



1

Document Identifier: DSP0274

2

Date: 2023-10-08

3

Version: 1.2.2

4

# Security Protocol and Data Model (SPDM) Specification

5

Supersedes: 1.2.1

6

Document Class: Normative

7

Document Status: Published

8

Document Language: en-US

Copyright Notice

Copyright © 2023 DMTF. All rights reserved.

- 9 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.
- 10 Implementation of certain elements of this standard or proposed standard may be subject to third-party patent rights, including provisional patent rights (herein “patent rights”). DMTF makes no representations to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose, or identify any or all such third-party patent right owners or claimants, nor for any incomplete or inaccurate identification or disclosure of such rights, owners, or claimants. DMTF shall have no liability to any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize, disclose, or identify any such third-party patent rights, or for such party’s reliance on the standard or incorporation thereof in its 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patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a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.
- 11 For information about patents held by third parties which have notified DMTF that, in their opinion, such patents may relate to or impact implementations of DMTF standards, visit <https://www.dmtf.org/about/policies/disclosures>.
- 12 This document’s normative language is English. Translation into other languages is permitted.

## CONTENTS

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 Sizes and lengths	12
2.2.5 SPDM data types	12
2.2.6 Version encoding	12
2.2.7 Notations	14
2.2.8 Text or string encoding	14
2.2.9 Deprecated material	15
3 Scope	16
4 Normative references	17
5 Terms and definitions	19
6 Symbols and abbreviated terms	23
7 SPDM message exchanges	24
7.1 Security capability discovery and negotiation	24
7.2 Identity authentication	24
7.2.1 Identity provisioning	25
7.2.1.1 Device certificate models	25
7.2.2 Raw public keys	27
7.2.3 Runtime authentication	28
7.3 Firmware and configuration measurement	28
7.4 Secure sessions	28
7.5 Mutual authentication overview	28
7.6 Custom environments	29
8 SPDM messaging protocol	30
8.1 SPDM connection model	32
8.2 SPDM bits-to-bytes mapping	32
8.3 Generic SPDM message format	33
8.3.1 SPDM version	34
8.4 SPDM request codes	34

8.5 SPDM response codes	36
8.6 SPDM request and response code issuance allowance	38
8.7 Concurrent SPDM message processing	39
8.8 Requirements for Requesters	40
8.9 Requirements for Responders	40
9 Timing requirements	41
9.1 Timing measurements	41
9.2 Timing specification table	41
10 SPDM messages	49
10.1 Capability discovery and negotiation	49
10.1.1 Negotiated state preamble	49
10.2 GET_VERSION request and VERSION response messages	50
10.3 GET_CAPABILITIES request and CAPABILITIES response messages	53
10.4 NEGOTIATE_ALGORITHMS request and ALGORITHMS response messages	62
10.4.1 Connection Behavior after VCA	76
10.5 Responder identity authentication	77
10.6 Requester identity authentication	79
10.6.1 Certificates and certificate chains	79
10.7 GET_DIGESTS request and DIGESTS response messages	80
10.8 GET_CERTIFICATE request and CERTIFICATE response messages	81
10.8.1 Mutual authentication requirements for GET_CERTIFICATE and CERTIFICATE messages	83
10.8.2 SPDM certificate requirements and recommendations	83
10.8.2.1 Extended Key Usage authentication OIDs	85
10.8.2.2 SPDM Non-Critical Certificate Extension OID	86
10.8.2.2.1 Hardware identity OID	86
10.8.2.2.2 Mutable certificate OID	87
10.9 CHALLENGE request and CHALLENGE_AUTH response messages	87
10.9.1 CHALLENGE_AUTH signature generation	90
10.9.2 CHALLENGE_AUTH signature verification	91
10.9.2.1 Request ordering and message transcript computation rules for M1 and M2	92
10.9.3 Basic mutual authentication	94
10.9.3.1 Mutual authentication message transcript	95
10.10 Firmware and other measurements	96
10.11 GET_MEASUREMENTS request and MEASUREMENTS response messages	97
10.11.1 Measurement block	101
10.11.1.1 DMTF specification for the Measurement field of a measurement block	102
10.11.1.2 Device mode field of a measurement block	104
10.11.2 MEASUREMENTS signature generation	105
10.11.3 MEASUREMENTS signature verification	107
10.12 ERROR response message	108
10.12.1 Standard body or vendor-defined header	116
10.13 RESPOND_IF_READY request message format	116

10.14	VENDOR_DEFINED_REQUEST request message	117
10.15	VENDOR_DEFINED_RESPONSE response message	118
10.15.1	VendorDefinedReqPayload and VendorDefinedRespPayload defined by DMTF specifications	119
10.16	KEY_EXCHANGE request and KEY_EXCHANGE_RSP response messages	120
10.16.1	Session-based mutual authentication	128
10.16.1.1	Specify Requester certificate for session-based mutual authentication	129
10.17	FINISH request and FINISH_RSP response messages	129
10.17.1	Transcript and transcript hash calculation rules	131
10.18	PSK_EXCHANGE request and PSK_EXCHANGE_RSP response messages	134
10.19	PSK_FINISH request and PSK_FINISH_RSP response messages	141
10.20	HEARTBEAT request and HEARTBEAT_ACK response messages	142
10.20.1	Heartbeat additional information	143
10.21	KEY_UPDATE request and KEY_UPDATE_ACK response messages	143
10.21.1	Session key update synchronization	144
10.21.2	KEY_UPDATE transport allowances	147
10.22	GET_ENCAPSULATED_REQUEST request and ENCAPSULATED_REQUEST response messages	150
10.22.1	Encapsulated request flow	150
10.22.2	Optimized encapsulated request flow	150
10.22.3	Triggering GET_ENCAPSULATED_REQUEST	153
10.22.4	Additional constraints	154
10.23	DELIVER_ENCAPSULATED_RESPONSE request and ENCAPSULATED_RESPONSE_ACK response messages	154
10.23.1	Additional information	156
10.23.2	Allowance for encapsulated requests	157
10.23.3	Certain error handling in encapsulated flows	157
10.23.3.1	Response not ready	157
10.23.3.2	Timeouts	158
10.24	END_SESSION request and END_SESSION_ACK response messages	158
10.25	Certificate provisioning	159
10.25.1	GET_CSR request and CSR response messages	160
10.25.2	SET_CERTIFICATE request and SET_CERTIFICATE_RSP response messages	161
10.26	Large SPDM message transfer mechanism	163
10.26.1	CHUNK_SEND request and CHUNK_SEND_ACK response message	164
10.26.2	CHUNK_GET request and CHUNK_RESPONSE response message	167
10.26.3	Additional chunk transfer requirements	169
11	Session	171
11.1	Session handshake phase	171
11.2	Application phase	172
11.3	Session termination phase	172
11.4	Simultaneous active sessions	172
11.5	Records and session ID	173

12	Key schedule	174
12.1	DHE secret computation	176
12.2	Transcript hash in key derivation	176
12.3	TH1 definition	177
12.4	TH2 definition	177
12.5	Key schedule major secrets	178
12.5.1	Request-direction handshake secret	178
12.5.2	Response-direction handshake secret	178
12.5.3	Request-direction data secret	178
12.5.4	Response-direction data secret	178
12.6	Encryption key and IV derivation	179
12.7	finished_key derivation	179
12.8	Deriving additional keys from the Export Master Secret	180
12.9	Major secrets update	180
13	Application data	181
13.1	Nonce derivation	181
14	General opaque data format	182
15	Signature generation	184
15.1	Signing algorithms in extensions	185
15.2	RSA and ECDSA signing algorithms	185
15.3	EdDSA signing algorithms	186
15.3.1	Ed25519 sign	186
15.3.2	Ed448 sign	186
15.4	SM2 signing algorithm	186
15.5	Signature algorithm references	186
16	Signature verification	188
16.1	Signature verification algorithms in extensions	188
16.2	RSA and ECDSA signature verification algorithms	189
16.3	EdDSA signature verification algorithms	189
16.3.1	Ed25519 verify	189
16.3.2	Ed448 verify	189
16.4	SM2 signature verification algorithm	190
17	General ordering rules	191
18	ANNEX A (informative) TLS 1.3	192
19	ANNEX B (informative) Device certificate example	193
20	ANNEX C (informative) OID reference	195
21	ANNEX D (informative) variable name reference	196
22	ANNEX E (informative) change log	197
22.1	Version 1.0.0 (2019-10-16)	197
22.2	Version 1.1.0 (2020-07-15)	197
22.3	Version 1.2.0 (2021-11-01)	197
22.4	Version 1.2.1 (2022-05-11)	200
22.5	Version 1.2.2 (2023-10-08)	201

23 Bibliography ..... 205

# 14 **1 Foreword**

---

15 The [Security Protocols and Data Models \(SPDM\) Working Group](#) of [DMTF](#) prepared the *Security Protocol and Data Model (SPDM) Specification* (DSP0274). DMTF is a not-for-profit association of industry members that promotes enterprise and systems management and interoperability. For information about DMTF, see <https://www.dmtf.org>.

16 This version supersedes version 1.1 and its errata versions. For a list of the changes, see [ANNEX E \(informative\) change log](#).

## 17 **1.1 Acknowledgments**

---

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

### 19 **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 Natu — Intel Corporation
- Chandra Nelogal — Dell Technologies
- Edward Newman — Hewlett Packard Enterprise



- Alexander Novitskiy — Intel Corporation
- Jim Panian — Qualcomm Inc.
- Scott Phuong — Cisco Systems Inc.
- 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.
- Srikanth Varadarajan — Intel Corporation
- Peng Xiao — Alibaba Group
- Qing Yang — Alibaba Group
- Jiewen Yao — Intel Corporation

## 20 **2 Introduction**

---

21 The *Security Protocol and Data Model (SPDM) Specification* defines [messages](#), data objects, and sequences for performing message exchanges between [devices](#) over a variety of transport and physical media. The description of message exchanges includes [authentication](#) 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-SPDM- and DMTF-defined mechanisms, can use the SPDM.

### 22 **2.1 Advice**

---

23 The authors recommend that readers visit tutorial and education materials at the [Security Protocols and Data Models \(SPDM\)](#) section of the DMTF website before or while reading this specification to help understand this specification.

### 24 **2.2 Conventions**

---

25 The following conventions apply to all SPDM specifications.

#### 26 **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.

#### 27 **2.2.2 Reserved and unassigned values**

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

29 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.

#### 30 **2.2.3 Byte ordering**

31 Unless otherwise specified, for all SPDM specifications [byte](#) ordering of multibyte numeric fields or multibyte bit fields is “little endian” (that is, the lowest byte offset holds the least significant byte, and higher offsets hold the more significant bytes).

**32 2.2.3.1 Hash byte order**

33 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.

34 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 byte 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  $H_7$ .

**35 2.2.3.2 Encoded ASN.1 byte order**

36 For fields or values containing DER, CER, or BER encoded data, SPDM preserves the byte order as described in the [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.

**37 2.2.3.3 Octet string byte order**

38 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 a octet string into an SPDM field, the  $i^{\text{th}}$  byte of the octet string shall be placed in the  $i^{\text{th}}$  offset of that field.

39 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 .

**40 2.2.3.4 Signature byte order**

41 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 the [Edwards-Curve Digital Signature Algorithm](#). Unless otherwise specified, the byte order of a signature for a given signature algorithm shall be the same as the [octet string byte order](#).

#### 42 2.2.3.4.1 ECDSA signatures byte order

43 [FIPS PUB 186-4](#) defines `r`, `s`, and ECDSA signatures to be  $(r, s)$ , where `r` and `s` are 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`.

#### 44 2.2.3.4.2 SM2 signatures byte order

45 [GB/T 32918.2-2016](#) defines `r`, `s`, and SM2 signatures to be  $(r, s)$ , where `r` and `s` are 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](#).

### 46 2.2.4 Sizes and lengths

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

### 48 2.2.5 SPDM data types

49 [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](#).

50 **Table 1 — SPDM data types**

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

### 51 2.2.6 Version encoding

52 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.

53 EXAMPLE:

54 Version 3.7 → 0x37

55 Version 1.0 → 0x10

56 Version 1.2 → 0x12

57 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 interoperation with previous minor versions of the SPDM specification is an implementation-specific decision.

58 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`.

59 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.
- Errata versions (indicated by a non-zero value in the `UpdateVersionNumber` field for the `GET_VERSION` request and `VERSION` response messages) clarify existing behaviors in the SPDM specification. They maintain bitwise compatibility with the base version, except as required to fix security vulnerabilities or to correct mistakes from the base version.

60 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](#).

61 **2.2.7 Notations**

62 SPDM specifications use the following notations:

Notation	Description
<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, 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 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 }::[[S<sub>x</sub>]]</code>	Used mostly in figures, this notation indicates 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 }::[[S<sub>2</sub>]]</code> shows that the Heartbeat message is encrypted and/or authenticated by the keys derived from major secret <code>S<sub>2</sub></code> .

63 **2.2.8 Text or string encoding**

64 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 where 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 on until the last character is placed in the last byte.

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

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

67 **Table 2 — “spdm” encoding example**

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

68 **2.2.9 Deprecated material**

69 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 backwards compatibility. Although implementations can use deprecated elements, they are directed to use the favored elements instead.

70 The following typographical convention indicates deprecated material:

---

71 DEPRECATED

72 Deprecated material appears here.

73 DEPRECATED

---

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

## 75 **3 Scope**

---

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

77 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.



## 78 4 Normative references

---

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

- *ISO/IEC Directives, Part 2, Principles and rules for the structure and drafting of ISO and IEC documents - 2018 (8th edition)*
- DMTF DSP0004, *Common Information Model (CIM) Metamodel*, [https://www.dmtf.org/sites/default/files/standards/documents/DSP0004\\_3.0.1.pdf](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](https://www.dmtf.org/sites/default/files/standards/documents/DSP0223_1.0.1.pdf)
- DMTF DSP0236, *MCTP Base Specification 1.3.0*, [https://www.dmtf.org/sites/default/files/standards/documents/DSP0236\\_1.3.0.pdf](https://www.dmtf.org/sites/default/files/standards/documents/DSP0236_1.3.0.pdf)
- DMTF DSP0239, *MCTP IDs and Codes 1.6.0*, [https://www.dmtf.org/sites/default/files/standards/documents/DSP0239\\_1.6.0.pdf](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](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](https://www.dmtf.org/sites/default/files/standards/documents/DSP1001_1.2.0.pdf)
- IETF TLS DTLS13-43, *The Datagram Transport Layer Security (DTLS) Protocol Version 1.3*, 30 April 2021
- IETF RFC2986, *PKCS #10: Certification Request Syntax Specification*, November 2000
- IETF RFC4716, *The Secure Shell (SSH) Public Key File Format*, November 2006
- IETF RFC5234, *Augmented BNF for Syntax Specifications: ABNF*, January 2008
- IETF RFC5280, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, May 2008
- IETF RFC7250, *Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*, June 2014
- IETF RFC7919, *Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)*, August 2016
- IETF RFC8017, *PKCS #1: RSA Cryptography Specifications Version 2.2*, November 2016
- IETF RFC8032, *Edwards-Curve Digital Signature Algorithm (EdDSA)*, January 2017
- IETF RFC8446, *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 RFC8439, [ChaCha20 and Poly1305 for IETF Protocols](#), June 2018
- IETF RFC8998, [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 186-4 Digital Signature Standard \(DSS\)](#)
  - Appendix D: Recommended Elliptic Curves for Federal Government Use in [FIPS PUB 186-4 Digital Signature Standard \(DSS\)](#)
- [ANSI X9.62](#), 2005
- **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](#)

## 80 5 Terms and definitions

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

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

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

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

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

86 This specification uses these terms:

Term	Definition
alias certificate	Certificate that is dynamically generated by the <i>component</i> 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 exist 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.
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.

