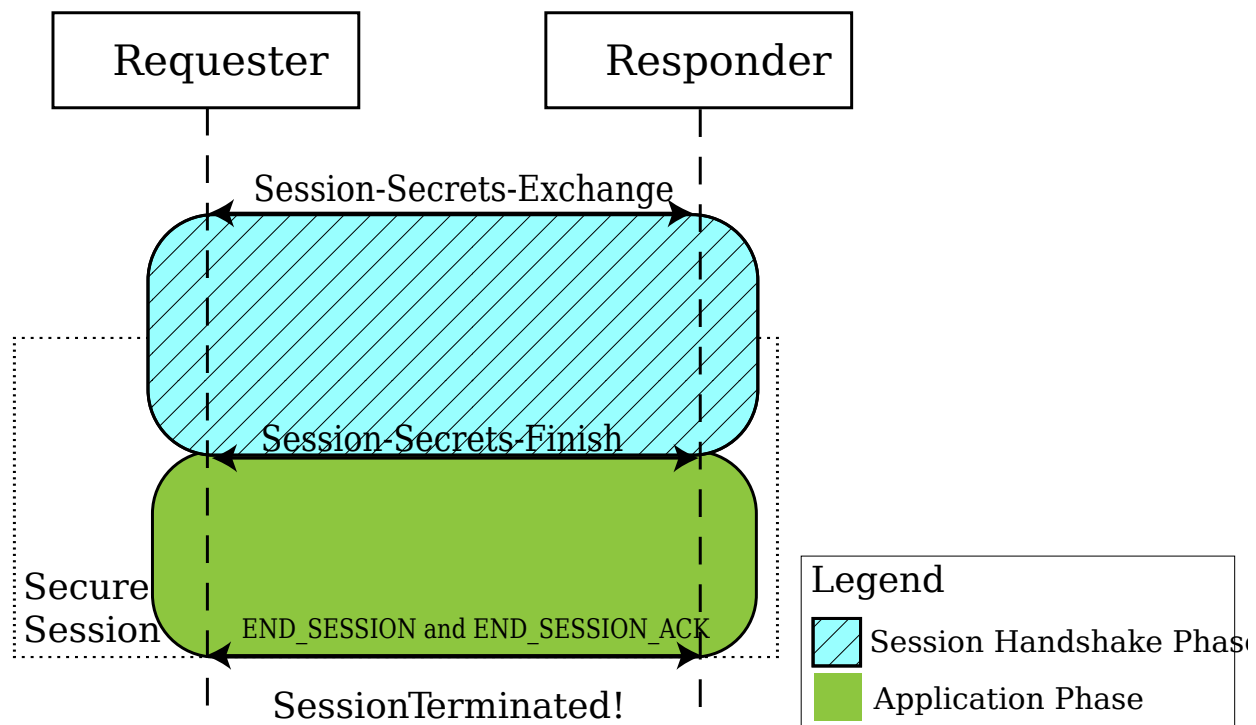
11 Session

Sessions enable a Requester and Responder to have multiple channels of communication. More importantly, it enables a Requester and Responder to build a secure communication channel with cryptographic information that is bound ephemerally. Specifically, an SPDM session provides either encryption or message authentication or both.

A session has three phases, as [Figure 30 — Session phases](#) shows:

- The handshake
- The application
- Termination

Figure 30 — Session phases



11.1 Session handshake phase

The session handshake phase begins with either `KEY_EXCHANGE` or `PSK_EXCHANGE`. This phase also allows for the authentication of the Requester if the Responder indicated this earlier in its `ALGORITHMS` response. Furthermore, this

phase of the session uses the handshake secrets to secure the communication as described in the [Key schedule](#) clause.

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

958 In this phase of the session, `GET_ENCAPSULATED_REQUEST` and `DELIVER_ENCAPSULATED_RESPONSE` shall be used to obtain requests from the Responder to complete the authentication of the Requester, if the Responder indicated this in its `ALGORITHMS` response. During this phase, the Responder shall not asynchronously send requests to the Requester. The only requests allowed to be encapsulated shall be `GET_DIGESTS` and `GET_CERTIFICATE`. The Requester shall provide a signature in the `FINISH` request, as the [FINISH request and FINISH_RSP response messages](#) clause describes.

959 If an `ERROR` message of `ErrorCode=DecryptError` occurs in this phase, the session shall immediately terminate and proceed to session termination.

960 A successful handshake ends with either `FINISH_RSP` or `PSK_FINISH_RSP` and the application phase begins.

961 11.2 Application phase

962 Once the handshake completes and all validation passes, the session reaches the application phase where either the Responder or the Requester can send application data.

963 During this phase, a Requester can send SPDM messages such as `GET_MEASUREMENTS`. These messages might involve transcript calculations. If such calculations are required, they shall be calculated on a per session basis. Once a session has been established, subsequent messages sent outside of a session shall not contribute to the transcript within a session.

964 The application phase ends when the `HEARTBEAT` requirements fail, or with an `END_SESSION` message, or with an `ERROR` message of `ErrorCode=DecryptError`. The next phase is the session termination phase.

965 11.3 Session termination phase

966 This phase signals the end of the application phase and the enactment of internal clean-up procedures by the endpoints. Requesters and Responders can have various reasons for terminating a session, which are outside the scope of this specification.

967 SPDM provides the `END_SESSION` / `END_SESSION_ACK` message pair to explicitly trigger the session termination phase if needed but, depending on the transport, it might simply be an internal phase with no explicit SPDM messages sent or received.

968 When a session terminates, both Requester and Responder shall destroy or clean up all session secrets such as

derived major secrets, DHE secrets and encryption keys. Endpoints might have other internal data associated with a session that they should also clean up.

969 11.4 Simultaneous active sessions

970 If a Responder supports key exchanges, the maximum number of simultaneous active sessions shall be at least 1. If a `KEY_EXCHANGE` or `PSK_EXCHANGE` request would cause the Responder's number of simultaneous active sessions to exceed this maximum, the Responder shall respond with an `ERROR` message of `ErrorCode=SessionLimitExceeded`.

971 This specification does not prohibit concurrent sessions in which the same Requester and Responder reverse roles. For example, SPDM endpoint ABC, acting as a Requester, can establish a session to SPDM endpoint XYZ, which is acting as a Responder. At the same time, SPDM endpoint XYZ, now acting as a Requester, can establish a session to SPDM endpoint ABC, now acting as a Responder. Because these two sessions are distinct and separate, the two endpoints would ensure they do not mix sessions. To ensure proper session handling, each endpoint would ensure that their portion of the session IDs are unique at the time of Session-Secrets-Exchange. This would form a final unique session ID for that new session. Additionally, the endpoints can use information at the transport layer to further ensure proper handling of sessions.

972 11.5 Records and session ID

973 When the session starts, the communication of secured data is done using records. A record represents a chunk or unit of data that is either encrypted or authenticated or both. This data can be either an SPDM message or application data. Usually, the record contains the session ID resulting from one of the Session-Secrets-Exchange messages to aid both the Responder and Requester in binding the record to the respective derived session secrets.

974 The actual format and other details of a record are outside the scope of this specification. It is generally assumed that the transport protocol will define the format and other details of the record.

12 Key schedule

A key schedule describes how the various keys such as encryption keys used by a session are derived and when each key is used. The default SPDM key schedule makes heavy use of `HKDF-Extract` and `HKDF-Expand`, which [RFC 5869](#) describes. SPDM defines this additional function:

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

where

- `BinConcat` shall be the concatenation of binary data in the order that [Table 130 — BinConcat details](#) shows:

Table 130 — BinConcat details

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

If `Context` is `null`, `BinConcat` is the concatenation of the first three components only.

[Table 131 — Version details](#) describes the version details.

Table 131 — Version details

SPDM version	Version text
SPDM 1.1	"spdm1.1 "
SPDM 1.2	"spdm1.2 "
SPDM 1.3	"spdm1.3 "

The `HKDF-Expand` function prototype as used by the default SPDM key schedule is as follows:

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

983 The HKDF-Extract function prototype is described as follows:

```
HKDF-Extract(salt, IKM);
```

984 where

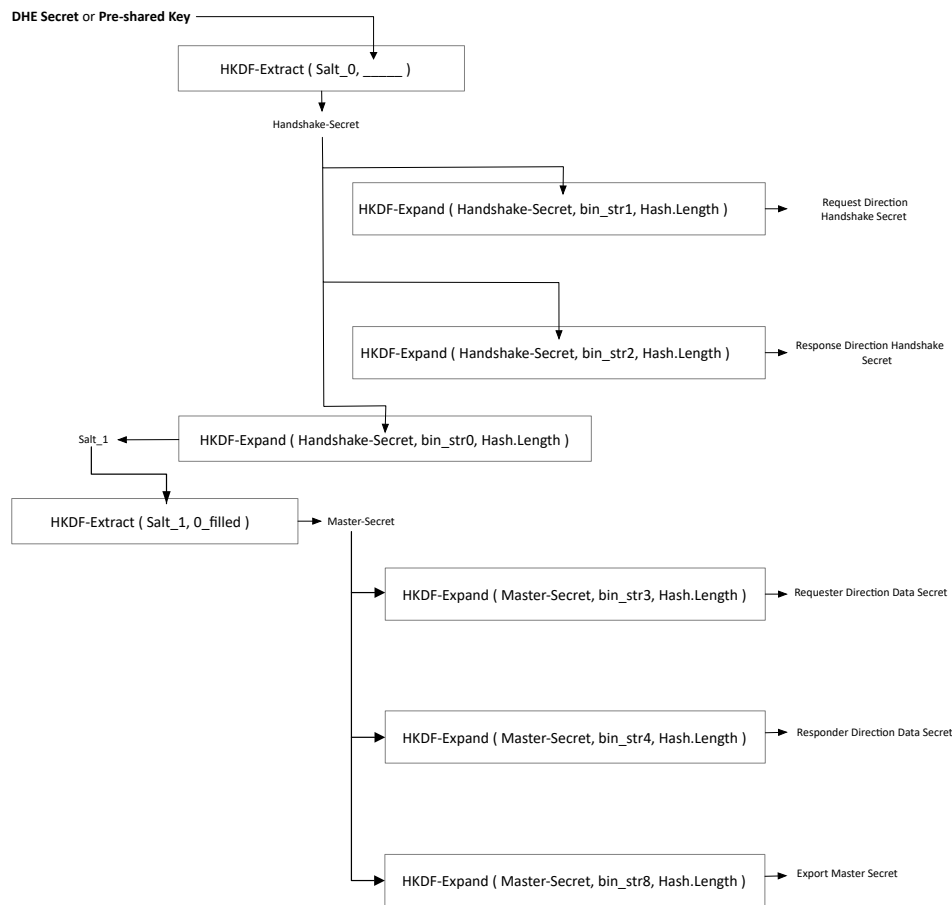
- IKM is the Input Keying Material.

985 For HKDF-Expand and HKDF-Extract, the hash function shall be the selected hash function in the ALGORITHMS response. Hash . Length shall be the length of the output of the hash function selected by the ALGORITHMS response.

986 Both Responder and Requester shall use the key schedule that [Figure 31 — Key schedule](#) shows.

987 **Figure 31 — Key schedule**

988



989 In the figure, arrows going out of the box are outputs of that box. Arrows going 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 names for clarity.

990 [Table 132 — Key schedule](#) accompanies the figure to complete the key schedule. The Responder and Requester shall also adhere to the definition of this table.

991 **Table 132 — Key schedule**

Variable	Definition	Value is secret?
Salt_0	A zero-filled array of Hash.Length length for KEY_EXCHANGE session. A 0xFF-filled array of Hash.Length length for PSK_EXCHANGE session.	No
Salt_1	Used to generate the Master-Secret.	Yes

Variable	Definition	Value is secret?
0_filled	A zero-filled array of length <code>Hash.Length</code> .	No
bin_str0	<code>BinConcat(Hash.Length, Version, "derived", NULL)</code>	No
bin_str1	<code>BinConcat(Hash.Length, Version, "req hs data", TH1)</code>	No
bin_str2	<code>BinConcat(Hash.Length, Version, "rsp hs data", TH1)</code>	No
bin_str3	<code>BinConcat(Hash.Length, Version, "req app data", TH2)</code>	No
bin_str4	<code>BinConcat(Hash.Length, Version, "rsp app data", TH2)</code>	No
DHE Secret	This shall be the secret derived from <code>KEY_EXCHANGE/KEY_EXCHANGE_RSP</code> .	Yes
Pre-Shared Key	PSK	Yes

992 **Note:** With common hash functions, any label longer than 12 characters requires an additional iteration of the hash function to compute. As in [RFC 8446](#), the previously defined labels have all been chosen to fit within this limit.

993 12.1 DHE secret computation

994 The DHE secret is a shared secret, and its computation is different per algorithm or algorithm class. These clauses define the format and computation for DHE algorithms.

995 For `ffdhe2048`, `ffdhe3072`, `ffdhe4096`, `secp256r1`, `secp384r1`, and `secp521r1`, the format and computation of the DHE secret shall be the shared secret, which section 7.4 of [RFC 8446](#) defines.

996 For `SM2_P256`, the parameters of this curve are defined in the [TCG Algorithm Registry](#). The DHE secret shall be K_A and K_B as defined in [GB/T 32918.3-2016](#). The Requester shall compute K_A , and the Responder shall compute K_B to arrive at the same secret value. K_A and K_B are the results of a KDF. This specification shall use the KDF as defined by [GB/T 32918.3-2016](#). The size of the DHE secret, referred to as `klen` in the KDF of [GB/T 32918.3](#) specification, shall be the key size of the selected AEAD algorithm in `RespAlgStruct`. Lastly, [GB/T 32918.3](#) allows for a flexible hash algorithm. The hash algorithm shall be the selected hash algorithm in `BaseHashSel` or `ExtHashSel`.

997 12.2 Transcript hash in key derivation

998 The key schedule uses two transcript hashes:

- **TH1**
- **TH2**

999 12.3 TH1 definition

1000 If the Requester and Responder used `KEY_EXCHANGE` / `KEY_EXCHANGE_RSP` to exchange initial keying information, **TH1** shall be the output of applying the negotiated hash function to the concatenation of the following:

1. `VCA`
2. `[DIGESTS] . *` (if issued and if `MULTI_KEY_CONN_RSP` is true).
3. Hash of the specified certificate chain in DER format (that is, `Param2` of `KEY_EXCHANGE`) or hash of the public key in its provisioned format, if a certificate is not used.
4. `[KEY_EXCHANGE] . *`
5. `[KEY_EXCHANGE_RSP] . *` except for the `ResponderVerifyData` field

1001 If the Requester and Responder used `PSK_EXCHANGE` / `PSK_EXCHANGE_RSP` to exchange initial keying information, **TH1** shall be the output of applying the negotiated hash function to the concatenation of the following:

1. `VCA`
2. `[PSK_EXCHANGE] . *`
3. `[PSK_EXCHANGE_RSP] . *` except for the `ResponderVerifyData` field

1002 12.4 TH2 definition

1003 If the Requester and Responder used `KEY_EXCHANGE` / `KEY_EXCHANGE_RSP` to exchange initial keying information, **TH2** shall be the output of applying the negotiated hash function to the concatenation of the following:

1. `VCA`
2. `[DIGESTS] . *` (if issued and if `MULTI_KEY_CONN_RSP` is true).
3. Hash of the specified certificate chain in DER format (that is, `Param2` of `KEY_EXCHANGE`) or hash of the public key in its provisioned format, if a certificate is not used.
4. `[KEY_EXCHANGE] . *`
5. `[KEY_EXCHANGE_RSP] . *`
6. `[DIGESTS] . *` (if encapsulated `DIGEST` is issued and if `MULTI_KEY_CONN_REQ` is true).
7. Hash of the specified certificate chain in DER format (that is, `Param2` of `FINISH`) or hash of the public key in its provisioned format, if a certificate is not used. (Valid only in mutual authentication)
8. `[FINISH] . *`
9. `[FINISH_RSP] . *`

1004 If the Requester and Responder used `PSK_EXCHANGE` / `PSK_EXCHANGE_RSP` to exchange initial keying information, **TH2** shall be the output of applying the negotiated hash function to the concatenation of the following:

1. VCA
2. [PSK_EXCHANGE] . *
3. [PSK_EXCHANGE_RSP] . *
4. [PSK_FINISH] . * (if issued)
5. [PSK_FINISH_RSP] . * (if issued)

1005 12.5 Key schedule major secrets

1006 The key schedule produces four major secrets:

- Request-direction handshake secret (S_0)
- Response-direction handshake secret (S_1)
- Request-direction data secret (S_2)
- Response-direction data secret (S_3)

1007 Each secret applies in a certain direction of transmission and is only valid during a certain time frame. Each of these four major secrets will be used to derive their respective encryption keys and IV values to be used in the AEAD function as selected in the ALGORITHMS response.

1008 12.5.1 Request-direction handshake secret

1009 This secret shall only be used during the session handshake phase and shall be applied to all requests after KEY_EXCHANGE or PSK_EXCHANGE up to and including FINISH or PSK_FINISH .

1010 12.5.2 Response-direction handshake secret

1011 This secret shall only be used during the session handshake phase and shall be applied to all responses after KEY_EXCHANGE_RSP or PSK_EXCHANGE_RSP up to and including FINISH_RSP or PSK_FINISH_RSP .