Term	Definition
device certificate	Certificate that contains information that identifies the component. Can be a leaf certificate or an <a href="#">intermediate certificate</a> .
device	Physical entity such as a network controller or a fan.
DMTF	Formerly known as the Distributed Management Task Force, DMTF creates open manageability standards that span diverse emerging and traditional information technology (IT) infrastructures, including cloud, virtualization, network, servers, and storage. Member companies and alliance partners worldwide collaborate on standards to improve the interoperable management of IT.
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 <a href="#">GET_ENCAPSULATED_REQUEST request and ENCAPSULATED_REQUEST response messages</a> .
endpoint	Logical entity that communicates with other endpoints over one or more transport protocols.
intermediate certificate	Certificate that is neither a root certificate nor a leaf certificate.
invasive debug mode	A device mode that enables debug access that might expose or allow modification of security-critical firmware, hardware, or settings. Invasive debug mode might include access to the device <a href="#">TCB</a> .
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.
measurement	Representation of firmware/software or configuration data on an endpoint.
message	See <a href="#">SPDM message</a> .
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.
most significant byte (MSB)	Highest order <a href="#">byte</a> in a number consisting of multiple bytes.

Term	Definition
Negotiated State	<p>Set of parameters that represent 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. Nonce may be generated by combining a pseudo random number of at least 64 bits, optionally concatenated with a monotonic counter of size suitable for the application.
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 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. In some instances, a field can be both a payload field and a transport field.
physical transport binding	Specifications that define how a base messaging protocol is implemented on a particular physical transport type and medium, such as SMBus/I <sup>2</sup> C or PCI Express™ Vendor Defined Messaging.
record	A unit or chunk of data that is either encrypted and/or authenticated.
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 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, typically self-signed.
secure session	Provides either or both of encryption or message authentication for communicating data over a transport.

Term	Definition
Security Protocols and Data Models (SPDM)	Working group under DMTF that is responsible for the <i>SPDM Specification</i> , which focuses on enabling authentication, attestation, and key exchange to enhance infrastructure security. In addition to developing the core <i>SPDM Specification</i> , the group collaborates with other standards organizations and developers to support alignment across the industry in the areas of component authentication, confidentiality, and integrity.
session keys	Any secrets, derived cryptographic keys, or any cryptographic information bound to a session.
Session-Secrets-Exchange	Any SPDM request and its 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 <a href="#">Generic SPDM message format</a> .
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 to the security policy.  Reference: <a href="https://en.wikipedia.org/wiki/Trusted_computing_base">https://en.wikipedia.org/wiki/Trusted_computing_base</a>

## 87 6 Symbols and abbreviated terms

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

89 The following additional abbreviations are used in this document.

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

## 90 7 SPDM message exchanges

---

91 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.

92 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*.

93 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.

### 94 7.1 Security capability discovery and negotiation

---

95 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.

### 96 7.2 Identity authentication

---

97 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.

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

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



## 99 7.2.1 Identity provisioning

100 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 device certificate ( `DeviceCert` ) 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.

### 101 7.2.1.1 Device certificate models

102 If trust in a device public key is established through certificates, typically the device certificate ( `DeviceCert` ) is 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 `DeviceCert` either directly or indirectly through a number of intermediate CAs. [Authentication initiators](#) use the `RootCert` to verify the validity of device certificate chains.

103 The certificate chain may be built according to one of two models, both of which are shown in the [Figure 1 — SPDM certificate chain models](#). In one model, shown on the left in the following figure, the leaf certificate is a `DeviceCert` , which contains the public key that corresponds to the device private key. In the other model, shown on the right in the following figure, the leaf certificate is an alias certificate ( `AliasCert` ), in which case there may be one or more intermediate `AliasCert` certificates between the `DeviceCert` and the leaf `AliasCert` . In the `AliasCert` model, the device private key signs the next level `AliasCert` , and then the private key associated with the public key in each `AliasCert` signs the `AliasCert` 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.

104 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 `AliasCert` should be treated as mutable.

105 Through the certificate chain, the root CA indirectly endorses the device public key in the `DeviceCert` . When the `AliasCert` model is in use, the `AliasCert` s are endorsed by the device private key, meaning that the `AliasCert` s are also indirectly endorsed by the root CA.

106 The certificate chain should contain at least one certificate that includes hardware identity information, and the hardware identity information should be present in the device certificate, whether the `DeviceCert` or `AliasCert` model is in use. Though existing deployments cannot include the hardware identity information 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 version of firmware running on the device. The [Hardware identity OID](#) should be used to indicate hardware identity certificates.

107 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 immutable, and 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.

108 In addition, when the `AliasCert` model is used, one or more `AliasCert` s can contain firmware identity information. Other standards bodies might define the format of the firmware identity information. That definition is outside the scope of this specification.

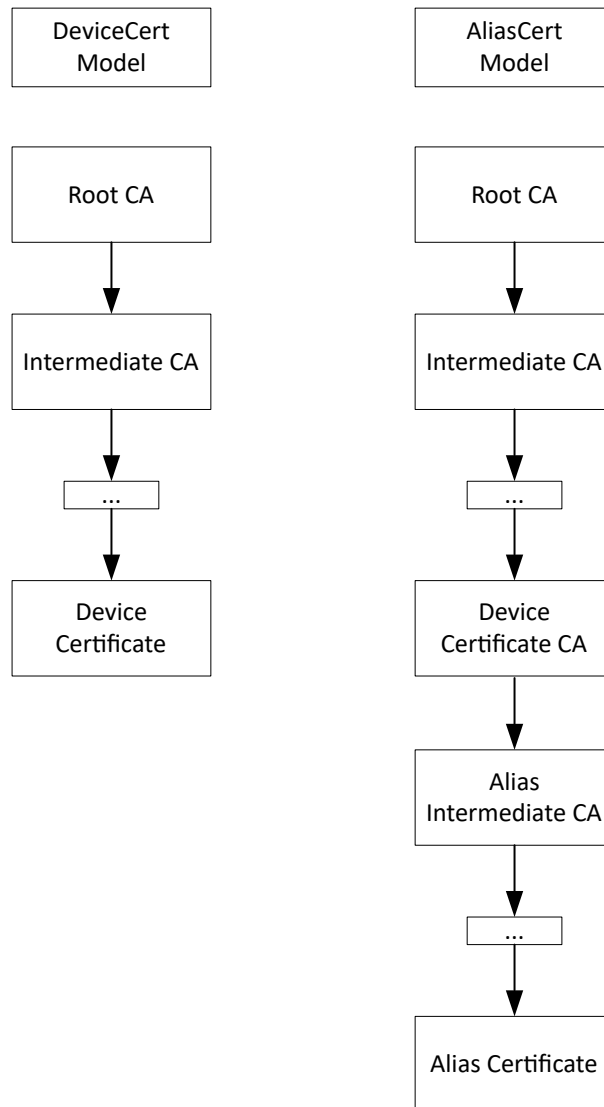
109 Note that a signature algorithm used with a mutable alias certificate can insert random data during signing, which would cause the digest of the certificate chain to change each time it is regenerated. An implementer can use a mechanism that is outside the scope of this specification to ensure that such a signature does not change between reads of the certificate chain.

110 A Responder can use the `DeviceCert` model or the `AliasCert` model. A Requester should be capable of performing [Runtime authentication](#) on a certificate chain that conforms to either model.

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

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

113



114 **7.2.2 Raw public keys**

115 As an alternative to 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. The definition of a trusted environment is outside the scope of this specification.

116 The format of the provisioned public key is out of scope of this specification. Vendors can use proprietary formats or public key formats that other standards define, such as [RFC7250](#) and [RFC4716](#).

### 117 **7.2.3 Runtime authentication**

118 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.

119 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. The transport layer should handle device identification, which is outside the scope of this specification.

## 120 **7.3 Firmware and configuration measurement**

---

121 A measurement is a representation of firmware/software or configuration data on an endpoint. A measurement is typically 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.

## 122 **7.4 Secure sessions**

---

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

124 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, which can be considered equivalent to [Runtime authentication](#). For more details, see the [Session clause](#).

125 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](#).

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

## 127 **7.5 Mutual authentication overview**

---

128 The ability for 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 regard to the Responder but the details also apply to the Requester.

129 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.

## 130 **7.6 Custom environments**

---

131 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 also forfeit certain SPDM features such as interoperability. However, the security posture and guarantees of these environments are out of scope of this specification.

## 132 8 SPDM messaging protocol

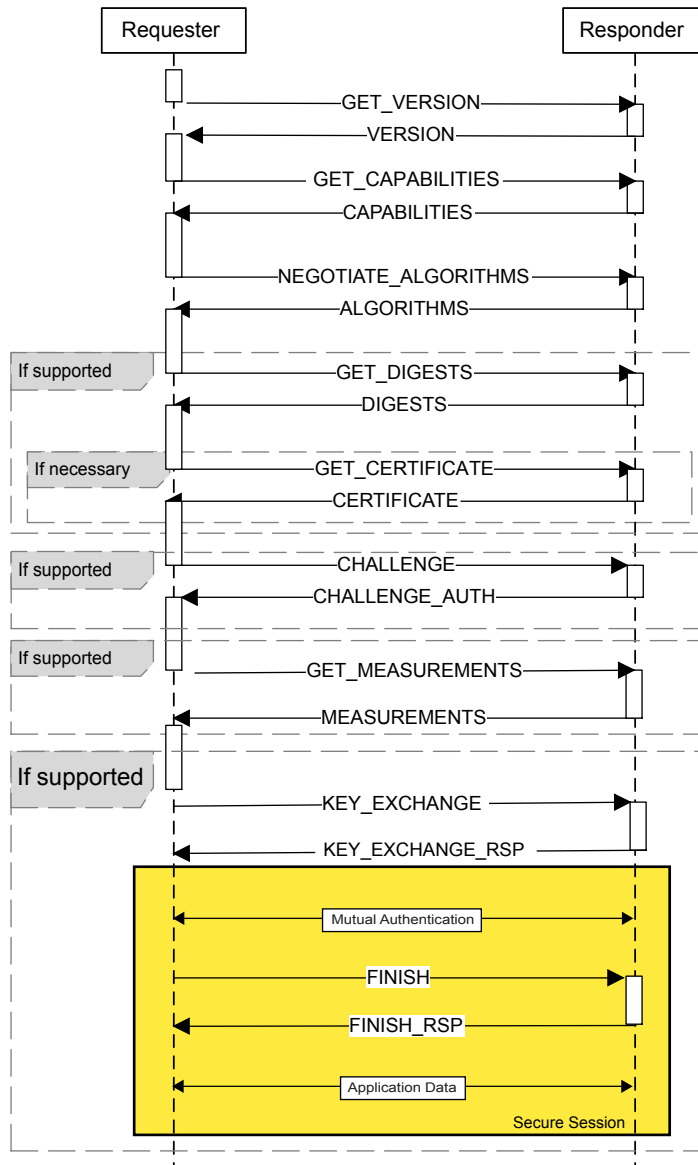
---

133 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.

134 [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.

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

136



137 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.

138 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.

## 139 8.1 SPDM connection model

---

140 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 Requester–Responder pair 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 over that transport.

141 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 separate connections. 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.

142 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.

143 Within a connection, the Requester can establish one or more secure sessions. These secure sessions are considered to be a part of the same connection. Secure sessions can terminate or 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.

## 144 8.2 SPDM bits-to-bytes mapping

---

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

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

147 **Figure 3 — One-byte field bit map**



148 Example: A One-Byte Field

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

149 [Figure 4 — Two-byte field bit map](#) shows the two-byte field bit map:

150 **Figure 4 — Two-byte field bit map**

151 Example: A Two-Byte Field

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

152 **8.3 Generic SPDM message format**

153 [Table 3 — Generic SPDM message field definitions](#) defines the fields that constitute a generic SPDM message, including the message header and payload:

154 **Table 3 — Generic SPDM message field definitions**

Byte offset	Bit offset	Size (bits)	Field	Description
0	[7:4]	4	SPDM Major Version	The major version of the SPDM Specification. An endpoint shall not communicate by using an incompatible SPDM version value. See <a href="#">Version encoding</a> .
0	[3:0]	4	SPDM Minor Version	The minor version of the SPDM Specification. A specification with a given minor version extends a specification with a lower minor version as long as they share the major version. See <a href="#">Version encoding</a> .

Byte offset	Bit offset	Size (bits)	Field	Description
1	[7:0]	8	Request Response Code	The request message code or response code, which <a href="#">Table 4 — SPDM request codes</a> and <a href="#">Table 5 — SPDM response codes</a> 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	The first one-byte parameter. The contents of the parameter is specific to the Request Response Code .
3	[7:0]	8	Param2	The second one-byte parameter. The contents of the parameter is specific to the Request Response Code .
4	See the description.	Variable	<i>SPDM message payload</i>	Zero or more bytes that are specific to the Request Response Code .

### 155 8.3.1 SPDM version

156 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.

157 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.

158 The version of this specification can be found on the title page or the header or footer of each page in this document.

159 The `SPDMVersion` for the version of this specification shall be `0x12`.

160 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"`.

### 161 8.4 SPDM request codes

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

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

164 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 27 — GET_DIGESTS request message format
GET_CERTIFICATE	0x82	Optional	Table 29 — GET_CERTIFICATE request message format
CHALLENGE	0x83	Optional	Table 33 — CHALLENGE request message format
GET_VERSION	0x84	Required	Table 8 — GET_VERSION request message format
CHUNK_SEND	0x85	Optional	Table 81 — CHUNK_SEND request format table
CHUNK_GET	0x86	Optional	Table 85 — CHUNK_GET request format
GET_MEASUREMENTS	0xE0	Optional	Table 38 — GET_MEASUREMENTS request message format
GET_CAPABILITIES	0xE1	Required	Table 11 — GET_CAPABILITIES request message format
NEGOTIATE_ALGORITHMS	0xE3	Required	Table 13 — NEGOTIATE_ALGORITHMS request message format
KEY_EXCHANGE	0xE4	Optional	Table 56 — KEY_EXCHANGE request message format
FINISH	0xE5	Optional	Table 59 — FINISH request message format
PSK_EXCHANGE	0xE6	Optional	Table 61 — PSK_EXCHANGE request message format
PSK_FINISH	0xE7	Optional	Table 63 — PSK_FINISH request message format
HEARTBEAT	0xE8	Optional	Table 65 — HEARTBEAT request message format
KEY_UPDATE	0xE9	Optional	Table 67 — KEY_UPDATE request message format
GET_ENCAPSULATED_REQUEST	0xEA	Optional	Table 70 — GET_ENCAPSULATED_REQUEST request message format

Request	Code value	Implementation requirement	Message format
DELIVER_ENCAPSULATED_RESPONSE	0xEB	Optional	Table 72 — DELIVER_ENCAPSULATED_RESPONSE request message format
END_SESSION	0xEC	Optional	Table 74 — END_SESSION request message format
GET_CSR	0xED	Optional	Table 77 — GET_CSR request message format
SET_CERTIFICATE	0xEE	Optional	Table 79 — SET_CERTIFICATE request message format
VENDOR_DEFINED_REQUEST	0xFE	Optional	Table 53 — VENDOR_DEFINED_REQUEST request message format
RESPOND_IF_READY	0xFF	Required	Table 52 — RESPOND_IF_READY request message format
Reserved	All other values		SPDM implementations compatible with this version shall not use the reserved request codes.

## 166 8.5 SPDM response codes

167 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](#).

168 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.

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

### 170 Table 5 — SPDM response codes

Response	Value	Implementation requirement	Message format
DIGESTS	0x01	Optional	Table 28 — Successful DIGESTS response message format
CERTIFICATE	0x02	Optional	Table 30 — Successful CERTIFICATE response message format
CHALLENGE_AUTH	0x03	Optional	Table 34 — Successful CHALLENGE_AUTH response message format

Response	Value	Implementation requirement	Message format
VERSION	0x04	Required	Table 9 — Successful VERSION response message format
CHUNK_SEND_ACK	0x05	Optional	Table 83 — CHUNK_SEND_ACK response message format
CHUNK_RESPONSE	0x06	Optional	Table 86 — CHUNK_RESPONSE response format
MEASUREMENTS	0x60	Optional	Table 41 — Successful MEASUREMENTS response message format
CAPABILITIES	0x61	Required	Table 12 — Successful CAPABILITIES response message format
ALGORITHMS	0x63	Required	Table 19 — Successful ALGORITHMS response message format
KEY_EXCHANGE_RSP	0x64	Optional	Table 58 — Successful KEY_EXCHANGE_RSP response message format
FINISH_RSP	0x65	Optional	Table 60 — Successful FINISH_RSP response message format
PSK_EXCHANGE_RSP	0x66	Optional	Table 62 — PSK_EXCHANGE_RSP response message format
PSK_FINISH_RSP	0x67	Optional	Table 64 — Successful PSK_FINISH_RSP response message format
HEARTBEAT_ACK	0x68	Optional	Table 66 — HEARTBEAT_ACK response message format
KEY_UPDATE_ACK	0x69	Optional	Table 68 — KEY_UPDATE_ACK response message format
ENCAPSULATED_REQUEST	0x6A	Optional	Table 71 — ENCAPSULATED_REQUEST response message format
ENCAPSULATED_RESPONSE_ACK	0x6B	Optional	Table 73 — ENCAPSULATED_RESPONSE_ACK response message format
END_SESSION_ACK	0x6C	Optional	Table 76 — END_SESSION_ACK response message format

Response	Value	Implementation requirement	Message format
CSR	0x6D	Optional	<a href="#">Table 78 — CSR response message format</a>
SET_CERTIFICATE_RSP	0x6E	Optional	<a href="#">Table 80 — Successful SET_CERTIFICATE_RSP response message format</a>
VENDOR_DEFINED_RESPONSE	0x7E	Optional	<a href="#">Table 54 — VENDOR_DEFINED_RESPONSE response message format</a>
ERROR	0x7F	Required	<a href="#">Table 44 — ERROR response message format</a>
Reserved	All other values		SPDM implementations compatible with this version shall not use the reserved response codes.

## 171 8.6 SPDM request and response code issuance allowance

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

173 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.

174 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.

175 A request and its corresponding response can have the 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.

176 A request and its corresponding response that has 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.

177 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.

178 For details, see the [Session](#) clause.

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

Request	Response	Session	Outside of a session	Session phases
GET_MEASUREMENTS	MEASUREMENTS	Allowed	Allowed	Application Phase
FINISH	FINISH_RSP	Allowed	Conditional (*)	Session Handshake
PSK_FINISH	PSK_FINISH_RSP	Allowed	Prohibited	Session Handshake
HEARTBEAT	HEARTBEAT_ACK	Allowed	Prohibited	Application Phase
KEY_UPDATE	KEY_UPDATE_ACK	Allowed	Prohibited	Application Phase
END_SESSION	END_SESSION_ACK	Allowed	Prohibited	Application Phase
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	Allowed	Allowed	Application Phase
CHUNK_SEND	CHUNK_SEND_ACK	Allowed	Allowed	All Phases
CHUNK_GET	CHUNK_RESPONSE	Allowed	Allowed	All Phases
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
All others	All others	Prohibited	Allowed	Not Applicable

180 (\*) Prohibited when `HANDSHAKE_IN_THE_CLEAR_CAP = 0` , Allowed when `HANDSHAKE_IN_THE_CLEAR_CAP = 1` .

181 For `ERROR` response in the session handshake or application phase of a session, the Requester is only allowed in certain situations to send the `ERROR` response.

## 182 8.7 Concurrent SPDM message processing

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

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

## 185 8.8 Requirements for Requesters

---

- 186 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 messages transfer mechanism](#), a single large SPDM request message and a single `CHUNK_SEND` request can be outstanding at the same time.
- 187 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](#).
- 188 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.
- 189 A Requester might send simultaneous request messages to different Responders.

## 190 8.9 Requirements for Responders

---

- 191 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 messages 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.
- 192 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` response message with `ErrorCode=Busy` or silently discard the request message.
- 193 If a Responder is working on a request message from a Requester, the Responder can respond with `ErrorCode=Busy` .
- 194 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.
-



## 195 9 Timing requirements

---

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

197 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 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.

198 If the transport is not reliable, then the Responder should support retry by identifying whether a received request is a retried one or a new one. If the Responder supports retry, then the response to a retried request shall be identical to the original response. If the transport is reliable, then the Responder may support retry.

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

### 200 9.1 Timing measurements

---

201 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. With the exception of **RDT**, 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.

### 202 9.2 Timing specification table

---

203 The **Ownership** column in [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.

204 **Table 7 — Timing specification for SPDM messages**

Timing parameter	Ownership	Value	Units	Description
RTT	Requester	See the description.	μs	<p>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>Shall be the maximum amount of time the Responder has to provide a response to requests that do not require cryptographic processing, including the <code>GET_CAPABILITIES</code>, <code>GET_VERSION</code>, and <code>NEGOTIATE_ALGORITHMS</code> request messages. If the Responder cannot respond within <code>ST1</code> for a non-cryptographic request other than <code>GET_VERSION</code>, the Responder shall respond with an <code>ERROR</code> response with <code>ErrorCode=ResponseNotReady</code> to request more time.</p> <p>See <a href="#">Table 11</a> — <code>GET_CAPABILITIES</code> request message format, <a href="#">Table 8</a> — <code>GET_VERSION</code> request message format, and <a href="#">Table 13</a> — <code>NEGOTIATE_ALGORITHMS</code> request message format.</p>

Timing parameter	Ownership	Value	Units	Description
T1	Requester	$RTT + ST1$	$\mu s$	<p>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 <sup>CTExponent</sup>	μs	<p>CTExponent is reported in the <code>GET_CAPABILITIES</code> request message and <code>CAPABILITIES</code> response message.</p> <p>This timing parameter shall be the maximum amount of time the endpoint has to provide any response requiring cryptographic processing, including the <code>GET_MEASUREMENTS</code>, <code>CHALLENGE</code>, and <code>SET_CERTIFICATE</code> request messages. If the Responder cannot respond within <code>CT</code>, the Responder shall respond with an <code>ERROR</code> response with <code>ErrorCode=ResponseNotReady</code> to request more time.</p> <p>This parameter is applicable to both a Responder and Requester as the <b>Ownership</b> columns 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>See <a href="#">Table 11 — GET_CAPABILITIES request message format</a>, <a href="#">Table 12 — Successful CAPABILITIES response message format</a>, <a href="#">Table 38 — GET_MEASUREMENTS request message format</a>, <a href="#">Table 33 — CHALLENGE request message format</a>, and <a href="#">Table 79 — SET_CERTIFICATE request message format</a>.</p>

Timing parameter	Ownership	Value	Units	Description
T2	Requester	RTT + CT	μs	<p>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 CT timing parameter.</p>
RDT	Responder	$2^{\text{RDTEExponent}}$	μs	<p>Recommended additional amount of time, in microseconds that the Responder needs to complete the requested cryptographic operation. When the Responder cannot complete cryptographic processing response within the CT time, it shall provide RDTEExponent as part of the ERROR response as <a href="#">Table 44 — ERROR response message format</a> shows. See <a href="#">Table 46 — ResponseNotReady extended error data</a> for the RDTEExponent value.</p> <p>For details, see <a href="#">ErrorCode=ResponseNotReady</a> in <a href="#">Table 46 — ResponseNotReady extended error data</a>. An SPDM Responder measures the RDT value from the end of the transmission of the ERROR message of <a href="#">ErrorCode=ResponseNotReady</a>, to the beginning of the reception of the next <a href="#">RESPOND_IF_READY</a> request message.</p>

Timing parameter	Ownership	Value	Units	Description
WT	Requester	RDT	μs	<p>Amount of time that the Requester should wait before issuing the RESPOND_IF_READY request message as <a href="#">Table 52 — RESPOND_IF_READY request message format</a> shows.</p> <p>The Requester shall measure this timing parameter from the reception of the ERROR response to the transmission of RESPOND_IF_READY request. The Requester can include the transmission time of the ERROR from the Responder to Requester as time spent waiting for WT to expire. For example, if a Responder indicates WT is two seconds and the ERROR response takes one second to transport to the Requester, the Requester only needs to wait an additional one second upon reception of the ERROR response.</p> <p>For details, see the RDT timing parameter.</p>

Timing parameter	Ownership	Value	Units	Description
WT <sub>Max</sub>	Requester	(RDT*RDTM)-RTT	μs	<p>Maximum wait time the Requester has to issue the RESPOND_IF_READY request message, as <a href="#">Table 52 — RESPOND_IF_READY request message format</a> shows, unless the Requester issued a successful RESPOND_IF_READY request message, as <a href="#">Table 52 — RESPOND_IF_READY request message format</a> shows, earlier. The RESPOND_IF_READY message follows the most recently received ERROR message with ErrorCode = ResponseNotReady, which shall specify the wait time for that cycle. The Requester shall start measuring time from the reception of the first ERROR message with ErrorCode = ResponseNotReady with the same Token until WT<sub>Max</sub> μs elapses or the corresponding Response is successfully received.</p> <p>After this time the Responder is allowed to drop the response. The Requester shall take into account the transmission time of the ERROR response, as <a href="#">Table 44 — ERROR response message format</a> shows, from the Responder to Requester when calculating WT<sub>Max</sub>.</p> <p>The RDTM value appears in <a href="#">Table 46 — ResponseNotReady extended error data</a>.</p> <p>The Responder should ensure that WT<sub>Max</sub> does not result in less than WT in determination of RDTM.</p> <p>See ErrorCode=ResponseNotReady in <a href="#">Table 46 — ResponseNotReady extended</a></p>

Timing parameter	Ownership	Value	Units	Description
				error data.
HeartbeatPeriod	Requester and Responder	Variable	s	See the <a href="#">HEARTBEAT request</a> and <a href="#">HEARTBEAT_ACK response</a> clause.



## 205 10 SPDM messages

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

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

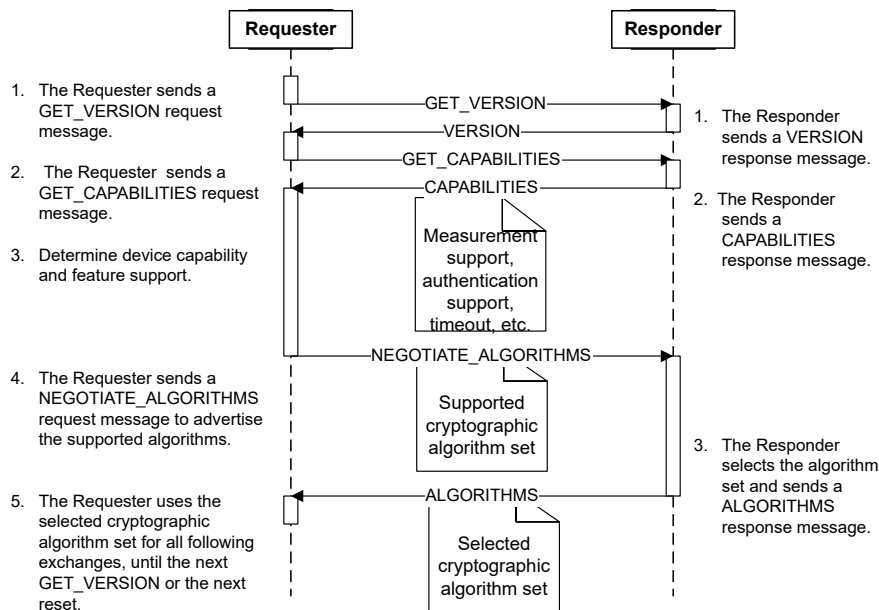
### 207 10.1 Capability discovery and negotiation

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

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

210 **Figure 5 — Capability discovery and negotiation flow**

211



#### 212 10.1.1 Negotiated state preamble

213 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.

214 If the two endpoints do not support session key establishment with the PSK (Pre-Shared Key) option, or 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` . 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.

215 If the two endpoints support session key establishment with the PSK and the negotiated capabilities and algorithms (the `C` and `A` of the `VCA` ) are provisioned to both endpoints alongside the PSK, then the Requester shall not issue `GET_CAPABILITIES` and `NEGOTIATE_ALGORITHMS` .

## 216 10.2 GET\_VERSION request and VERSION response messages

---

217 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.

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

219 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.

220 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 except for `ErrorCode` s specified in this clause. 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 different version than the selected version in other Requests, the Responder should return an `ERROR` message with `ErrorCode = VersionMismatch` or the Responder can silently discard the Request.

221 A Requester can issue a `GET_VERSION` request message to a Responder at any time, which is as an exception to [Requirements for Requesters](#) to allow for scenarios where a Requester shall restart the protocol due to an internal error or Reset.