1012 12.5.3 Requester-direction data secret

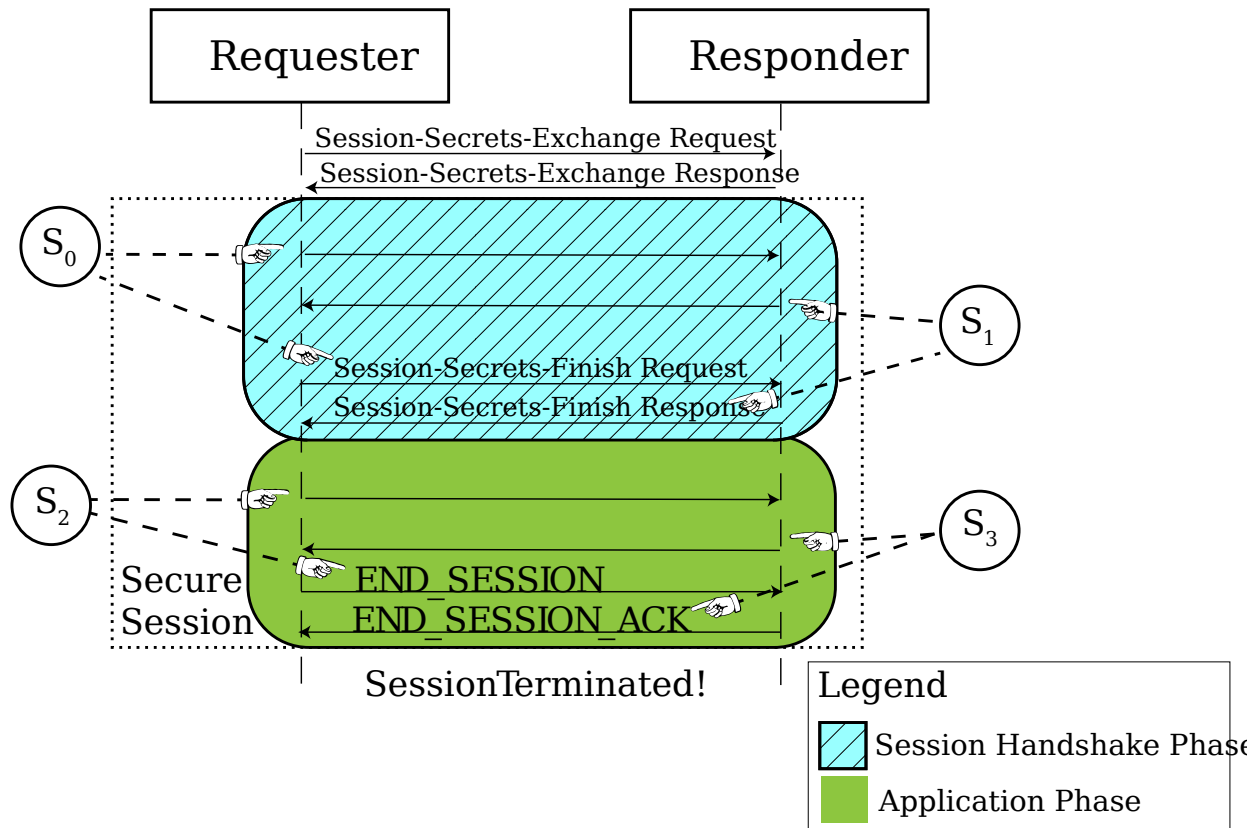
1013 This secret shall be used for any data transmitted during the application phase of the session. This secret shall only be applied for all data traveling from the Requester to the Responder.

1014 12.5.4 Responder-direction data secret

1015 This secret shall be used for any data transmitted during the application phase of the session. This secret shall only be applied for all data traveling from the Responder to the Requester.

1016 [Figure 32 — Secrets usage](#) illustrates where each of the major secrets are used, as described previously.

Figure 32 — Secrets usage



12.6 Encryption key and IV derivation

For each key schedule major secret, the following function shall be applied to obtain the encryption key and IV value.

```
EncryptionKey = HKDF-Expand(major-secret, bin_str5, key_length);
IV = HKDF-Expand(major-secret, bin_str6, iv_length);

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

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

1022 12.7 finished_key derivation

1023 This key shall be used to compute the `RequesterVerifyData` and `ResponderVerifyData` fields 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(Hash.Length, Version, "finished", NULL);
```

1024 The handshake-secret shall be either a request-direction handshake secret or a response-direction handshake secret.

1025 12.8 Deriving additional keys from the Export Master Secret

1026 After a successful SPDM key exchange, additional keys can be derived from the Export Master Secret. How keys are derived from this secret is outside the scope of this specification. The Export Master Secret is not a [major secret](#) and is not updated through a [major secrets update](#). How the Export Master Secret is updated, if required, is outside the scope of this specification.

```
Export Master Secret = HKDF-Expand(Master-Secret, bin_str8, Hash.Length);
bin_str8 = BinConcat(Hash.Length, Version, "exp master", TH2);
```

1027 12.9 Major secrets update

1028 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.

1029 The major secrets shall be re-keyed as a result of this request. To compute the new secret for each new major data secret, the following algorithm shall be applied.

```
new_secret = HKDF-Expand(current_secret, bin_str9, Hash.Length);
bin_str9 = BinConcat(Hash.Length, Version, "traffic upd", NULL);
```

1030 In computing the new secret, `current_secret` shall be either the current Requester-Direction Data Secret or the 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.

1031 13 Application data

1032 SPDM utilizes authenticated encryption with associated data (AEAD) cipher algorithms in much the same way that TLS 1.3 does to protect the confidentiality and integrity of data that shall remain secret as well as to protect the integrity of data that needs to be transmitted in the clear but shall still be protected from manipulation, as is the case for protocol headers. AEAD algorithms provide both encryption and message authentication. Each algorithm specifies 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.

1033 AEAD functions shall provide the following functions and comply with the requirements defined in [RFC 5116](#):

```
AEAD_Encrypt(encryption_key, nonce, associated_data, plaintext);
AEAD_Decrypt(encryption_key, nonce, associated_data, ciphertext);
```

1034 where

- `AEAD_Encrypt` is the function that fully encrypts the `plaintext`, computes the MAC across both the `associated_data` and `plaintext`, and produces the `ciphertext`, which includes the MAC.
- `AEAD_Decrypt` is the function that verifies the MAC and, if validation is successful, fully decrypts the `ciphertext` and produces the original `plaintext`.
- `encryption_key` is the derived encryption key for the respective direction. See the [Key schedule](#) clause.
- `nonce` is the nonce computation. See the [Nonce derivation](#) clause.
- `associated_data` is the associated data.
- `plaintext` is the data to encrypt.
- `ciphertext` is the data to decrypt.

1035 13.1 Nonce derivation

1036 Certain AEAD ciphers have specific requirements for nonce construction because their security properties can be compromised by the accidental reuse of a nonce value. Implementations should follow the requirements, such as those provided in [RFC 5116](#) for nonce derivation.

14 General opaque data format

The general opaque data format allows for a variety of data defined by an assortment of vendors, standards bodies, and transport mechanisms to accompany an SPDM message without namespace collisions.

If the `OpaqueDataFmt1` bit is selected in `OtherParamsSelection` of `ALGORITHMS`, then all opaque data fields in SPDM messages shall use the format that [Table 133 — General opaque data format](#) defines.

Table 133 — General opaque data format

Byte offset	Field	Size (bytes)	Description
0	TotalElements	1	Shall be the total number of elements in <code>OpaqueList</code> .
1	Reserved	3	Reserved.
4	OpaqueList	Variable	Shall be a list of opaque elements. See Table 134 — Opaque element .

[Table 134 — Opaque element](#) defines the format for each element in `OpaqueList`.

Table 134 — Opaque element

Byte offset	Field	Size (bytes)	Description
0	ID	1	Shall be one of the values in the <code>ID</code> column of Table 60 — Registry or standards body ID .
1	VendorIDLen	1	Shall be the length in bytes of the <code>VendorID</code> field. If the data in <code>OpaqueElementData</code> belongs to a standards body, this field shall be 0. Otherwise, the data in <code>OpaqueElementData</code> belongs to the vendor and therefore, this field shall be the length indicated in the "Vendor ID length" column of Table 60 — Registry or standards body ID for the respective <code>ID</code> .
2	VendorID	<code>VendorIDLen</code>	If <code>VendorIDLen</code> is greater than zero, this field shall be the ID of the vendor corresponding to the <code>ID</code> field. Otherwise, this field shall be absent.
2 + <code>VendorIDLen</code>	OpaqueElementDataLen	2	Shall be the length of <code>OpaqueElementData</code> .
4 + <code>VendorIDLen</code>	OpaqueElementData	<code>OpaqueElementDataLen</code>	Shall be the data defined by the vendor or standards body.

Byte offset	Field	Size (bytes)	Description
$4 + \text{VendorIDLen} + \text{OpaqueElementDataLen}$	AlignPadding	$\text{AlignPaddingSize} = 0, 1, 2, \text{ or } 3$	If $4 + \text{VendorIDLen} + \text{OpaqueElementDataLen}$ does not fall on a 4-byte boundary, this field shall be present and of the correct length to ensure that $4 + \text{VendorIDLen} + \text{OpaqueElementDataLen} + \text{AlignPaddingSize}$ is a multiple of 4. The value of this field shall be all zeros, and the size of this field shall be 0, 1, 2, or 3.

1043 15 Signature generation

1044 The `SPDMsign` function used in various part of this specification defines the signature generation algorithm while accounting for the differences in the various supported cryptographic signing algorithms in the `ALGORITHMS` message.

1045 The signature generation function takes this form:

```
signature = SPDMsign(PrivKey, data_to_be_signed, context);
```

1046 The `SPDMsign` function shall take these input parameters:

- `PrivKey` : a secret key
- `data_to_be_signed` : a bit stream of the data that will be signed
- `context` : a string

1047 The function shall output a signature using `PrivKey` and a selected cryptographic signing algorithm.

1048 The signing function shall follow these steps to create `spdm_prefix` and `spdm_context` (See [Text or string encoding](#) for encoding rules):

1. Create `spdm_prefix`. The `spdm_prefix` shall be the repetition, four times, of the concatenation of "dmtf-spdm-v", `SPDMversionString` and ".". This will form a 64-character string.
2. Create `spdm_context`. If the Requester is generating the signature, `spdm_context` shall be the concatenation of "requester-" and `context`. If the Responder is generating the signature, the `spdm_context` shall be the concatenation of "responder-" and `context`.

1049 Now follows an example, designated Example 1, of creating a `combined_spdm_prefix`.

1050 The version of this specification for this example is 1.4.3, the Responder is generating a signature, and the `context` is "my example context". Thus, the `spdm_prefix` is "dmtf-spdm-v1.4.*dmtf-spdm-v1.4.*dmtf-spdm-v1.4.*dmtf-spdm-v1.4.*". The `spdm_context` is "responder-my example context".

1051 Next, the `combined_spdm_prefix` is formed. The `combined_spdm_prefix` shall be the concatenation of four elements: `spdm_prefix`, a byte with a value of zero, `zero_pad`, and `spdm_context`. The size of `zero_pad` shall be the number of bytes needed to ensure that the length of `combined_spdm_prefix` is 100 bytes. The size of `zero_pad` can be zero. The value of `zero_pad` shall be zero.