222 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, 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

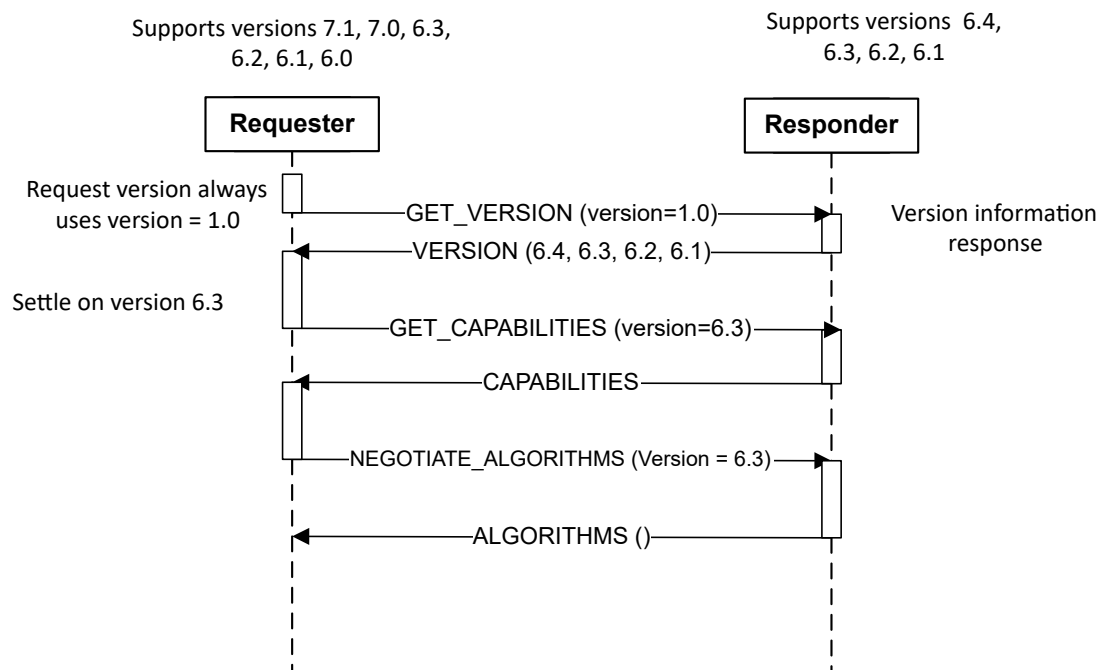
Requester shall not affect the session state. The `ERROR` message resulting from an invalid `GET_VERSION` request shall have the value of the `SPDMVersion` field set to `0x10`.

223 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 silently discard any request received outside of a session, or respond with `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 57 — Session Policy](#) at the time of session establishment.

224 [Figure 6 — Discovering the common major version](#) shows the process:

225 **Figure 6 — Discovering the common major version**

226



227 [Table 8 — GET\\_VERSION request message format](#) shows the `GET_VERSION` request message format:

228 **Table 8 — GET\_VERSION request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be <code>0x10</code> (V1.0).
1	RequestResponseCode	1	<code>0x84</code> = <code>GET_VERSION</code> . See <a href="#">Table 4 — SPDM request codes</a> .
2	Param1	1	Reserved.

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

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

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

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be <code>0x10</code> (V1.0).
1	RequestResponseCode	1	<code>0x04</code> = <code>VERSION</code> . See <a href="#">Table 5 — SPDM response codes</a> .
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 <a href="#">Table 10 — VersionNumberEntry definition</a> .

231 [Table 10 — VersionNumberEntry definition](#) shows the `VersionNumberEntry` definition. See [Version encoding](#) for more details.

232 **Table 10 — VersionNumberEntry definition**

Bit offset	Field	Description
[15:12]	MajorVersion	Version of the specification with changes that are incompatible with one or more functions in earlier major versions of the specification.
[11:8]	MinorVersion	Version of the specification with changes that are compatible with functions in earlier minor versions of this major version specification.
[7:4]	UpdateVersionNumber	Version of the specification with editorial updates and errata fixes. Informational; ignore when checking versions for interoperability.
[3:0]	Alpha	Pre-release work-in-progress version of the specification. Backward compatible with earlier minor versions of this major version specification. However, because the <code>Alpha</code> value represents an in-development version of the specification, versions that share the same major and minor version numbers but have different <code>Alpha</code> versions might not be fully interoperable. Released versions shall have an <code>Alpha</code> value of zero ( <code>0</code> ).

233 **10.3 GET\_CAPABILITIES request and CAPABILITIES response messages**

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

235 [Table 11 — GET\\_CAPABILITIES request message format](#) shows the GET\_CAPABILITIES request message format.

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

237 [Table 11 — Flag fields definitions for the Requester](#) shows the flag fields definitions for the Requester.

238 Likewise, [Table 12 — Flag fields definitions for the Responder](#) shows the flag fields definitions for the Responder.

239 To properly support transferring of SPDM messages, the Requester and Responder shall indicate two buffer sizes:

- One for receiving a single SPDM message called DataTransferSize .
- One for indicating their maximum internal buffer size for processing a single received SPDM message called MaxSPDMmsgSize .

240 Additionally, the Requester and Responder can have a transmit buffer. The transmit buffer size is not communicated to the other SPDM endpoint but can be less than the DataTransferSize of the receiving SPDM endpoint.

241 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, referred to as MinDataTransferSize , shall be the size, in bytes, of the SPDM message with the largest size in this list:

- GET\_VERSION
- VERSION assuming no versions returned contain 'Alpha' versions in 'VersionNumberEntry' and version entries are not duplicated.
- GET\_CAPABILITIES
- CAPABILITIES
- 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

242 The calculation of MinDataTransferSize shall assume all fields are present. For this version of the specification, the MinDataTransferSize shall be 42.

243 **GET\_CAPABILITIES request 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	0xE1 = GET_CAPABILITIES . See <a href="#">Table 4 — SPDM request codes</a> .
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	Reserved	1	Reserved.
5	CTExponent	1	<p>Shall be exponent of base 2, which is used to calculate <math>CT</math> .</p> <p>See <a href="#">Table 7 — Timing specification for SPDM messages</a>.</p> <p>The equation for <math>CT</math> shall be <math>2^{CTExponent}</math> microseconds (<math>\mu s</math>).</p> <p>For example, if <math>CTExponent</math> is 10 , <math>CT</math> is <math>2^{10} = 1024 \mu s</math> .</p>
6	Reserved	2	Reserved.
8	Flags	4	See <a href="#">Table 11 — Flag fields definitions for the Requester</a> .
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 $MinDataTransferSize$ . The $DataTransferSize$ shall exclude transport headers, encryption headers, and MAC. This field helps the sender of the SPDM message know whether or not it needs to utilize the <a href="#">Large SPDM message transfer mechanism</a> .
16	MaxSPDMmsgSize	4	<p>If the Requester supports the <a href="#">Large SPDM message transfer mechanism</a> this field shall indicate the maximum size, in bytes, of the internal buffer used to reassemble a single and complete <a href="#">Large SPDM message</a>. This field shall be greater than or equal to <math>DataTransferSize</math> . 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 Requester does not support the Large SPDM message transfer mechanism, this field shall be equal to the <math>DataTransferSize</math> of the Requester.</p>

244 **Successful CAPABILITIES response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0x61</code> = CAPABILITIES. See <a href="#">Table 5 — SPDM response codes</a> .
2	Param1	1	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 <a href="#">Table 7 — Timing specification for SPDM messages</a>.</p> <p>The equation for <code>CT</code> shall be <math>2^{\text{CTExponent}}</math> microseconds (<math>\mu\text{s}</math>).</p> <p>For example, if <code>CTExponent</code> is <code>10</code>, <code>CT</code> is <math>2^{10} = 1024 \mu\text{s}</math>.</p>
6	Reserved	2	Reserved.
8	Flags	4	See <a href="#">Table 12 — Flag fields definitions for the Responder</a> .
12	DataTransferSize	4	<p>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 MAC. This field helps the sender of the SPDM message know whether or not it needs to utilize the <a href="#">Large SPDM message transfer mechanism</a>.</p>

Byte offset	Field	Size (bytes)	Description
16	MaxSPDMmsgSize	4	<p>If the Responder supports the <a href="#">Large SPDM message transfer mechanism</a> this field shall indicate the maximum size, in bytes, of the internal buffer used to reassemble a single and complete <a href="#">Large SPDM message</a>. 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>

245 As described in other parts of this specification, a Requester or Responder can reverse roles or be both roles for certain SPDM messages and flows. Thus, in general, 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 receiving 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 `ErrorCode=ResponseTooLarge`. Likewise, if the size of a request message exceeds the size of the `MaxSPDMmsgSize` of the responding SPDM endpoint, the responding SPDM endpoint shall respond with `ErrorCode=RequestTooLarge` or silently discard the request. Additionally, an SPDM endpoint is expected to provide graceful error handling (for example, buffer overflow/underflow protection) in the event they receive an SPDM messages that exceed their `MaxSPDMmsgSize`.

246 [Table 11 — Flag fields definitions for the Requester](#) shows the flag fields definitions for the Requester.

247 Unless otherwise stated, if a Requester indicates support of 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 `CERT_CAP` bit to `1`, the Requester can receive a `GET_CERTIFICATE` request and send back a successful `CERTIFICATE` response message.

248 **Table 11 — Flag fields definitions for the Requester**

Byte offset	Bit offset	Field	Description
0	0	Reserved	Reserved.
0	1	CERT_CAP	If set, Requester shall support <code>DIGESTS</code> and <code>CERTIFICATE</code> response messages.
0	2	CHAL_CAP	<b>DEPRECATED:</b> If set, Requester shall support <code>CHALLENGE_AUTH</code> response message.
0	[5:3]	Reserved	Reserved.



Byte offset	Bit offset	Field	Description
0	6	ENCRYPT_CAP	If set, Requester shall support 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.
0	7	MAC_CAP	If set, Requester shall support 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. <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.
1	0	MUT_AUTH_CAP	If set, Requester shall support mutual authentication.
1	1	KEY_EX_CAP	If set, Requester shall support <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	Pre-shared key capabilities of the Requester. <ul style="list-style-type: none"> <li>• <code>00b</code> . Requester shall not support pre-shared key capabilities.</li> <li>• <code>01b</code> . Requester shall support pre-shared key</li> <li>• <code>10b</code> and <code>11b</code> . Reserved.</li> </ul> If supported, <code>ENCRYPT_CAP</code> or <code>MAC_CAP</code> shall be set.
1	4	ENCAP_CAP	If set, Requester shall support <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.  <code>ENCAP_CAP</code> was previously deprecated because <a href="#">Basic mutual authentication</a> is deprecated. Deprecation is removed since other messages may require <code>ENCAP_CAP</code> such as <code>KEY_UPDATE</code> which does not require mutual authentication.
1	5	HBEAT_CAP	If set, Requester shall support <code>HEARTBEAT</code> messages.
1	6	KEY_UPD_CAP	If set, Requester shall support <code>KEY_UPDATE</code> messages.

Byte offset	Bit offset	Field	Description
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, 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 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 shall support <a href="#">Large SPDM message transfer mechanism</a> messages.
2	[7:2]	Reserved	Reserved.
3	[7:0]	Reserved	Reserved.

249 [Table 12 — Flag fields definitions for the Responder](#) shows the flag fields definitions for the Responder.

250 Unless otherwise stated, if a Responder indicates support of 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 `CERT_CAP` bit to `1`, the Responder can receive a `GET_CERTIFICATE` request and send back a successful `CERTIFICATE` response message.

251 **Table 12 — Flag fields definitions for the Responder**

Byte offset	Bit offset	Field	Description
0	0	CACHE_CAP	If set, the Responder shall support 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 shall support DIGESTS and CERTIFICATE response messages.
0	2	CHAL_CAP	If set, Responder shall support CHALLENGE_AUTH response message.
0	[4:3]	MEAS_CAP	<p>MEASUREMENTS response capabilities of the Responder.</p> <ul style="list-style-type: none"> <li>00b . The Responder shall not support MEASUREMENTS response capabilities.</li> <li>01b . The Responder shall support MEASUREMENTS response but cannot perform signature generation.</li> <li>10b . The Responder shall support MEASUREMENTS response and can generate signatures.</li> <li>11b . Reserved.</li> </ul> <p>Note that, apart from affecting MEASUREMENTS, this capability also affects Param2 of CHALLENGE, Param1 of KEY_EXCHANGE, Param1 of PSK_EXCHANGE, MeasurementSummaryHash field of KEY_EXCHANGE_RSP, CHALLENGE_AUTH, PSK_EXCHANGE_RSP. See the respective request and response clauses for further details.</p>
0	5	MEAS_FRESH_CAP	<p>0 . As part of MEASUREMENTS response message, the Responder may return MEASUREMENTS that were computed during the last Responder's Reset.</p> <ul style="list-style-type: none"> <li>1 . The Responder shall support recomputing all MEASUREMENTS without requiring a Reset, and shall always return fresh MEASUREMENTS as part of MEASUREMENTS response message.</li> </ul>

Byte offset	Bit offset	Field	Description
0	6	ENCRYPT_CAP	If set, Responder shall support message encryption in a secure session. If set, PSK_CAP or KEY_EX_CAP shall be set accordingly to indicate support.
0	7	MAC_CAP	If set, Responder shall support message authentication in a secure session. If set, PSK_CAP or KEY_EX_CAP shall be set accordingly to indicate support. 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 shall support mutual authentication.
1	1	KEY_EX_CAP	If set, Responder shall support KEY_EXCHANGE_RSP response message. If set, ENCRYPT_CAP or MAC_CAP shall be set.
1	[3:2]	PSK_CAP	<p>Pre-Shared Key capabilities of the Responder.</p> <ul style="list-style-type: none"> <li>00b . Responder shall not support Pre-Shared Key capabilities.</li> <li>01b . Responder shall support Pre-Shared Key without ResponderContext for session key derivation.</li> <li>10b . Responder shall support Pre-Shared Key with ResponderContext for session key derivation.</li> <li>11b . Reserved.</li> </ul> <p>If supported, ENCRYPT_CAP or MAC_CAP shall be set.</p>
1	4	ENCAP_CAP	<p>If set, Responder shall support 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 other messages may require ENCAP_CAP such as KEY_UPDATE which does not require mutual authentication.</p>

Byte offset	Bit offset	Field	Description
1	5	HBEAT_CAP	If set, Responder shall support <code>HEARTBEAT</code> messages.
1	6	KEY_UPD_CAP	If set, Responder shall support <code>KEY_UPDATE</code> 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, <code>KEY_EX_CAP</code> shall also be set. Setting this bit leads to changes in the contents of certain SPDM messages, discussed in other parts of this specification.</p> <p>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 <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 Responder was provisioned to the Requester. The transport layer is responsible for identifying the Requester. In this case, <code>CERT_CAP</code> and <code>ALIAS_CERT_CAP</code> of the Responder shall both be 0.
2	1	CHUNK_CAP	If set, Responder shall support <a href="#">Large SPDM message transfer mechanism</a> messages.
2	2	ALIAS_CERT_CAP	If set, the Responder uses the <code>AliasCert</code> model. See <a href="#">Identity provisioning</a> for details.
2	3	SET_CERT_CAP	If set, Responder shall support <code>SET_CERTIFICATE_RSP</code> response messages.
2	4	CSR_CAP	If set, Responder shall support <code>CSR</code> response messages. If this bit is set, <code>SET_CERT_CAP</code> shall be set.
2	5	CERT_INSTALL_RESET_CAP	If set, Responder may return an <code>ERROR</code> message with <code>ErrorCode = ResetRequired</code> to complete a certificate provisioning request. If this bit is set, <code>CSR_CAP</code> and/or <code>SET_CERT_CAP</code> shall be set.