1052 Continuing Example 1, [Table 135 — Combined SPDM prefix](#) shows the `combined_spdm_prefix` with offsets. Offsets increase from left to right and top to bottom. As shown, the length of `combined_spdm_prefix` is 100 bytes.

Furthermore, a number surrounded by double quotation marks indicates that the ASCII value of that number is used. See [Text or string encoding](#) for encoding rules. Table 94 concludes Example 1.

1053 **Table 135 — Combined SPDM prefix**

Offset	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0	d	m	t	f	-	s	p	d	m	-	v	"1"	.	"4"	.	*
0x10	d	m	t	f	-	s	p	d	m	-	v	"1"	.	"4"	.	*
0x20	d	m	t	f	-	s	p	d	m	-	v	"1"	.	"4"	.	*
0x30	d	m	t	f	-	s	p	d	m	-	v	"1"	.	"4"	.	*
0x40	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	r	e	s	p	o	n	d	e
0x50	r	-	m	y	space (0x20)	e	x	a	m	p	l	e	space (0x20)	c	o	n
0x60	t	e	x	t												

1054 The next step is to form the `message_hash`. The `message_hash` shall be the hash of `data_to_be_signed` using the selected hash function in either `BaseHashSel` or `ExtHashSel`. Many hash algorithms allow implementations to compute an intermediate hash, sometimes called a running hash. An intermediate hash allows for the updating of the hash as each byte of the ordered data of the message becomes known. Consequently, the ability to compute an intermediate hash allows for memory utilization optimizations where an SPDM endpoint can discard bytes of the message that are already covered by the intermediate hash while waiting for more bytes of the message to be received.

1055 If the Responder is generating the signature, the selected cryptographic signing algorithm is indicated in either `BaseAsymSel` or `ExtAsymSel` (but not both) in the `ALGORITHMS` message. If the Requester is generating the signature, the selected cryptographic signing algorithm is indicated in `ReqBaseAsymAlg` of `RespAlgStruct` in the `ALGORITHMS` message.

1056 Because each cryptographic signing algorithm is vastly different, these clauses define the binding of `SPDMsign` to those algorithms.

1057 15.1 Signing algorithms in extensions

1058 If an algorithm is selected in either the `ExtAsymSel` or `AlgExternal` of `ReqBaseAsymAlg` of `RespAlgStruct` in the `ALGORITHMS` response, its binding is outside the scope of this specification.

1059 15.2 RSA and ECDSA signing algorithms

1060 All RSA and ECDSA specifications do not define a specific hash function. Thus, the hash function to use shall be the hash function selected by the Responder in `BaseHashSel` or `ExtHashSel` .

1061 The private key, defined by the specification for these algorithms, shall be `PrivKey` .

1062 In the specification for these algorithms, the letter `M` denotes the message to be signed. `M` shall be the concatenation of `combined_spdm_prefix` and `message_hash` .

1063 RSA and ECDSA algorithms are described in [Signature algorithm references](#).

1064 The [FIPS PUB 186-5](#) supports deterministic ECDSA as a variant of ECDSA. [RFC 6979](#) describes this deterministic digital signature generation procedure. This variant does not impact the signature verification process. How an implementation chooses to support ECDSA or deterministic ECDSA is outside the scope of this specification.

1065 15.3 EdDSA signing algorithms

1066 These algorithms are described in [RFC 8032](#).

1067 The private key, defined by RFC 8032, shall be `PrivKey` .

1068 In the specification for these algorithms, the letter `M` denotes the message to be signed.

1069 15.3.1 Ed25519 sign

1070 This specification only defines Ed25519 usage and not its variants.

1071 `M` shall be the concatenation of `combined_spdm_prefix` and `message_hash` .

1072 15.3.2 Ed448 sign

1073 This specification only defines Ed448 usage and not its variants.

1074 `M` shall be the concatenation of `combined_spdm_prefix` and `message_hash` .

1075 Ed448 defines a context string, `C` . `C` shall be the `spdm_context` .

1076 15.4 SM2 signing algorithm

1077 This algorithm is described in [GB/T 32918.2-2016](#). GB/T 32918.2-2016 also defines the variable `M` and `IDA`.

- 1078 The private key defined by GB/T 32918.2-2016 shall be `PrivKey` .
- 1079 In the specification for SM2, the letter `M` denotes the message to be signed. `M` shall be the concatenation of `combined_spdm_prefix` and `message_hash` .
- 1080 The SM2 specification does not define a specific hash function. Thus, the hash function to use shall be the hash function selected by the Responder in `BaseHashSel` or `ExtHashSel` .
- 1081 Lastly, SM2 expects a distinguishing identifier, which identifies the signer and is indicated by the variable `IDA`. If this algorithm is selected, the ID shall be an empty string of size 0.

1082 15.5 Signature algorithm references

- 1083 These clauses provide basic information about each asymmetric algorithms SPDM supports, as [Table 136 — SPDM Asymmetric Signature Reference Information](#) shows. SPDM endpoints shall use the references in the **References** column for signature-related operations and the key size as indicated in the **Key Size** columns for the respective algorithm. The byte order for a signature when placing it into an SPDM signature field shall be [signature byte order](#).

1084 **Table 136 — SPDM Asymmetric Signature Reference Information**

Algorithm Name	Key Size (bits)	References
TPM_ALG_RSASSA_2048	2048	Section 8.2 of IETF RFC 8017
TPM_ALG_RSASSA_3072	3072	Section 8.2 of IETF RFC 8017
TPM_ALG_RSASSA_4096	4096	Section 8.2 of IETF RFC 8017
TPM_ALG_RSAPSS_2048	2048	Section 8.1 of IETF RFC 8017
TPM_ALG_RSAPSS_3072	3072	Section 8.1 of IETF RFC 8017
TPM_ALG_RSAPSS_4096	4096	Section 8.1 of IETF RFC 8017
TPM_ALG_ECDSA_ECC_NIST_P256	256	Section 6 of FIPS PUB 186-5 using <code>TPM_ECC_NIST_P256</code> curve parameters as TCG Algorithm Registry defines.
TPM_ALG_ECDSA_ECC_NIST_P384	384	Section 6 of FIPS PUB 186-5 using <code>TPM_ECC_NIST_P384</code> curve parameters as TCG Algorithm Registry defines.
TPM_ALG_ECDSA_ECC_NIST_P521	521	Section 6 of FIPS PUB 186-5 using <code>TPM_ECC_NIST_P521</code> curve parameters as TCG Algorithm Registry defines.
TPM_ALG_SM2_ECC_SM2_P256	256	Section 6 of GB/T 32918.2-2016 using <code>TPM_ECC_SM2_P256</code> curve parameters as TCG Algorithm Registry defines.
EdDSA ed25519	256	IETF RFC 8032

Algorithm Name	Key Size (bits)	References
EdDSA ed448	456	IETF RFC 8032

1085 16 Signature verification

1086 The `SPDMsignatureVerify` function, used in various part of this specification, defines the signature verification algorithm while accounting for the differences in the various supported cryptographic signing algorithms in the `ALGORITHMS` message.

1087 The signature verification function takes this form:

```
SPDMsignatureVerify(PubKey, signature, unverified_data, context);
```

1088 The `SPDMsignatureVerify` function shall take these input parameters:

- `PubKey` : the public key
- `signature` : a digital signature
- `unverified_data` : a bit stream of data that needs to be verified
- `context` : a string

1089 The function shall verify the `unverified_data` using `signature`, `PubKey`, and a selected cryptographic signing algorithm. `SPDMsignatureVerify` shall return success if the signature verifies correctly and failure otherwise. Each cryptographic signing algorithm states the verification steps or criteria for successful verification.

1090 The verifier of the signature shall create `spdm_prefix`, `spdm_context`, and `combined_spdm_context` as described in [Signature generation](#).

1091 The next step is to form the `unverified_message_hash`. The `unverified_message_hash` shall be the hash of `unverified_data` using the selected hash function in either `BaseHashSel` or `ExtHashSel`.

1092 If the Responder generated the signature, the selected cryptographic signature verification algorithm is indicated in either `BaseAsymSel` or `ExtAsymSel` (but not both) in the `ALGORITHMS` message. If the Requester generated the signature, the selected cryptographic signature verification algorithm is indicated in `ReqBaseAsymAlg` of `RespAlgStruct` in the `ALGORITHMS` message.

1093 Because each cryptographic signature verification algorithm is vastly different, these clauses define the binding of `SPDMsignatureVerify` to those algorithms.

1094 16.1 Signature verification algorithms in extensions

1095 If an algorithm is selected in either the `ExtAsymSel` or `AlgExternal` of `ReqBaseAsymAlg` of `RespAlgStruct` in the `ALGORITHMS` response, its binding is outside the scope of this specification.

1096 16.2 RSA and ECDSA signature verification algorithms

- 1097 All RSA and ECDSA specifications do not define a specific hash function. Thus, the hash function to use shall be the hash function selected by the Responder in `BaseHashSel` or `ExtHashSel` .
- 1098 The public key, defined in the specification for these algorithms, shall be `PubKey` .
- 1099 In the specification for these algorithms, the letter `M` denotes the message that is signed. `M` shall be concatenation of the `combined_spdm_prefix` and `unverified_message_hash` .
- 1100 For RSA algorithms, `SPDMsignatureVerify` shall return success when the output of the signature verification operation, as defined in the RSA specification, is "valid signature". Otherwise, `SPDMsignatureVerify` shall return a failure.
- 1101 For ECDSA algorithms, `SPDMsignatureVerify` shall return success when the output of "ECDSA Signature Verification Algorithm" as defined in [FIPS PUB 186-5](#) is "accept" . Otherwise, `SPDMsignatureVerify` shall return failure.
- 1102 RSA and ECDSA algorithms are described in [Signature algorithm references](#).