Byte offset	Bit offset	Field	Description
2	[7:6]	Reserved	Reserved.
3	[7:0]	Reserved	Reserved.

252 In the case where an SPDM implementation incorrectly returns an illegal combination of capability flags, as these 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 and should set the `ErrorCode = InvalidRequest`. If a Requester detects an illegal capability flag combination reported by the Responder it should retry once by issuing `GET_VERSION` and `GET_CAPABILITIES`. If the illegal combination is returned again it should cease communicating with this particular Responder over SPDM and log an error in an implementation-specific manner to assist with identifying the problem.

## 253 10.4 NEGOTIATE\_ALGORITHMS request and ALGORITHMS response messages

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

255 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.

256 [Table 13 — NEGOTIATE\\_ALGORITHMS request message format](#) shows the `NEGOTIATE_ALGORITHMS` request message format.

257 **Table 13 — NEGOTIATE\_ALGORITHMS request message format**

Byte offset	Field	Size (bytes)	Description
0	<code>SPDMVersion</code>	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	<code>RequestResponseCode</code>	1	<code>0xE3</code> = <code>NEGOTIATE_ALGORITHMS</code> . See <a href="#">Table 4 — SPDM request codes</a> .
2	<code>Param1</code>	1	Number of algorithms structure tables in this request using <code>ReqAlgStruct</code>
3	<code>Param2</code>	1	Reserved.
4	<code>Length</code>	2	Length of the entire request message, in bytes. Length shall be less than or equal to 128 bytes.

Byte offset	Field	Size (bytes)	Description
6	MeasurementSpecification	1	<p>Bit mask. The measurement specification is used in the <code>MEASUREMENTS</code> response. The Requester can set zero or one bit to indicate the measurement specification support.</p> <ul style="list-style-type: none"><li>• Bit 0: This bit shall indicate support for the DMTF-defined measurement specification. See <a href="#">Table 43 — DMTF measurement specification format</a>.</li></ul>
7	OtherParamsSupport	1	<p>Selection Bit mask.</p> <p>Bits [3:0] - See <a href="#">Opaque Data Format Support and Selection Table</a></p> <p>Bits [7:4] - Reserved.</p>

Byte offset	Field	Size (bytes)	Description
8	BaseAsymAlgo	4	<p>Bit mask listing Requester-supported SPDM-enumerated asymmetric key signature algorithms for the purpose of signature verification. If the capabilities do not support this algorithm, this value is not used and shall be set to zero. Let <code>SigLen</code> be the size of the signature in bytes.</p> <ul style="list-style-type: none"> <li>• Byte 0 Bit 0. <a href="#">TPM_ALG_RSASSA_2048</a> where <code>SigLen = 256</code>.</li> <li>• Byte 0 Bit 1. <a href="#">TPM_ALG_RSAPSS_2048</a> where <code>SigLen = 256</code>.</li> <li>• Byte 0 Bit 2. <a href="#">TPM_ALG_RSASSA_3072</a> where <code>SigLen = 384</code>.</li> <li>• Byte 0 Bit 3. <a href="#">TPM_ALG_RSAPSS_3072</a> where <code>SigLen = 384</code>.</li> <li>• Byte 0 Bit 4. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P256</a> where <code>SigLen = 64</code> (32-byte <code>r</code> followed by 32-byte <code>s</code>).</li> <li>• Byte 0 Bit 5. <a href="#">TPM_ALG_RSASSA_4096</a> where <code>SigLen = 512</code>.</li> <li>• Byte 0 Bit 6. <a href="#">TPM_ALG_RSAPSS_4096</a> where <code>SigLen = 512</code>.</li> <li>• Byte 0 Bit 7. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P384</a> where <code>SigLen = 96</code> (48-byte <code>r</code> followed by 48-byte <code>s</code>).</li> <li>• Byte 1 Bit 0. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P521</a> where <code>SigLen = 132</code> (66-byte <code>r</code> followed by 66-byte <code>s</code>).</li> <li>• Byte 1 Bit 1. <a href="#">TPM_ALG_SM2_ECC_SM2_P256</a> where <code>SigLen = 64</code> (32-byte <code>SM2_R</code> followed by 32-byte <code>SM2_S</code>).</li> <li>• Byte 1 Bit 2. <a href="#">EdDSA ed25519</a> where <code>SigLen = 64</code> (32-byte <code>R</code> followed by 32-byte <code>S</code>).</li> <li>• Byte 1 Bit 3. <a href="#">EdDSA ed448</a> where <code>SigLen = 114</code> (57-byte <code>R</code> followed by 57-byte <code>S</code>).</li> <li>• All other values reserved.</li> </ul>



Byte offset	Field	Size (bytes)	Description
12	BaseHashAlgo	4	Bit mask listing Requester-supported SPDM- enumerated cryptographic hashing algorithms. If the capabilities do not support this algorithm, this value is not used and shall be set to zero. <ul style="list-style-type: none"> <li>• Byte 0 Bit 0. <a href="#">TPM_ALG_SHA_256</a></li> <li>• Byte 0 Bit 1. <a href="#">TPM_ALG_SHA_384</a></li> <li>• Byte 0 Bit 2. <a href="#">TPM_ALG_SHA_512</a></li> <li>• Byte 0 Bit 3. <a href="#">TPM_ALG_SHA3_256</a></li> <li>• Byte 0 Bit 4. <a href="#">TPM_ALG_SHA3_384</a></li> <li>• Byte 0 Bit 5. <a href="#">TPM_ALG_SHA3_512</a></li> <li>• Byte 0 Bit 6. <a href="#">TPM_ALG_SM3_256</a></li> <li>• All other values reserved.</li> </ul>
16	Reserved	12	Reserved.
28	ExtAsymCount	1	Number of Requester-supported extended asymmetric key signature algorithms (=A) for the purpose of signature verification. $A + E +$ $ExtAlgCount2 + ExtAlgCount3 + ExtAlgCount4$ $+ ExtAlgCount5$ shall be less than or equal to 20. If the capabilities do not support this algorithm, this value is not used and shall be set to zero.
29	ExtHashCount	1	Number of Requester-supported extended hashing algorithms (=E). $A + E + ExtAlgCount2 +$ $ExtAlgCount3 + ExtAlgCount4 + ExtAlgCount5$ shall be less than or equal to 20. If the capabilities do not support this algorithm, this value is not used and shall be set to zero.
30	Reserved	2	Reserved.
32	ExtAsym	$4 * A$	List of Requester-supported extended asymmetric key signature algorithms for the purpose of signature verification. <a href="#">Table 25 — Extended Algorithm field format</a> describes the format of this field.
$32 + 4 * A$	ExtHash	$4 * E$	List of the extended hashing algorithms supported by Requester. <a href="#">Table 25 — Extended Algorithm field format</a> describes the format of this field.
$32 + 4 * A + 4 * E$	ReqAlgStruct	AlgStructSize	See the <code>AlgStructure</code> request field.

258 `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.

259 [Table 14 — Algorithm request structure](#) shows the Algorithm request structure:

260 **Table 14 — Algorithm request structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	Type of algorithm. <ul style="list-style-type: none"> <li>• 0 and 1. Reserved.</li> <li>• 2. DHE.</li> <li>• 3. AEADCipherSuite .</li> <li>• 4. ReqBaseAsymAlg .</li> <li>• 5. KeySchedule .</li> <li>• All other values reserved.</li> </ul>
1	AlgCount	1	Requester supported fixed algorithms. <ul style="list-style-type: none"> <li>• Bit [7:4]. Number of bytes required to describe Requester supported SPDM-enumerated fixed algorithms (= FixedAlgCount ). FixedAlgCount + 2 shall be a multiple of 4.</li> <li>• Bit [3:0]. Number of Requester-supported extended algorithms (= ExtAlgCount ).</li> </ul>
2	AlgSupported	FixedAlgCount	Bit mask listing Requester-supported SPDM-enumerated algorithms.
2 + FixedAlgCount	AlgExternal	4* ExtAlgCount	List of Requester-supported extended algorithms. <a href="#">Table 25 — Extended Algorithm field format</a> describes the format of this field.

261 The following tables describe the Algorithm request structure mapped to their respective type:

- [Table 15 — DHE structure](#)
- [Table 16 — AEAD structure](#)
- [Table 17 — ReqBaseAsymAlg structure](#)
- [Table 18 — KeySchedule structure](#)

262 **Table 15 — DHE structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	0x02 = DHE
1	AlgCount	1	<ul style="list-style-type: none"> <li>• Bit [7:4]. Shall be a value of 2.</li> <li>• Bit [3:0]. Number of Requester-supported extended DHE groups (= ExtAlgCount2 ).</li> </ul>

Byte offset	Field	Size (bytes)	Description
2	AlgSupported	2	<p>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.</p> <ul style="list-style-type: none"> <li>• Byte 0 Bit 0. <a href="#">ffdhe2048</a> (D = 256).</li> <li>• Byte 0 Bit 1. <a href="#">ffdhe3072</a> (D = 384).</li> <li>• Byte 0 Bit 2. <a href="#">ffdhe4096</a> (D = 512).</li> <li>• Byte 0 Bit 3. <a href="#">secp256r1</a> (D = 64, C = 32).</li> <li>• Byte 0 Bit 4. <a href="#">secp384r1</a> (D = 96, C = 48).</li> <li>• Byte 0 Bit 5. <a href="#">secp521r1</a> (D = 132, C = 66).</li> <li>• Byte 0 Bit 6. <a href="#">SM2_P256 (Part 3 and Part 5 of GB/T 32918 specification)</a> (D = 64, C = 32).</li> <li>• All other values reserved.</li> </ul>
4	AlgExternal	4* ExtAlgCount2	<p>List of Requester-supported extended DHE groups. <a href="#">Table 25 — Extended Algorithm field format</a> describes the format of this field.</p>

263 **Table 16 — AEAD structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	0x03 = AEAD
1	AlgCount	1	<ul style="list-style-type: none"> <li>• Bit [7:4]. Shall be a value of 2.</li> <li>• Bit [3:0]. Number of Requester supported extended <a href="#">AEAD</a> algorithms (= ExtAlgCount3 ).</li> </ul>
2	AlgSupported	2	<p>Bit mask listing Requester-supported SPDM-enumerated AEAD algorithms.</p> <ul style="list-style-type: none"> <li>• Byte 0 Bit 0. <a href="#">AES-128-GCM</a>. 128-bit key; 96-bit IV (initialization vector); tag size is specified by transport layer.</li> <li>• Byte 0 Bit 1. <a href="#">AES-256-GCM</a>. 256-bit key; 96-bit IV; tag size is specified by transport layer.</li> <li>• Byte 0 Bit 2. <a href="#">CHACHA20_POLY1305</a>. 256-bit key; 96-bit IV; 128-bit tag.</li> <li>• Byte 0 Bit 3. <a href="#">AEAD_SM4_GCM</a>. 128-bit key; 96-bit IV; tag size is specified by transport layer.</li> <li>• All other values reserved.</li> </ul>
4	AlgExternal	4* ExtAlgCount3	<p>List of Requester-supported extended AEAD algorithms. <a href="#">Table 25 — Extended Algorithm field format</a> describes the format of this field.</p>

264 **Table 17 — ReqBaseAsymAlg structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	0x04 = ReqBaseAsymAlg
1	AlgCount	1	<ul style="list-style-type: none"> <li>Bit [7:4]. Shall be a value of 2.</li> <li>Bit [3:0]. Number of Requester supported extended asymmetric key signature algorithms for the purpose of signature generation (= ExtAlgCount4 ).</li> </ul>
2	AlgSupported	2	Bit mask listing Requester-supported SPDM-enumerated asymmetric key signature algorithms for the purpose of signature generation. <ul style="list-style-type: none"> <li>Byte 0 Bit 0. <a href="#">TPM_ALG_RSASSA_2048</a>.</li> <li>Byte 0 Bit 1. <a href="#">TPM_ALG_RSAPSS_2048</a>.</li> <li>Byte 0 Bit 2. <a href="#">TPM_ALG_RSASSA_3072</a>.</li> <li>Byte 0 Bit 3. <a href="#">TPM_ALG_RSAPSS_3072</a>.</li> <li>Byte 0 Bit 4. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P256</a>.</li> <li>Byte 0 Bit 5. <a href="#">TPM_ALG_RSASSA_4096</a>.</li> <li>Byte 0 Bit 6. <a href="#">TPM_ALG_RSAPSS_4096</a>.</li> <li>Byte 0 Bit 7. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P384</a>.</li> <li>Byte 1 Bit 0. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P521</a>.</li> <li>Byte 1 Bit 1. <a href="#">TPM_ALG_SM2_ECC_SM2_P256</a>.</li> <li>Byte 1 Bit 2. <a href="#">EdDSA ed25519</a>.</li> <li>Byte 1 Bit 3. <a href="#">EdDSA ed448</a>.</li> <li>All other values reserved.</li> </ul>
4	AlgExternal	4* ExtAlgCount4	List of Requester-supported extended asymmetric key signature algorithms for the purpose of signature generation. <a href="#">Table 25 — Extended Algorithm field format</a> describes the format of this field.

265

**Table 18 — KeySchedule structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	0x05 = KeySchedule
1	AlgCount	1	<ul style="list-style-type: none"> <li>Bit [7:4]. Shall be a value of 2.</li> <li>Bit [3:0]. Number of Requester supported extended key schedule algorithms (= ExtAlgCount5 ).</li> </ul>
2	AlgSupported	2	Bit mask listing Requester-supported SPDM-enumerated Key Schedule algorithms. <ul style="list-style-type: none"> <li>Byte 0 Bit 0. SPDM Key Schedule.</li> <li>All other values reserved.</li> </ul>

Byte offset	Field	Size (bytes)	Description
4	AlgExternal	4* ExtAlgCount5	List of Requester-supported extended key schedule algorithms. <a href="#">Table 25 — Extended Algorithm field format</a> describes the format of this field.

266 [Table 19 — ALGORITHMS response message format](#) shows the `ALGORITHMS` response message format.