1103 16.3 EdDSA signature verification algorithms

- 1104 [RFC 8032](#) describes these algorithms. RFC 8032, also, defines the `M` , `PH` , and `C` variables.
- 1105 The public key, also defined in RFC 8032, shall be `PubKey` .
- 1106 In the specification for these algorithms, the letter `M` denotes the message to be signed.

1107 16.3.1 Ed25519 verify

- 1108 `M` shall be the concatenation of `combined_spdm_prefix` and `unverified_message_hash` .
- 1109 `SPDMsignatureVerify` shall return success when step 1 does not result in an invalid signature and when the constraints of the group equation in step 3 are met as described in [RFC 8032](#) section 5.1.7. Otherwise, `SPDMsignatureVerify` shall return failure.

1110 16.3.2 Ed448 verify

- 1111 `M` shall be the concatenation of `combined_spdm_prefix` and `unverified_message_hash` .
- 1112 Ed448 defines a context string, `C` . `C` shall be the `spdm_context` .
- 1113 `SPDMsignatureVerify` shall return success when step 1 does not result in an invalid signature and when the

constraints of the group equation in step 3 are met as described in [RFC 8032](#) section 5.2.7. Otherwise, `SPDMsignatureVerify` shall return failure.

1114 **16.4 SM2 signature verification algorithm**

1115 This algorithm is described in [GB/T 32918.2-2016](#), which also defines the variable `M` and `IDA`.

1116 The public key, also defined in GB/T 32918.2-2016, shall be `PubKey` .

1117 In the specification for SM2, the variable `M'` is used to denote the message that is signed. `M'` shall be the concatenation of `combined_spdm_prefix` and `unverified_message_hash` .

1118 The SM2 specification does not define a specific hash function. Thus, the hash function to use shall be the hash function selected by the Responder in `BaseHashSel` or `ExtHashSel` .

1119 Lastly, SM2 expects a distinguishing identifier, which identifies the signer, and is indicated by the variable `IDA`. See [SM2 signing algorithm](#) to create the value for `IDA`.

1120 `SPDMsignatureVerify` shall return success when the Digital signature verification algorithm, as described in GB/T 32918.2-2016, outputs an "accept". Otherwise, `SPDMsignatureVerify` shall return failure.

1121 17 General ordering rules

- 1122 These general ordering rules apply to SPDM messages that form a transcript that eventually gets signed.
- 1123 When requests are received out of order, the Responder can either silently discard all requests (with the exception of `GET_VERSION`) or return an `ERROR` message of `ErrorCode=RequestResynch` until the Requester issues a `GET_VERSION` . Out-of-order requests shall nullify the transcript.
- 1124 A Requester can retry messages. The retries shall be identical to the first message, excluding transport variances. If the Responder sees two or more non-identical `NEGOTIATE_ALGORITHMS` , the Responder shall either return an `ERROR` message of `ErrorCode=UnexpectedRequest` or silently discard non-identical messages. Because a retried message is identical to the first, a retried message shall not be used in transcript hash calculations.
- 1125 If a Requester wants to retrieve a `CAPABILITIES` response with the Supported Algorithms included, the Requester should first issue `GET_CAPABILITIES` with Bit 1 in `Param1` set to 1. If the Responder does not support the Supported Algorithms block in its `CAPABILITIES` response, it responds with an `ERROR` response. At this point, the Requester can issue a second `GET_CAPABILITIES` with Bit 1 in `Param1` cleared to 0. In this case, the second request is not considered a retry, and both requests and their corresponding responses are used in transcript hash calculations. After a successful `CAPABILITIES` response, if the Responder sees two or more non-identical `GET_CAPABILITIES` requests, the Responder shall either return an `ERROR` message of `ErrorCode=UnexpectedRequest` or silently discard non-identical messages.
- 1126 For `CHALLENGE` and Session-Secrets-Exchange, the Responder should ensure it can distinguish between the respective retry and the respective original message. Failure to distinguish correctly might lead to an authentication failure, session handshake failures, and other failures. The response to a retried request should be identical to the original response.

1127 18 DMTF event types

1128 The DMTF-defined event types are sent using the [Event mechanism](#).

1129 The [DMTF event types table](#) shows the supported DMTF event types for the DMTF event group. The values in the **Event Type ID** column shall be the same values for `EventTypeId` field in [Event data table](#) for the DMTF event group for the corresponding event in the **Event Name** column. The version (`EventGroupVer`) of the DMTF Event Group shall be `1`.

1130 **Table 137 — DMTF event types table**

Event Type ID	Event Name	Requirement	Description
0	Reserved	Reserved	Reserved.
1	EventLost	Mandatory	Events were lost.
2	MeasurementChanged	Optional	One or more measurements changed.
3	MeasurementPreUpdate	Optional	A pending update will change one or more measurements. However, the update has not yet taken effect.
4	CertificateChanged	Optional	Information in one or more certificate slots has changed. This could be the certificate or the associated key.
All others	Reserved	Reserved	Reserved.

1131 18.1 Event type details

1132 Each DMTF event type has its own event-specific information, referred to as `EventDetail`, to describe the event. These clauses describe the format for each DMTF event type. The event types are listed in the [DMTF event types table](#).

1133 18.1.1 Event Lost

1134 This event (`EventTypeId=EventLost`) shall notify the Event Recipient that one or more events were lost. The reasons for event loss are varied and numerous, but one example is loss due to insufficient resources. This event should be retried until the Event Recipient acknowledges it. Retrying this event means that this event was not acknowledged previously.

1135 The [Event lost format table](#) describes the format for `EventDetail`.

1136 **Table 138 — Event lost format**

Offset	Field	Size (bytes)	Description
0	LastAckedEventInstID	4	Shall be the last event instance ID acknowledged by the Event Recipient.
4	LastLostEventInstID	4	Shall be the last lost event instance ID.

1137 The range of lost events shall be the range from (LastAckedEventInstID + 1) to LastLostEventInstID inclusive.

1138 If the Event Notifier cannot or can no longer track the information in [Event lost format](#), then LastAckedEventInstID and LastLostEventInstID shall both be 0xFFFF_FFFF.

1139 When resending an "event lost" event, the Event Notifier can update fields in [Event lost format](#) if new events are lost since the last time the "event lost" event was sent.

1140 18.1.2 Measurement changed event

1141 The measurement changed event (EventTypeId=MeasurementChanged) shall notify the Event Recipient when one or more [measurement blocks](#) have changed. The MeasurementChanged event is applicable only when TerminationPolicy = 1 in KEY_EXCHANGE or PSK_EXCHANGE . If TerminationPolicy = 0 , the session will be terminated upon measurement update. The EventDetail format for this event type shall be as [Measurement changed event details format](#) defines.

1142 [Table 139 — Measurement changed event details format](#) describes the format for EventDetail for the MeasurementChanged event.

1143 **Table 139 — Measurement changed event details format**

Offset	Field	Size (bytes)	Description
0	ChangedMeasurements	32	This field is a bit mask where each bit indicates changes to its corresponding measurement index. Specifically, the bit at bit offset X shall be set to indicate a change to the Measurement block at measurement index X. At least one bit in this field shall be set. Bits 0 and 255 shall be reserved.

1144 The Event Recipient can issue GET_MEASUREMENTS to obtain further details on the change.

1145 18.1.3 Measurement pre-update event

1146 The measurement pre-update event (EventTypeId=MeasurementPreUpdate) notifies the Event Recipient when one or more [Measurement blocks](#) will change due to a pending update. The EventDetail format for this event type shall be as [Measurement pre-update event details format](#) defines.

1147 [Table 140 — Measurement pre-update event details format](#) describes the format for `EventDetail` for the `MeasurementPreUpdate` event.

1148 **Table 140 — Measurement pre-update event details format**

Offset	Field	Size (bytes)	Description
0	PreUpdateMeasurementChanges	32	This field is a bit mask where each bit indicates pending changes to the corresponding measurement index in an update scenario such as a firmware update or pending configuration change. Specifically, the bit at bit offset X shall be set to indicate a potential change to the Measurement block at measurement index X as a result of an update. At least one bit in this field shall be set. Bits 0 and 255 shall be reserved.

1149 Upon receiving the `MeasurementPreUpdate` event, the Event Recipient may send `GET_MEASUREMENTS` with the `NewMeasurementRequested` option (see [Table 50 — GET_MEASUREMENTS request attributes](#)) to acquire and evaluate the Event Notifier's pending new measurements. If the Event Recipient deems the Event Notifier's new measurements unacceptable, the Event Recipient may terminate the session.

1150 The pre-update notification mechanism does not allow the Event Recipient to stop the Event Notifier from applying the update. However, an Event Notifier that has sent `MeasurementPreUpdate` to the Event Recipient should not apply the update until one of the following events happens:

- Arrival of `EVENT_ACK` from the Event Recipient
- Arrival of `END_SESSION` from the Event Recipient
- Event Recipient timeout (per [Timing requirements](#))

1151 18.1.4 Certificate changed event

1152 The certificate changed event (`EventTypeId=CertificateChanged`) shall notify the Event Recipient when data associated with one or more fields in the `DIGESTS` response have changed. The `EventDetail` format for this event type shall be the [Certificate changed event details format](#).

1153 [Table 141 — Certificate changed event details format table](#) describes the format for `EventDetail` for the `CertificateChanged` event.