267 **Table 19 — Successful ALGORITHMS response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0x63</code> = <code>ALGORITHMS</code> . See <a href="#">Table 5 — SPDM response codes</a> .
2	Param1	1	Number of algorithms structure tables in this response using <code>RespAlgStruct</code> .
3	Param2	1	Reserved.
4	Length	2	Length of the response message, in bytes.
6	MeasurementSpecificationSel	1	Bit mask. The Responder shall select one of the measurement specifications supported by the Requester and Responder. Thus, no more than one bit shall be set. The <code>MeasurementSpecification</code> field in <code>NEGOTIATE_ALGORITHMS</code> defines the format of this field.
7	OtherParamsSelection	1	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"> <li>• Bit [3:0]. See <a href="#">Opaque Data Format Support and Selection Table</a>.</li> <li>• Bit [7:4]. Reserved.</li> </ul>

Byte offset	Field	Size (bytes)	Description
8	MeasurementHashAlgo	4	<p>Bit mask indicating the SPDM-enumerated hashing algorithms used for measurements.</p> <ul style="list-style-type: none"> <li>• Byte 0 Bit 0. Raw Bit Stream Only.</li> <li>• Byte 0 Bit 1. <a href="#">TPM_ALG_SHA_256</a>.</li> <li>• Byte 0 Bit 2. <a href="#">TPM_ALG_SHA_384</a>.</li> <li>• Byte 0 Bit 3. <a href="#">TPM_ALG_SHA_512</a>.</li> <li>• Byte 0 Bit 4. <a href="#">TPM_ALG_SHA3_256</a>.</li> <li>• Byte 0 Bit 5. <a href="#">TPM_ALG_SHA3_384</a>.</li> <li>• Byte 0 Bit 6. <a href="#">TPM_ALG_SHA3_512</a>.</li> <li>• Byte 0 Bit 7. <a href="#">TPM_ALG_SM3_256</a>.</li> </ul> <p>• If the Responder supports <code>GET_MEASUREMENTS</code> and <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>.</p> <p>• All other values reserved.</p> <p>A Responder shall only select bit 0 if the Responder supports raw bit streams as the only form of measurement; otherwise, it shall select one of the other bits.</p>
12	BaseAsymSel	4	<p>Bit mask indicating the SPDM-enumerated asymmetric key signature algorithm selected for the purpose of signature generation. If the capabilities do not support this algorithm, this value is not used and shall be set to zero. The Responder shall set no more than one bit.</p>
16	BaseHashSel	4	<p>Bit mask indicating the SPDM-enumerated hashing algorithm selected. If the capabilities do not support this algorithm, this value is not used and shall be set to zero. The Responder shall set no more than one bit.</p>
20	Reserved	12	Reserved.
32	ExtAsymSelCount	1	<p>Number of extended asymmetric key signature algorithms selected for the purpose of signature generation. Shall be either <code>0</code> or <code>1</code> (=A'). If the capabilities do not support this algorithm, this value is not used and shall be set to zero.</p>
33	ExtHashSelCount	1	<p>The number of extended hashing algorithms selected. Shall be either <code>0</code> or <code>1</code> (=E'). If the capabilities do not support this algorithm, this value is not used and shall be set to zero.</p>
34	Reserved	2	Reserved.

Byte offset	Field	Size (bytes)	Description
36	ExtAsymSel	4 * A'	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 <a href="#">Extended algorithm field format</a> table describes the format of this field.
36 + 4 * A'	ExtHashSel	4 * E'	Extended hashing algorithm selected. The Responder shall use this hashing algorithm during all subsequent response messages to the Requester. The Requester shall use this hashing algorithm during all subsequent applicable request messages to the Responder. The <a href="#">Extended algorithm field format</a> table describes the format of this field.
36 + 4 * A' + 4 * E'	RespAlgStruct	AlgStructSize	See <a href="#">Table 20 — Response AlgStructure field format</a> .

268 AlgStructSize is the sum of the size of all Algorithm structure tables, as the following tables show. The algorithm structure table need be present only if the Responder supports that AlgType . AlgType shall monotonically increase for subsequent entries.

269 **Table 20 — Response AlgStructure field format**

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

Byte offset	Field	Size (bytes)	Description
2 + FixedAlgCount	AlgExternal	4 * ExtAlgCount'	If present: a Requester-supported, Responder-selected, extended algorithm. Responder shall select at most one external algorithm. <a href="#">Table 25 — Extended Algorithm field format</a> describes the format of this field.

270 The following tables describe each individual type and their associated fixed fields:

- [Table 21 — DHE structure](#)
- [Table 22 — AEAD structure](#)
- [Table 23 — ReqBaseAsymAlg structure](#)
- [Table 24 — KeySchedule structure](#)
- [Table 25 — Extended Algorithm field format](#)

271 **Table 21 — DHE structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	0x02 = DHE
1	AlgCount	1	<ul style="list-style-type: none"> <li>• Bit [7:4]. Shall be a value of 2.</li> <li>• Bit [3:0]. Number of Requester-supported, Responder-selected, extended DHE groups (= ExtAlgCount2'). This value shall be either 0 or 1.</li> </ul>
2	AlgSupported	2	<p>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"> <li>• Byte 0 Bit 0. <a href="#">ffdhe2048</a> (D = 256).</li> <li>• Byte 0 Bit 1. <a href="#">ffdhe3072</a> (D = 384).</li> <li>• Byte 0 Bit 2. <a href="#">ffdhe4096</a> (D = 512).</li> <li>• Byte 0 Bit 3. <a href="#">secp256r1</a> (D = 64, C = 32)</li> <li>• Byte 0 Bit 4. <a href="#">secp384r1</a> (D = 96, C = 48).</li> <li>• Byte 0 Bit 5. <a href="#">secp521r1</a> (D = 132, C = 66).</li> <li>• Byte 0 Bit 6. <a href="#">SM2_P256 (Part 3 and Part 5 of GB/T 32918)</a> (D = 64, C = 32).</li> <li>• All other values reserved.</li> </ul>
4	AlgExternal	4 * ExtAlgCount2'	If present: a Requester-supported, Responder-selected, extended DHE algorithm. <a href="#">Table 25 — Extended Algorithm field format</a> describes the format of this field.

272 **Table 22 — AEAD structure**



Byte offset	Field	Size (bytes)	Description
0	AlgType	1	0x03 = AEAD
1	AlgCount	1	<ul style="list-style-type: none"> <li>Bit [7:4]. Shall be a value of 2.</li> <li>Bit [3:0]. Number of Requester-supported, Responder-selected, extended AEAD algorithms (= ExtAlgCount3'). This value shall be either 0 or 1.</li> </ul>
2	AlgSupported	2	Bit mask for indicating a Requester-supported, Responder-selected, SPDM-enumerated AEAD algorithm. <ul style="list-style-type: none"> <li>Byte 0 Bit 0. <a href="#">AES-128-GCM</a>.</li> <li>Byte 0 Bit 1. <a href="#">AES-256-GCM</a>.</li> <li>Byte 0 Bit 2. <a href="#">CHACHA20_POLY1305</a>.</li> <li>Byte 0 Bit 3. <a href="#">AEAD_SM4_GCM</a>.</li> <li>All other values reserved.</li> </ul>
4	AlgExternal	4 * ExtAlgCount3'	If present: a Requester-supported, Responder-selected, extended AEAD algorithm. <a href="#">Table 25 — Extended Algorithm field format</a> describes the format of this field.

273 **Table 23 — ReqBaseAsymAlg structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	0x04 = ReqBaseAsymAlg
1	AlgCount	1	<ul style="list-style-type: none"> <li>Bit [7:4]. Shall be a value of 2.</li> <li>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.</li> </ul>

Byte offset	Field	Size (bytes)	Description
2	AlgSupported	2	<p>Bit mask for indicating a Requester-supported, Responder-selected, SPDM-enumerated asymmetric key signature algorithm for the purpose of signature verification.</p> <ul style="list-style-type: none"> <li>• Byte 0 Bit 0. <a href="#">TPM_ALG_RSASSA_2048</a>.</li> <li>• Byte 0 Bit 1. <a href="#">TPM_ALG_RSAPSS_2048</a>.</li> <li>• Byte 0 Bit 2. <a href="#">TPM_ALG_RSASSA_3072</a>.</li> <li>• Byte 0 Bit 3. <a href="#">TPM_ALG_RSAPSS_3072</a>.</li> <li>• Byte 0 Bit 4. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P256</a>.</li> <li>• Byte 0 Bit 5. <a href="#">TPM_ALG_RSASSA_4096</a>.</li> <li>• Byte 0 Bit 6. <a href="#">TPM_ALG_RSAPSS_4096</a>.</li> <li>• Byte 0 Bit 7. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P384</a>.</li> <li>• Byte 1 Bit 0. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P521</a>.</li> <li>• Byte 1 Bit 1. <a href="#">TPM_ALG_SM2_ECC_SM2_P256</a>.</li> <li>• Byte 1 Bit 2. <a href="#">EdDSA ed25519</a>.</li> <li>• Byte 1 Bit 3. <a href="#">EdDSA ed448</a>.</li> <li>• All other values reserved.</li> </ul>
4	AlgExternal	4* <code>ExtAlgCount4</code>	<p>If present: a Requester-supported, Responder-selected extended asymmetric key signature algorithm for the purpose of signature verification. <a href="#">Table 25 — Extended Algorithm field format</a> describes the format of this field.</p>

274

**Table 24 — KeySchedule structure**

Byte offset	Field	Size (bytes)	Description
0	AlgType	1	<code>0x05 = KeySchedule</code>
1	AlgCount	1	<ul style="list-style-type: none"> <li>• Bit [7:4]. Shall be a value of 2.</li> <li>• Bit [3:0]. Number of Requester-supported, Responder-selected, extended key schedule algorithms (= <code>ExtAlgCount5</code>). This value shall be either 0 or 1.</li> </ul>
2	AlgSupported	2	<p>Bit mask for indicating a Requester-supported, Responder-selected, SPDM-enumerated key schedule algorithm.</p> <ul style="list-style-type: none"> <li>• Byte 0 Bit 0. SPDM key schedule.</li> <li>• All other values are reserved.</li> </ul>
4	AlgExternal	4* <code>ExtAlgCount5</code>	<p>If present: a Requester-supported, Responder-selected, extended key schedule algorithm. <a href="#">Table 25 — Extended Algorithm field format</a> describes the format of this field.</p>

275 **Table 25 — Extended Algorithm field format**

276 Describes algorithms that are external to this specification.

Byte offset	Field	Size (bytes)	Description
0	Registry ID	1	Shall represent the registry or standards body. The <b>ID</b> column in <a href="#">Table 47 — Registry or standards body ID</a> 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 <a href="#">Table 47 — Registry or standards body ID</a> . At present, DMTF does not define any algorithms for use in extended algorithms fields.

277 **Opaque Data Format Support and Selection Table**

278 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 `OpaqueDataLen` field to a value of zero.

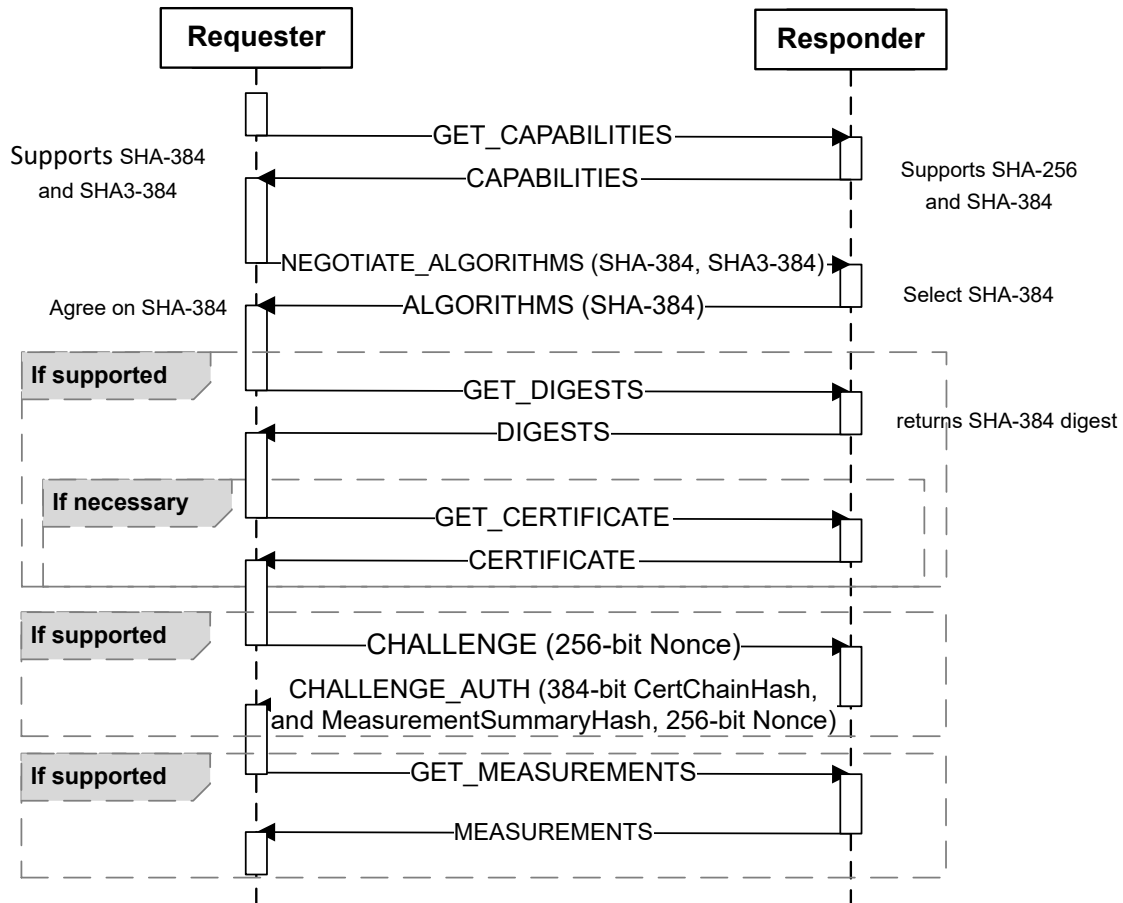
Bit offset	Field	Description
0	OpaqueDataFmt0	If set, this bit shall indicate that the format for all <code>OpaqueData</code> fields in this specification is defined by the device vendor or other standards body.
1	OpaqueDataFmt1	If set, this bit shall indicate that the format for all <code>OpaqueData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

279 For each algorithm type, a Responder shall not select both an SPDM-enumerated algorithm and an extended algorithm.

280 [Figure 7 — Hashing algorithm selection: Example 1](#) illustrates how two endpoints negotiate a base hashing algorithm. Endpoint A issues `NEGOTIATE_ALGORITHMS` request message and endpoint B selects an algorithm of which both endpoints are capable.

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

282



283 The SPDM protocol accounts for the possibility that both endpoints can 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 compared to the endpoint B Requester and endpoint A Responder communication pair.

284 **10.4.1 Connection Behavior after VCA**

285 With the successful completion of the `ALGORITHMS` message, all of 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 `ALGORITHMS`, 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 a `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 the Responder returns one or more digests of populated slots that perform ECDSA signatures or uses a different hash algorithm.

286 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.