1154 **Table 141 — Certificate changed event details format**

Offset	Field	Size (bytes)	Description
0	CertificateChanged	1	This field is a bit mask where each bit indicates certificate related changes to the corresponding certificate slot. Specifically, the bit at bit offset X shall be set to indicate a change to data associated with one or more fields in <code>DIGESTS</code> for certificate slot X. At least one bit in this field shall be set.

1155 The Event Recipient can issue `GET_DIGESTS` or `GET_CERTIFICATE` to obtain further details on the change.

1156 **19 ANNEX A (informative) TLS 1.3**

1157 This specification heavily models TLS 1.3. TLS 1.3, and consequently this specification, assumes the transport layers provide the following 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.

1158 While not all transports are created equal, if a transport cannot meet these capabilities, adoption of SPDM is still possible. In these transports, this specification recommends [The Datagram Transport Layer Security \(DTLS\) Protocol Version 1.3](#).

1159 20 ANNEX B (informative) Device certificate example

1160 [Device certificate example](#) shows an example device certificate:

1161 Device 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::ACME:WIDGET:0123456789
Signature Algorithm: ecdsa-with-SHA256
Signature Value:
    30:45:02:20:1e:5a:a6:ed:5c:b6:2b:f5:9e:22:28:9c:ef:c7:
    aa:db:1c:87:83:48:c1:50:cb:25:04:ab:c9:6e:7c:f5:6b:01:
    02:21:00:da:48:d4:49:a5:65:5c:2c:83:fc:05:00:66:48:98:
    f8:f0:cb:63:b7:2e:87:db:c8:63:58:6c:21:91:7a:68:95
-----BEGIN CERTIFICATE-----
MIIC4jCCAoigAwIBAgIBCDABggqhkJOPQDAjBcMQswCQYDVQQGEwJDQTElMAkG
A1UECAwCTkxDTALBgNVBACMBGNpdHkxDTALBgNVBAoMBEFDTUUxFTATBgNVBASM
DEFDTUUGRGV2aWNlczELMAkGA1UEAwwCQ0EwIBcNNzAwMTAxMDAwMDAwWhgPOTk5
OTEyMzEyMzU5NTlaMH0xCzAJBgNVBAYTAlVTMQswCQYDVQQIDAJ0QzEiMCAGA1UE
CgwZQUNNRSBXawRnZXQgTWfudWZhY3R1cmZzEnMCUGA1UECwweQUNNRSBXawRn
ZXQgTWfudWZhY3R1cmZzYyBVbml0MRQwEgYDVQQDDAt3MDEyMzU5NTU5OTk5OTk5
DQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALpnR3J42iiB2YGb24gD4RCKkbhI
7WtwP0yiaKk7X3j8rkrRHGN2VKhAMSZ//z7gv5VcSrRvEVbKyBFTI+Edonql8CLY
svtD2t29UmvmpT8P02C4dNtWCNnuoDBKAYEe7mCt5AB6bmsyHCh+n0jDVNTj/R/R
RiCe74CIAF8l289DRsYfUbl/mCOE0IhHXVG0EWJvDyh3pyA083QngnCnllsbuxDn
lWLiN0u6IE48yRiyzUtYcKuivPYv7S9Ikr5azFxeq0qdY0j4hX3ADS9qCHTRL+he
Pbc1ph3SpgS205BDZjXhdBColztJBVFhB8YIARzcqF+eMJeoGGz5sSxW6GcCAwEA
AaNNMEswCQYDVROTBAlwADALBgNVHQ8EBAMCBeAwMQYDVROBRCowKKAmBgorBgEE
AYMcghIBoBgMFkFDTUU6V0LER0VU0jAXMjM0NTY3ODkwCgYIKoZIzj0EAwIDSAAw
RQIgHlqm7Vy2K/WeIiic78eq2xyHg0jBUMslBKvJbnz1awECIQDaSNRjpWVcLIP8
BQBmSJj48Mtjty6H28hjWGwhkXpolQ==
-----END CERTIFICATE-----

```

21 ANNEX C (informative) OID reference

Table 142 — Object identifiers (OIDs) lists all object identifiers (OIDs) that this specification defines:

Table 142 — Object identifiers (OIDs)

OID	Identifier	Definition	Use
{ 1 3 6 1 4 1 412 }	id-DMTF	DMTF OID	Enterprise ID for DMTF
{ id-DMTF 274 }	id-DMTF-spdm	SPDM OID	Base OID for all SPDM OIDs
{ id-DMTF-spdm 1 }	id-DMTF-device-info	SPDM certificate requirements and recommendations	Certificate device information.
{ id-DMTF-spdm 2 }	id-DMTF-hardware-identity	Identity provisioning	Hardware certificate identifier.
{ id-DMTF-spdm 3 }	id-DMTF-eku-responder-auth	Extended Key Usage authentication OIDs	Certificate Extended Key Usage - SPDM Responder Authentication.
{ id-DMTF-spdm 4 }	id-DMTF-eku-requester-auth	Extended Key Usage authentication OIDs	Certificate Extended Key Usage - SPDM Requester Authentication.
{ id-DMTF-spdm 5 }	id-DMTF-mutable-certificate	Identity provisioning	Mutable certificate identifier.
{ id-DMTF-spdm 6 }	id-DMTF-SPDM-extension	SPDM Non-Critical Certificate OID	To contain other OIDs in a certificate extension.

22 ANNEX D (informative) variable name reference

Throughout this document, various sizes and offsets are referred to by a variable. [Table 143 — Variables](#) lists variables used in this document, the definition of the variable, and the location in this document that shows how the variable is set.

Table 143 — Variables

Symbol	Definition	Set location
A	Number of Requester-supported extended asymmetric key signature algorithms.	Table 15 — NEGOTIATE_ALGORITHMS request message format
A'	Number of extended asymmetric key signature algorithms selected by the Requester.	Table 21 — Successful ALGORITHMS response message format
D	The size of D (and C for ECDHE) that is derived from the selected DHE group.	See the <code>KEY_EXCHANGE</code> request message format in Table 69 — KEY_EXCHANGE request message format .
E	Number of Requester-supported extended hashing algorithms.	Table 15 — NEGOTIATE_ALGORITHMS request message format
E'	The number of Requester-supported extended hashing algorithms selected by the Responder.	Table 21 — Successful ALGORITHMS response message format
L _x where x is a number	A generic variable used to indicate the sizes of a field. The x is a number starting with zero. An example of L _x is L ₀ , L ₁ and so forth. The scope of this variable is always local to the table that uses it. For example, L ₀ often appears in more than one table but there is no relationship between an L ₀ in one table and an L ₀ in another table.	Numerous tables
H	The output size, in bytes, of the hash algorithm agreed upon in <code>NEGOTIATE_ALGORITHMS</code> .	Table 21 — Successful ALGORITHMS response message format
HEM	Hash-extend measurement.	Hash-extend measurements clause .
MS	The length of the cryptographic hash or raw bit stream, as indicated in <code>DMTFSpecMeasurementValueType[7]</code> .	Table 45 — DMTF measurement specification format

Symbol	Definition	Set location
MSHLength	The length of the MeasurementSummaryHash field in the <code>CHALLENGE_AUTH</code> , <code>KEY_EXCHANGE_RSP</code> , and <code>PSK_EXCHANGE_RSP</code> messages.	Table 45 — Successful CHALLENGE_AUTH response message format
NL	The length of the Nonce field in the <code>GET_MEASUREMENTS</code> request and the <code>MEASUREMENTS</code> response.	GET_MEASUREMENTS request attributes
n	Number of version entries in the <code>VERSION</code> response message.	Table 9 — Successful VERSION response message format
Q	Length of the ResponderContext.	Table 75 — PSK_EXCHANGE_RSP response message format
P	Length of the <code>PSKHint</code> .	Table 74 — PSK_EXCHANGE request message format
R	Length of the <code>RequesterContext</code> .	Table 74 — PSK_EXCHANGE request message format
SigLen	The size of the asymmetric-signing algorithm output, in bytes, that the Responder selected in the last <code>ALGORITHMS</code> response message.	Table 21 — Successful ALGORITHMS response message format

1168 23 ANNEX E (informative) change log

1169 23.1 Version 1.0.0 (2019-10-16)

- Initial Release

1170 23.2 Version 1.1.0 (2020-07-15)

- Minor typographical fixes
- USB Authentication Specification 1.0 link updated
- Tables are no longer numbered. They are now named.
- Fix internal document links in SPDM response codes table.
- Added sentence to paragraph 97 to clarify on the potential to skip messages after a reset.
- Removed text at paragraph 181.
- `Subject Alternative Name otherName` field in [Optional fields](#) references DMTF OID section.
- `DMTFOtherName` definition changed to properly meet ASN.1 syntax.
- Text in figures is now searchable.
- Corrected example of a leaf certificate in Annex A.
- Minor edits to figures for clarity.
- Clarified that transcript hash could include hash of the raw public key if a certificate is not used.
- New:
 - Added [Session](#) support.
 - Added SPDM request and response messages to support initiating, maintaining and terminating a secure session.
 - Added [Key schedule](#) for session secrets derivation.
 - Added [Application Data](#) to provide overview of how data is encrypted and authenticated in a session.
 - Introduce new terms and definitions.
 - Added Measurement Manifest to `DMTFSpecMeasurementValueType`.
 - Added [mutual authentication](#).
 - Added [Encapsulated request flow](#) to support master-slave types of transports.

1171 23.3 Version 1.2.0 (2021-11-01)

- Clarified SPDM version selection after receiving `VERSION` Response with error handling for certain scenarios.
 - Fix improper reference in `DMTFSpecMeasurementValue` field in "Measurement field format when MeasurementSpecification field is Bit 0 = DMTF" table.
-

- Certificate digests in `DIGESTS` calculation clarified.
- Format of certificate in `CertChain` parameter of `CERTIFICATE` message clarified.
- Validity period of X.509 v3 certificate clarified in [Required Fields](#)
- Remove `InvalidSession` error code.
- Clarified transport responsibilities in `PSK_EXCHANGE` and `PSK_EXCHANGE_RSP`.
- Clarified the usage of `MutAuthRequested` field in `KEY_EXCHANGE_RSP`.
- Added recommendation of PSK usage when an SPDM endpoint can be a Requester and Responder.
- Added recommendation for usage of `RequesterContext` in PSK scenarios.
- Clarified capabilities for Requester and Responder in `GET_CAPABILITIES` and `CAPABILITIES` messages.
- Clarified [timing requirements](#) for [encapsulated requests](#).
- Clarified out of order and retries
- Clarified error handling actions when unexpected requests occur during various mutual authentication flows.