## 287 **10.5 Responder identity authentication**

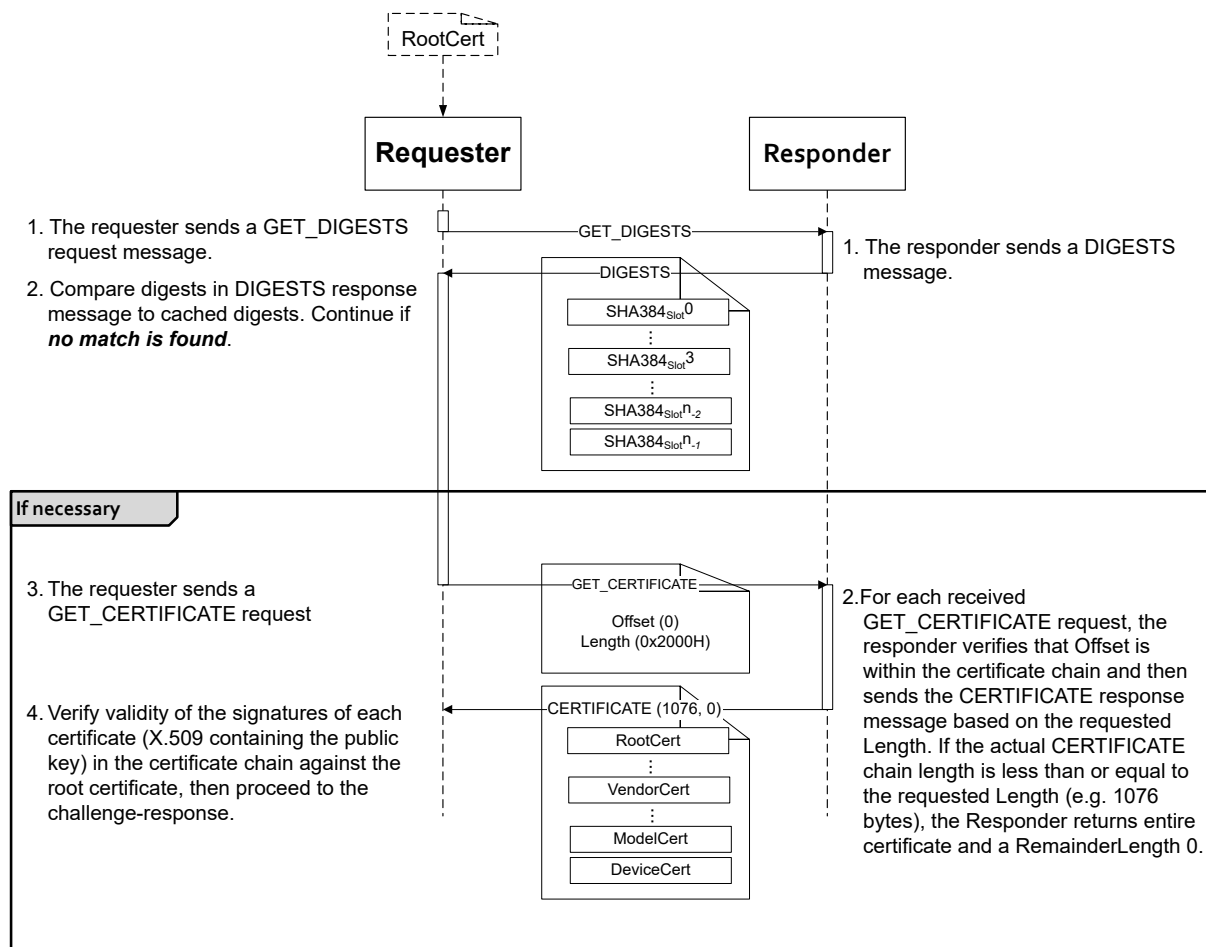
---

288 This clause describes request messages and response messages associated with the identity of the Responder authentication operations. The `GET_DIGESTS` and `GET_CERTIFICATE` messages shall be supported by a Responder that returns `CERT_CAP = 1` in the `CAPABILITIES` response message. The `CHALLENGE` message that this clause defines shall be supported by a Responder that returns `CHAL_CAP = 1` in the `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.

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

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

291



292 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 each of the certificate chains stored on an endpoint is returned with the DIGESTS response message, such that the Requester can cache the previously retrieved certificate chain hash values to detect any change to the certificate chains stored on the device before issuing the GET\_CERTIFICATE request message.

293 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 36 — Request ordering and message transcript computation rules for M1/M2](#).

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

295 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 with the latest GET\_VERSION request received.

## 296 10.6 Requester identity authentication

---

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

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

### 299 10.6.1 Certificates and certificate chains

300 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 26 — Certificate chain format](#) shows. In the context of this specification, a complete certificate chain is one where: (i) the first certificate is either 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*.

301 The SPDM endpoint shall contain a single public-private key pair per supported algorithm for its leaf certificate, regardless of how many certificate chains are stored on the device. The Responder selects a single asymmetric key signature algorithm per Requester.

302 Certificate chains are stored in logical locations called *slots*. Each slot shall either be empty or contain one complete certificate chain. A device shall not contain more than eight slots. Slots are numbered zero through seven inclusively. Slot 0 is populated by default. If a Responder device uses the `DeviceCert` model (`ALIAS_CERT_CAP=0b` in `CAPABILITIES` response), then the certificate chain in every populated slot shall use `DeviceCert` s. If a Responder device uses the `AliasCert` model (`ALIAS_CERT_CAP=1b` in `CAPABILITIES` response), then the certificate chain in every populated slot shall use `AliasCert` s. Additional slots may be populated through the supply chain such as by a platform integrator or by an end user such as the IT administrator. A slot mask identifies the certificate chains from the eight slots. Similarly, if the Requester supports mutual authentication, it 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 a Requester does not have capability flags to indicate the certificate model.

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

304 To allow for flexibility in supporting multiple certification models, the minimum number of certificates within a chain compliant with this specification shall be one, in which case the single certificate shall be the `DeviceCert` leaf certificate.

305 [Table 26 — Certificate chain format](#) describes the certificate chain format:

306 **Table 26 — Certificate chain format**

Byte offset	Field	Size (bytes)	Description
0	Length	2	Total length of the certificate chain, in bytes, including all fields in this table. This field is little endian.
2	Reserved	2	Reserved.
4	RootHash	H	Digest of the Root Certificate. Note that Root Certificate is ASN.1 DER-encoded for this digest. This field shall be in <a href="#">Hash byte order</a> . 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)	A <a href="#">complete</a> certificate chain, consisting of one or more ASN.1 DER-encoded <a href="#">X.509 v3</a> certificates. This field shall be in <a href="#">Encoded ASN.1 byte order</a> .

## 307 10.7 GET\_DIGESTS request and DIGESTS response messages

308 This request message shall retrieve the certificate chain digests.

309 [Table 27 — GET\\_DIGESTS request message format](#) shows the `GET_DIGESTS` request message format.

310 The digests in [Table 28 — Successful DIGESTS response message format](#) shall be computed over the certificate chain as [Table 26 — Certificate chain format](#) shows.

### 311 Table 27 — GET\_DIGESTS request message format

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

### 312 Table 28 — Successful DIGESTS response message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0x01</code> = <code>DIGESTS</code> . See <a href="#">Table 5 — SPDM response codes</a> .
2	Param1	1	Reserved.



Byte offset	Field	Size (bytes)	Description
3	Param2	1	Slot mask. The bit in position K of this byte shall be set to 1b if and only if slot number K contains a certificate chain for the protocol version in the SPDMVersion field. (Bit 0 is the least significant bit of the byte.) The number of digests returned shall be equal to the number of bits set in this byte. The digests shall be returned in order of increasing slot number.
4	Digest[0]	H	Digest of the first certificate chain. This field shall be in <a href="#">Hash byte order</a> .
...	...	...	...
4 + (H * (n - 1))	Digest[n-1]	H	Digest of the last (n <sup>th</sup> ) certificate chain. This field shall be in <a href="#">Hash byte order</a> .

313 **10.8 GET\_CERTIFICATE request and CERTIFICATE response messages**

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

315 [Table 29 — GET\\_CERTIFICATE request message format](#) shows the GET\_CERTIFICATE request message format.

316 [Table 30 — Successful CERTIFICATE response message format](#) shows the CERTIFICATE response message format.

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

318 **Table 29 — GET\_CERTIFICATE request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	0x82 = GET_CERTIFICATE. See <a href="#">Table 4 — SPDM request codes</a> .
2	Param1	1	Bit [7:4]. Reserved. Bit [3:0]. SlotID. Slot number of the Responder certificate chain to read. The value in this field shall be between 0 and 7 inclusive.
3	Param2	1	Reserved.

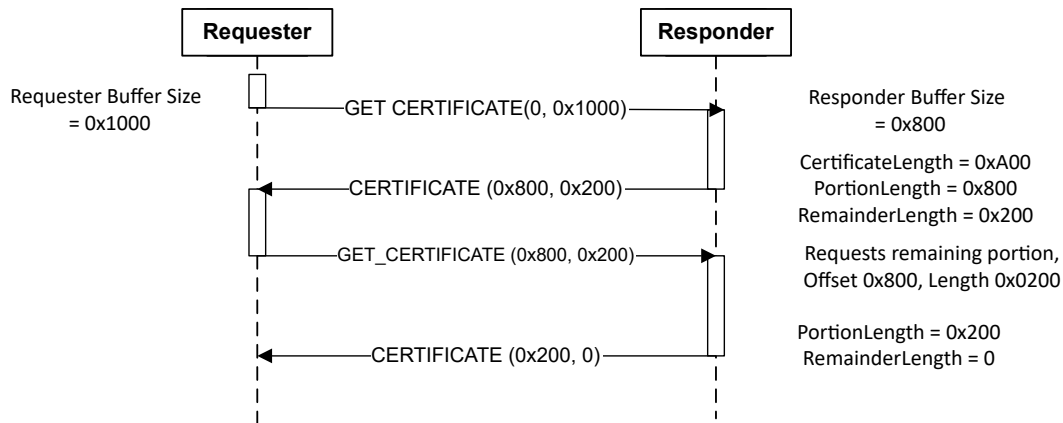
Byte offset	Field	Size (bytes)	Description
4	Offset	2	Offset in bytes from the start of the certificate chain to where the read request message begins. The Responder shall send its certificate chain starting from this offset. For the first <code>GET_CERTIFICATE</code> request for a given slot, the Requester 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	Length of certificate chain data, in bytes, to be returned in the corresponding response. This field is an unsigned 16-bit integer.

319 **Table 30 — Successful CERTIFICATE response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0x02</code> = CERTIFICATE . See <a href="#">Table 5 — SPDM response codes</a> .
2	Param1	1	Bit [7:4]. Reserved. Bit [3:0]. <code>SlotID</code> . Slot number of the certificate chain returned.
3	Param2	1	Reserved.
4	PortionLength	2	Number of bytes of this portion of certificate chain. This should be less than or equal to <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.
6	RemainderLength	2	Number of bytes of the certificate chain that have not been sent yet, after the current response. For the last response, this field shall be 0 as an indication to the Requester that the entire certificate chain has been sent.
8	CertChain	<code>PortionLength</code>	Requested contents of target certificate chain, as described in <a href="#">Certificates and certificate chains</a> .

320 [Figure 9 — Responder cannot return full length data flow](#) shows the high-level request-response message flow for Responder response when it cannot return the entire data requested by the Requester in the first response.321 **Figure 9 — Responder cannot return full length data flow**

322



323 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 the `DataTransferSize` of the Requester or the transmit buffer of the Responder. Specifically:

- 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, `RemainderLength` to 0 and store the contents of the complete certificate chain in `CertChain` in the CERTIFICATE response. The Responder shall then fragment and serve this response message in chunks, as per the clauses presented in [CHUNK\\_GET request and CHUNK\\_RESPONSE response message](#). In this case, the Responder shall not return a partial certificate chain.

324 **10.8.1 Mutual authentication requirements for GET\_CERTIFICATE and CERTIFICATE messages**

325 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 [GET\\_CERTIFICATE request and CERTIFICATE response messages](#) clause shall also apply to the Responder. These two statements essentially describe a role reversal.

326 **10.8.2 SPDM certificate requirements and recommendations**

327 This specification defines a number of X.509 v3 [required](#) and [optional](#) fields for compliant SPDM certificates. 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](#).

328 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](#).

329 **Table 31 — Required fields**

Field	Description
Version	Version of the encoded certificate shall be present and shall be 3 (encoded as value 2).
Serial Number	CA-assigned serial number shall be present with a positive integer value.
Signature Algorithm	Signature algorithm that CA uses shall be present.
Issuer	CA distinguished name shall be specified.
Subject Name	Subject name shall be present and shall represent the distinguished name associated with the leaf certificate.
Validity	See <a href="#">Certificate validity details</a> , and <a href="#">RFC5280</a> .
Subject Public Key Info	Device public key and the algorithm shall be present.
Key Usage	Shall be present and key usage bit for digital signature shall be set.

330 **Table 32 — Optional fields**

Field	Description
Basic Constraints	If present, the CA value shall be FALSE in the leaf certificate.
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 <a href="#">Definition of otherName using the DMTF OID</a> . 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 <a href="#">Extended Key Usage authentication OIDs</a> .
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 <a href="#">SPDM Non-Critical Certificate Extension OID</a> .

331 **Certificate validity details**

332 As per [RFC5280](#), 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 ).

333 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 . In general, 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.

334 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 cause certificate expiration and necessitate certificate renewal, and thus device re-certification, during the lifetime of the device.

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

336 **Definition of otherName using the DMTF OID**

```

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

id-DMTF-spdm 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})

```

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

**338 10.8.2.1 Extended Key Usage authentication OIDs**

339 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.

340 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.

341 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.

### 342 10.8.2.2 SPDM Non-Critical Certificate Extension OID

343 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 [RFC5280](#) and its encoding shall follow the `Extension` syntax also defined in [RFC5280](#). 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.