- Refer to slot number fields as `SlotID` and normalize `SlotID` fields to 4 bits where possible.
- Changed `PSK_FINISH` and `FINISH` changes in [Table 6 — SPDM request and response messages validity](#).
- Clarified `HANDSHAKE_IN_THE_CLEAR_CAP` usage in `PSK_EXCHANGE`.
- Change `SPDMVersion` field in every request and response message, except `GET_VERSION` / `VERSION` messages, to point to a central location in this specification where it explains the appropriate value to populate for this field.
- Clarified use case for `Token` field in `ResponseNotReady`.
- Clarified the format of the certificate chain used in the Transcript hash calculation in [Transcript hash calculation rules](#).
- Renamed `Measurement` field format when `MeasurementSpecification` field is Bit 0 = DMTF table to [Table 45 — DMTF measurement specification format](#).
- Clarified the `ENCAP_CAP` field in the capabilities of the Requester and Responder.
- Renamed Mutual Authentication in `KEY_EXCHANGE` to Session-based mutual authentication.
- `ERROR` responses are no longer required in most error scenarios.
- Clarify the definition of backward-compatible changes in [Version encoding](#).
- Enhanced requirements for when a firmware update occurred on a Responder in [GET_VERSION request and VERSION response messages](#).
- Clarified error code `ResponseNotReady` for M1/M2 and L1/L2 computation.
- Clarified byte order for ASN.1 encoded data, hashes and digests.
- Requester should not use `PSK_EXCHANGE` if `CHALLENGE_AUTH` and/or `MEASUREMENTS` with signature was received from Responder.
- Required `GET_VERSION`, `VERSION`, `GET_CAPABILITIES`, `CAPABILITIES`, `NEGOTIATE_ALGORITHMS`, and `ALGORITHMS` in transcript even if negotiated state is supported.
- Enhanced signature generation and verification with a prefix to mitigate signature misuse attacks.
- Clarified behavior of `END_SESSION` with respect to Negotiated State when there are multiple active sessions.
- Added new defined term `Reset` to mean device reset. Updated use of the word reset for M1/M2, L1/L2.
- Clarified that a Measurement Manifest should support both hash and raw bit stream formats.

- Clarified Measurement Summary Hash construction rules.
- Clarified minimum timing for [HEARTBEAT request and HEARTBEAT_ACK response messages](#) to be sufficiently greater than `T1`. Removed command-specific guidance on retry timing.
- Table codification changed to be consistent with DMTF template.
- New:
 - Added support for `AliasCert` s.
 - Compliant Requesters must support a Responder that uses either `DeviceCert` s or `AliasCert` s.
 - Added [Certain error handling in encapsulated flows](#)
 - Added Slot 0 certificate-provisioning methodology.
 - Added [Allowance for encapsulated requests](#).
 - Allowed `GET_CERTIFICATE` followed by `CHALLENGE` flow after a reset in `M1` and `M2` message transcript.
 - Added new features for `GET_MEASUREMENTS` and `MEASUREMENTS` :
 - More measurement value types.
 - Allow Requester to request hash or raw bit stream for measurement from the Responder.
 - Added [Advice](#).
 - Added structured representation of device mode [Device mode field of a measurement block](#).
 - Added [Text or string encoding](#).
 - Signature Clarification:
 - Added [Signature generation](#) and [Signature verification](#) for clarity and interoperability.
 - Change `Sign` and `Verify` abstract function to `SPDMsign` and `SPDMsignatureVerify` respectively.
 - Added [General ordering rules](#) and references to it, to describe additional requirements for the various transcript and message transcripts.
 - Added additional clause for checking `FINISH` . `Param2` if handshake is in the clear.
 - Added OIDs to represent:
 - Hardware certificate identifier ([Identity provisioning](#))
 - Certificate Extended Key Usage - SPDM Responder Authentication ([Extended Key Usage authentication OIDs](#))
 - Certificate Extended Key Usage - SPDM Requester Authentication ([Extended Key Usage authentication OIDs](#))
 - Mutable certificate identifier ([Identity provisioning](#))
 - Added [SM2](#) to Base Asymmetric Algorithms and Key Exchange Protocols.
 - Added [SM3](#) to Base Hash Algorithms and Measurement Hash Algorithms.
 - Added [SM4](#) to AEAD Algorithms.
 - Changed symbol "S" denoting signature size to "SigLen" throughout document.
 - Removed potentially confusing mention of "mutual authentication" in `PSK_EXCHANGE` section.
 - Add method to transfer large SPDM messages. See [Large SPDM message transfer mechanism](#).
 - Changed Measurement Summary Hash concatenation function inputs.
 - Clarified requirements for compliant certificate chains.
 - Tables and figures are now numbered. Though these numbers might change in future versions of

specification, the titles will remain the same.

- Allowed Requester to specify session termination policy when Responder completes firmware or configuration update.

1172 23.4 Version 1.3.0 (2023-04-05)

- Change attribution for this standard from the [Platform Management Communications Infrastructure \(PMCI\)](#) Working Group to the [Security Protocols and Data Models](#) Working Group.
- Fix minor typographical errors.
- Clarified `CSRdata` requirements.
- Correct indication that Identity Provisioning OIDs are in the certificate Extended Key Usage, and add SPDM Non-Critical Certificate Extension OID to [Table 43 — Optional fields](#).
- Added [Signature Algorithm References](#) clauses to clarify basic information about asymmetric algorithms.
- Clarified `Offset` and `Length` fields in `GET_CERTIFICATE` message.
- Clarified measurement specification related fields in `NEGOTIATE_ALGORITHMS`, `ALGORITHMS` and [Table 53 — Measurement block format](#).
- Added recommended `ErrorCode` for the case when the Responder detects overlapping `SET_CERTIFICATE` commands.
- Clarified `DataTransferSize` and `MaxSPDMmsgSize` in `GET_CAPABILITIES` and `CAPABILITIES` messages.
- Updated [General ordering rules](#) to include discussion of the `CAPABILITIES` response with the Support Algorithms block.
- Allow the sender to utilize the [Large SPDM message transfer mechanism](#) when the transmit buffer size of the sender is less than the `DataTransferSize` of the receiving SPDM endpoint.
- Clarified that `ENCRYPT_CAP` and `MAC_CAP` apply to all phases of a secure session.
- Clarified the relationship between `MAC_CAP` and `ResponderVerifyData` or `RequesterVerifyData` in Session-Secret-Exchange and Session-Secret-Finish messages.
- Provide more description for `HANDSHAKE_IN_THE_CLEAR_CAP` in `GET_CAPABILITIES` and `CAPABILITIES` messages.
- Added `VERSION` to the chunking forbidden list.
- Added definition of [opaque data](#).
- Make the layout of tables 62 and 63 consistent with other tables.
- Clarified DER encoding for 'RequesterInfo'
- Added more guidance to `RawBitStreamRequested` in `GET_MEASUREMENTS` request.
- Changed ANNEX B from "normative" to "informative".
- Corrected Requester to Responder in Table 71 Successful KEY_EXCHANGE_RSP response message format.
- Correct values in Field and Size columns of Table 61
- Changed the message validity of `VENDOR_DEFINED_REQUEST` and `VENDOR_DEFINED_RESPONSE` to "Vendor-defined".
- Clarified measurement method for various timing parameters in [Timing specification table](#).

- Corrected the signing algorithm in the FINISH request's Signature field.
- Correct [Figure 1 — SPDM certificate chain models](#) to show AliasCert model.
- Clarify how retried messages affect transcript hash in [General ordering rules](#).
- Update [Table 7 — Timing specification for SPDM messages](#) to clarify that Responders can exceed `ST1` and `CT` using `ErrorCode=ResponseNotReady`.
- Clarified rules around when the old key can be discarded during `KEY_UPDATE`.
- Updated link and information for IETF DTLS 1.3.
- Clarified that `AlgCount` field in Algorithm request and response structures shall be a value of 2.
- Edit Figure 22 so that a Secure Session does not encompass Session-Secrets-Exchange.
- Clarified measurement signing capabilities in `SignatureRequested` field of `GET_MEASUREMENTS`.
- Clarified retries from the perspective Responder and Requester in [Timing requirements](#).
- Changed "or" to "and" in Large SPDM message transfer mechanism section.
- Clarified that `MeasurementHashAlgo` should be zero if `MeasurementSpecificationSel` is zero.
- Remove "in general" from normative text.
- Clarified that the use of `BaseAsymAlgo` in the `NEGOTIATE_ALGORITHMS` request is dependent on the capabilities of the Responder.
- Removed directive to save the public key of the leaf certificate retrieved through the `GET_CERTIFICATE` request.
- Added trusted environment to glossary.
- Clarified how the value of `MinDataTransferSize` is calculated.
- Added `LargeResponse` error to description of chunking certificates.
- Clarified that if endpoint does not support chunking then it must set `MaxSPDMmsgSize` equal to `DataTransferSize`.
- Clarified effects on out-of-order message on the transcript and other clarifications in [General ordering rules](#).
- Clarified the definition of Session-Secret-Exchange and removed the duplicate definition of it.
- Replaced wording of "internal buffer" in `GET_CERTIFICATE` with `DataTransferSize` and "transmit buffer".
- Specify the hashing algorithm for `MeasurementSummaryHash` in multiple tables.
- Added normative statement that `VERSION` entries should be unique.
- Clarified conditions for `LargeResponse` error.
- Clarified `CERTIFICATE` response when the `Length` field of `GET_CERTIFICATE` is zero.
- Clarified the assumption that version entries are not duplicated when calculating `MinDataTransferSize`.
- Introduced `Context` field in `CHALLENGE` and `GET_MEASUREMENTS` requests.
- Clarified restrictions on Bit 0 through 2 of the `MutAuthRequested` field of `KEY_EXCHANGE_RSP`.
- Separated nonce and non-repeating counter in `PSK_EXCHANGE` and `PSK_EXCHANGE_RSP`.
- Added definitions for *sequentially decreasing*, *sequentially increasing*, and *monotonically increasing*.
- Clarified updating keys in `KEY_UPDATE`.
- Added size of the transmit buffer as a condition for `CHUNK_SEND`.
- Clarified measurement support in the `MeasurementHashAlgo` field of the `ALGORITHMS` response.
- Clarified conditions under which `CERT_CAP` must be `0b`.

- Allowed `GET_DIGESTS` and `GET_CERTIFICATE` in session.
- Clarified that extended algorithms are external to this specification.
- Changed "should" to "shall" in the `LargeMessageSize` field of `CHUNK_SEND`.
- Clarified (A1, B4, C1) message flow is permitted.
- Required root certificate to always be included in `SET_CERTIFICATE`.
- Changed "cancel" to "invalidate state and data associated with" in `GET_VERSION` and `VERSION` response messages.
- Removed non-normative text from the `Length` field of `GET_CERTIFICATE`.
- Changed link to VCA from acronym to definition in the "transcript computation rules for M1/M2" table.
- Clarified Session-Secrets-Exchange in Optimized encapsulated request flow
- Clarified the `Request ID` for the first message in an optimized encapsulated request flow.
- Clarified the presence of the `SlotIDParam` field in `GET_MEASUREMENTS`.
- Removed informative statement that chunks are equal in size.
- Clarified that SPDM messages sent outside of a session do not contribute to in-session transcripts.
- Fixed typo in table 88.
- Deprecated the `CHAL_CAP` capability of the Requester.
- Clarified value of `HANDSHAKE_IN_THE_CLEAR_CAP` when using Pre-Shared Keys.
- Removed "after Reset" from M1/M2 ordering.
- Clarify that [Integers](#) are unsigned.
- Clarified requirements for chunking the `CERTIFICATE` response.
- Clarified relevant capabilities in `BaseAsymAlgo`, `BaseHashAlgo`.
- Clarified that Export Master Secret does not get updated with `KEY_UPDATE`.
- Removed the "full" modifier in front of `MeasurementRecord` in the [MEASUREMENTS response table](#).
- Fixed typos and removed redundant grammar in Table 50.
- Fixed OID value for id-DMTF-device-info to match earlier releases.
- Clarified definition of `DecryptError`.
- Clarified that endpoints must ensure proper ordering and existence of messages when calculating transcripts hashes.
- Fixed typo in table 90.
- Move `DMTFSpecMeasurementValueType[6:0]` to its own table to improve readability.
- Changed instances of `Concatenation()` to the defined `Concatenate()` operator.
- Clarified slots 1-7 certificate provisioning.
- Removed normative text that prohibited reuse of session IDs.
- Clarified that non-encapsulated requests are prohibited during the session handshake phase.
- Removed potentially confusing statements on Slot provisioning for `GET_CSR`.
- Removed normative error statement from the `BasicMutAuthReq` field of `CHALLENGE_AUTH`.
- Clarified exclusion of signature in `CHALLENGE_AUTH` and usage of concatenation in [Table 47](#)
- Clarified CSR signing.

- Removed encapsulation requirements from MUT_AUTH_CAP definition.
- Removed deprecation status from ENCAP_CAP.
- Clarified that a provisioned public key can be used to generate the [Transcript for KEY_EXCHANGE_RSP HMAC](#).
- Clarified use of `DataTransferSize` and `MaxSPDMmsgSize` in [GET_CAPABILITIES request and CAPABILITIES response messages](#).
- Fixed typo in table 52.
- Replaced links to ITU-T X.509 with RFC5280 and removed ITU-T X.509 from the Normative references section.
- Moved general text for transcript calculations from "Transcript and transcript hash calculation rules" to the "SPDM messaging protocol" section.
- Clarified that `KEY_EX_CAP` only applies to Requester's request message and Responder's response message.
- Clarified that if either Requester or Responder do not support Heartbeat then the value of `HeartbeatPeriod` would be 0.
- Renamed "VendorLen" to "VendorIDLen".
- Used different `Salt_0` value for PSK session in key schedule.
- Corrected `PK` to `PubKey` in [CHALLENGE_AUTH signature verification](#).
- Removed quotation mark of VCA in L1/L2 definition.
- Clarified which portions of a certificate chain in the [Alias certificate model](#) is immutable.
- Updated link and version to [ISO/IEC Directives, Part 2](#).
- Fixed size of MeasurementSummaryHash field to include 0 as a possible size value when the field is absent.
- Renamed the `HMAC-Hash` to `HKDF-Extract`.
- Moved message and field notation to [Notations](#).
- Clarified `VCA` for the case where capabilities and algorithms are provisioned alongside PSK.
- Clarified that `ProvisionedSlotMask` in the `CHALLENGE_AUTH` response is dependent on the negotiated algorithms.
- Clarified runtime measurement change detection.
- Removed "between devices" in the introduction of SPDM.
- Used different `Salt_0` value for PSK session in key schedule.
- Removed the restriction to set `Length` to be 0xFFFF in `GET_CERTIFICATE` if both endpoints support the large SPDM message transfer mechanism.
- Clarified `RequesterContext` in PSK_EXCHANGE.
- The Responder now always returns error ResponseTooLarge and no longer silently discards the request that caused this error.
- Clarified certificate chain validation in Figure 8.
- Clarified that a `GET_VERSION` request can also cancel a pending request at the responder in section about Requirement for Requesters.
- Restructure the [Identity provisioning](#) clause. Split the existing content into multiple clauses to help organization and incorporate the [Generic certificate model](#). Make the use of Device Certificate and Alias Certificate consistent rather than using the terms `DeviceCert` and `AliasCert` to refer to specific certificates.
- Add missing `ffdhe3072` in DHE secret computation section.

- Clarified that the Requester should not use `PSK_EXCHANGE` after receiving any Responder-signed response messages.
- Clarified that SPDM certificates are still compliant to the requirements of RFC 5280.
- Clarified field requirements for SPDM certificates and clarified that RFC 5280 defines the certificate format.
- Clarify allowed session phases for `GET_CSR`, `SET_CERTIFICATE`, `GET_DIGESTS`, and `GET_CERTIFICATE` in [Table 6 — SPDM request and response messages validity](#).
- Clarified `RESPOND_IF_READY` request validity.
- New:
 - Added [Signature byte order](#) and [Octet string byte order](#) clauses.
 - Add the [Manifest format for a measurement block](#) to define a measurement manifest header format that leverages the `SVH` format.
 - Added `SET_CERT_CAP`, `CSR_CAP` and `CERT_INSTALL_RESET_CAP` capabilities bits.
 - Add a section to discuss differences in cryptographic and non-cryptographic [Timing parameters](#).
 - Added option in `SET_CERTIFICATE` to delete existing certificate chain from slot.
 - Add a `SlotSizeRequested` request attribute to the [GET_CERTIFICATE request and CERTIFICATE response messages](#).
 - Added the IANA CBOR registry and VESA standards body to [Registry or standards body ID](#).
 - Added a tracking tag in [GET_CSR request and CSR response messages](#) for use after a reset.
 - Added missing `MaxSPDMmsgSize` to `GET_CAPABILITIES` request and `CAPABILITIES` response messages.
 - Add an `Overwrite` bit to the `GET_CSR` request.
 - Added requirements on population of Slot 0 in [Certificates and certificate chains](#).
 - Added [GET_ENDPOINT_INFO request and ENDPOINT_INFO response messages](#).
 - Added the `InvalidPolicy` error code.
 - Added [Supported algorithms block](#) to [Successful CAPABILITIES response message format](#).
 - Added column to table 132 that specifies whether values are secret or not.
 - Added new request `GET_MEASUREMENT_EXTENSION_LOG` and response `MEASUREMENT_EXTENSION_LOG`, measurement extension log formats, and examples.
 - Added new "hash-extended" measurement type.
 - Added [Multiple asymmetric key support](#).
 - Added [Generic certificate model](#).
 - Added [Notification overview](#) and [Event Mechanism](#)
 - Added [DMTF event types](#)
 - Added [Custom environments](#) clauses.
 - Added `NewMeasurementRequested` in `GET_MEASUREMENTS`.
 - Add missing `ffdhe3072` in DHE secret computation section.
 - Change [FIPS PUB 186-4](#) reference to [FIPS PUB 186-5](#).
 - Defined the data models for the first four bytes of `VendorDefinedReqPayload` and `VendorDefinedRespPayload` when standards body is DMTF.

1173 24 Bibliography

1174 DMTF DSP4014, [*DMTF Process for Working Bodies 2.6*](#).