#### 344 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
}

```

345 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`.

346 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`.

#### 347 10.8.2.2.1 Hardware identity OID

348 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.

#### 349 Hardware identity OID format

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

**350 10.8.2.2.2 Mutable certificate OID**

351 Mutable certificates may include the `id-DMTF-mutable-certificate` OID to identify them as mutable. If used, this  
 352 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.

**352 Mutable certificate OID format**

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

**353 10.9 CHALLENGE request and CHALLENGE\_AUTH response messages**

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

355 [Table 33 — CHALLENGE request message format](#) shows the `CHALLENGE` request message format.

356 [Table 34 — Successful CHALLENGE\\_AUTH response message format](#) shows the `CHALLENGE_AUTH` response message  
 format.

357 [Table 35 — Table 37 — CHALLENGE\\_AUTH response attribute](#) shows the `CHALLENGE_AUTH` response attribute.

**358 Table 33 — CHALLENGE request message format**

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

Byte offset	Field	Size (bytes)	Description
3	Param2	1	Type of measurement summary hash requested: <ul style="list-style-type: none"> <li>• 0x0 . No measurement summary hash requested.</li> <li>• 0x1 . <i>TCB</i> measurements only.</li> <li>• 0xFF . All measurements.</li> <li>• All other values reserved.</li> </ul> <p>If a Responder does not support measurements ( MEAS_CAP=00b in CAPABILITIES response), the Requester shall set this value to 0x0 .</p>
4	Nonce	32	The Requester should choose a random value.

359 **Table 34 — Successful CHALLENGE\_AUTH response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	0x03 = CHALLENGE_AUTH . See <a href="#">Table 5 — SPDM response codes</a> .
2	Param1	1	Response Attribute Field. See <a href="#">Table 35 — CHALLENGE_AUTH response attribute</a> .
3	Param2	1	Slot mask. The bit in position K of this byte shall be set to 1b if and only if slot number K contains a certificate chain for the protocol version in the SPDMVersion field. Bit 0 is the least significant bit of the byte. This field is reserved if the public key of the Responder was provisioned to the Requester in a trusted environment.
4	CertChainHash	H	Hash of the certificate chain, as <a href="#">Table 26 — Certificate chain format</a> describes or public key (if the public key of the Responder was provisioned to the Requester in a trusted environment) used for authentication. The Requester can use this value to check that the certificate chain or public key matches the one requested.  This field shall be in <a href="#">Hash byte order</a> .
4 + H	Nonce	32	Responder-selected random value.



Byte offset	Field	Size (bytes)	Description
36 + H	MeasurementSummaryHash	H	<p>If the Responder does not support measurements ( <code>MEAS_CAP=00b</code> in <code>CAPABILITIES</code> response) or 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(Concatenation(MeasurementBlock[0], MeasurementBlock[1], ...))</code> where <code>MeasurementBlock[x]</code> denotes a measurement of an element in the TCB. Measurements are concatenated in ascending order based on their measurement index as <a href="#">Table 42 — Measurement block format</a> describes.</p> <p>When the requested <code>Param2 = 0x1</code> and 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>Param2 = 0xFF</code>, this field shall be computed as <code>hash(Concatenation(MeasurementBlock[0], MeasurementBlock[1], ..., MeasurementBlock[n]))</code> of all supported measurement blocks available in the measurement index range from <code>0x01</code> to <code>0xFE</code>, concatenated in ascending index order. Indices with no associated measurements shall not be included in the hash calculation. See the <a href="#">Measurement index assignments</a> clause.</p> <p>If the Responder supports both raw bit stream and digest representations for a given measurement index, then the Responder shall use the digest form.</p> <p>This field shall be in <a href="#">Hash byte order</a>.</p>
36 + 2H	OpaqueDataLength	2	<p>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.</p>
38 + 2H	OpaqueData	<code>OpaqueDataLength</code>	<p>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>.</p>

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

360 **Table 35 — 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	<b>DEPRECATED:</b> BasicMutAuthReq	<b>DEPRECATED:</b> When mutual authentication is supported by both Responder and Requester, the Responder shall set this bit to indicate 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 <a href="#">Basic mutual authentication flow</a> . If mutual authentication is not supported, this bit shall be zero.

361 **10.9.1 CHALLENGE\_AUTH signature generation**

362 To complete the CHALLENGE\_AUTH signature generation process, the Responder shall complete these steps:

- 363 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.

364 where:

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

- 366 ◦ If a response contains ErrorCode=ResponseNotReady :

367 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.

- 368 ◦ If a response contains an ErrorCode other than ResponseNotReady :

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

370 2. The Responder shall generate:

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

371 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.

## 372 10.9.2 CHALLENGE\_AUTH signature verification

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

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

375 1. The Requester shall perform:

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

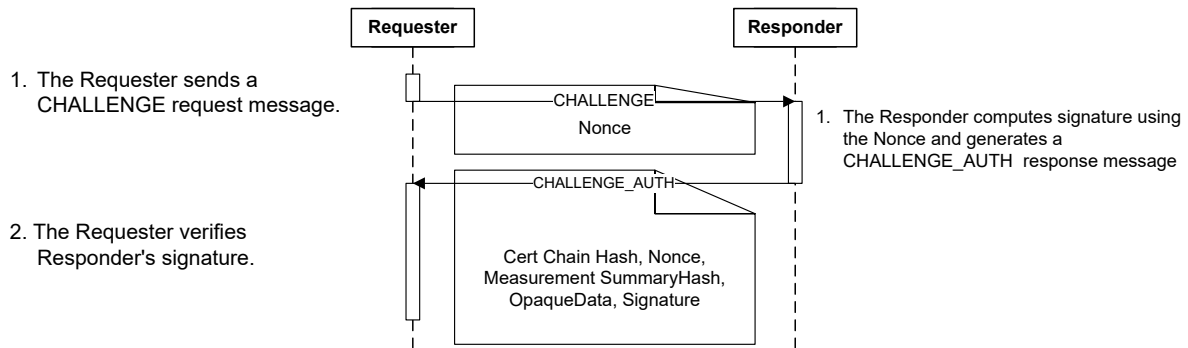
376 where:

- `SPDMsignatureVerify` is described in [Signature verification](#). When `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.

377 [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.

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

379



**380 10.9.2.1 Request ordering and message transcript computation rules for M1 and M2**

381 This clause applies to Responder-only authentication.

382 [Table 36 — 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.

383 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)

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

385 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.

386 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.

387 For additional rules, see [General ordering rules](#).

**388 Table 36 — Request ordering and message transcript computation rules for M1/M2**

Requests	Implementation conditions	M1/M2=Concatenate (A, B, C)
Initial value	N/A	M1/M2=null

Requests	Implementation conditions	M1/M2=Concatenate (A, B, C)
GET_VERSION issued	Requester issues this request to allow the Requester and Responder to determine an agreed upon Negotiated State . Also issued if the Requester detects an out of sync condition, 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	Requester skipped issuing these requests after a Reset or a completed CHALLENGE_AUTH response, that caused M1/M2 to re-initialize to null , if the Responder has 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	Requester issued these requests in this order after NEGOTIATE_ALGORITHMS request completion, or after a Reset or a completed CHALLENGE_AUTH response, that caused M1/M2 to re-initialize to null , if it chose to skip the previous three requests.	B1=Concatenate(GET_DIGESTS, DIGESTS, GET_CERTIFICATE, CERTIFICATE)
GET_DIGESTS , GET_CERTIFICATE skipped	Requester skipped both requests after a Reset or a completed CHALLENGE_AUTH response, that caused M1/M2 to re-initialize to null , because it could use previously cached certificate information.	B2=null
GET_DIGESTS issued, GET_CERTIFICATE skipped	Requester skipped GET_CERTIFICATE request after a Reset or a completed CHALLENGE_AUTH response, that caused M1/M2 to re-initialize to null because it could use the previously cached CERTIFICATE response.	B3=Concatenate(GET_DIGESTS, DIGESTS)
GET_DIGESTS skipped, GET_CERTIFICATE issued	Requester skipped GET_DIGESTS request after a Reset or a completed CHALLENGE_AUTH response, that caused M1/M2 to re-initialize to null . The Requester uses the previously cached CERTIFICATE response for 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 <a href="#">Table 33 — CHALLENGE request message format</a> .
CHALLENGE completion	Completion of CHALLENGE sets M1/M2 to null .	M1/M2=null

Requests	Implementation conditions	M1/M2=Concatenate (A, B, C)
Other issued	If the Requester issued GET_MEASUREMENTS or KEY_EXCHANGE or FINISH or PSK_EXCHANGE or PSK_FINISH or KEY_UPDATE or HEARTBEAT or GET_ENCAPSULATED_REQUEST or DELIVER_ENCAPSULATED_RESPONSE or END_SESSION request(s) and skipped CHALLENGE completion, M1/M2 are set to null .	M1/M2=null

389 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.

390 DEPRECATED

### 391 10.9.3 Basic mutual authentication

392 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. These two statements essentially describe a role reversal, unless otherwise stated.

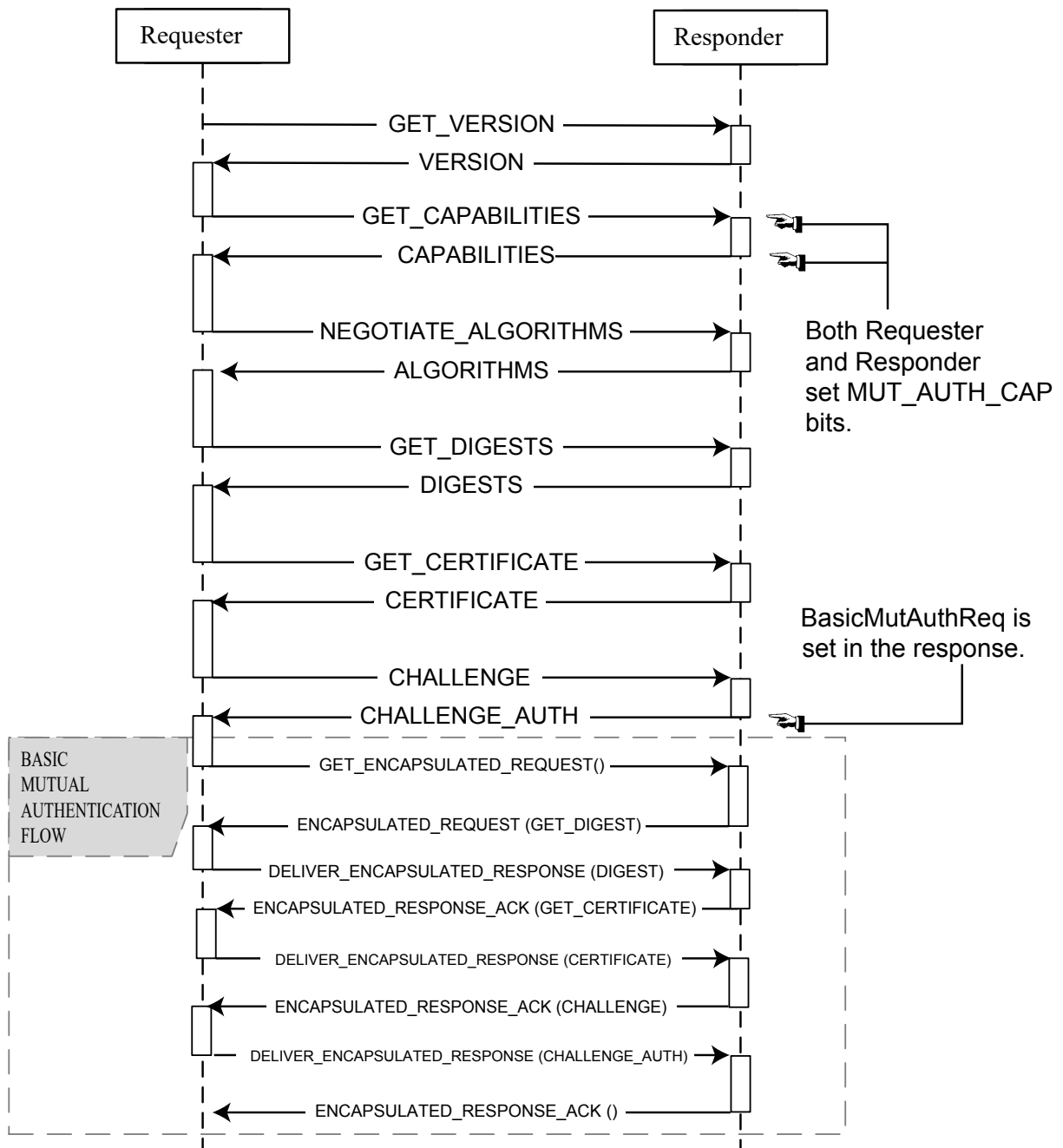
393 The basic mutual authentication flow shall start when the Requester successfully receives a [CHALLENGE\\_AUTH](#) with **BasicMutAuthReq** set. This flow shall utilize message encapsulation as described in [GET\\_ENCAPSULATED\\_REQUEST request and ENCAPSULATED\\_REQUEST response messages](#) to retrieve request messages. The basic mutual authentication flow shall end when the encapsulated request flow ends.

394 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](#) response with [ErrorCode=UnexpectedRequest](#) and shall terminate the flow.

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

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

397



**398 10.9.3.1 Mutual authentication message transcript**

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

400 [Table 37 — 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.

401 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*)

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

403 **Table 37 — 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)

404 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.

405 DEPRECATED

## 406 10.10 Firmware and other measurements

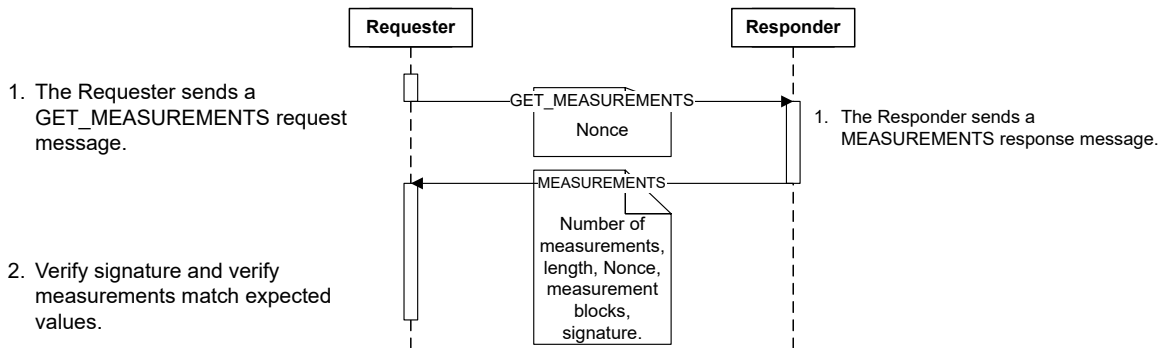
407 This clause describes request messages and response messages associated with endpoint measurement. All request messages in this clause shall be supported by an endpoint that returns `MEAS_CAP=01b` or `MEAS_CAP=10b` in `CAPABILITIES` response.

408 [Figure 12 — Measurement retrieval flow](#) shows the high-level request-response flow and sequence for endpoint measurement. If `MEAS_FRESH_CAP` bit in the `CAPABILITIES` response message returns 0, and the Requester requires fresh measurements, the Responder shall be Reset before `GET_MEASUREMENTS` is resent. The mechanisms employed for Resetting the Responder are outside the scope of this specification.

409 **Figure 12 — Measurement retrieval flow**



410



411 **10.11 GET\_MEASUREMENTS request and MEASUREMENTS response messages**

412 Measurements in SPDM are represented in the form of measurement *blocks*. [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 more than one measurement blocks is called a *measurement record*.

413 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.

414 [Table 38 — GET\\_MEASUREMENTS request message format](#) shows the `GET_MEASUREMENTS` request message format.

415 [Table 39 — GET\\_MEASUREMENTS request attributes](#) shows the `GET_MEASUREMENTS` request message attributes.

416 [Table 41 — Successful MEASUREMENTS response message format](#) shows the `MEASUREMENTS` response message format. The measurement blocks in `MeasurementRecord` shall be sorted in ascending order by index.

417 **Table 38 — GET\_MEASUREMENTS request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0xE0</code> = <code>GET_MEASUREMENTS</code> . See <a href="#">Table 4 — SPDM request codes</a> .
2	Param1	1	Request attributes. See <a href="#">Table 39 — GET_MEASUREMENTS request attributes</a> .

Byte offset	Field	Size (bytes)	Description
3	Param2	1	<p>Measurement operation.</p> <ul style="list-style-type: none"> <li>A value of 0x0 shall query the Responder for the total number of measurement blocks available.</li> <li>A value of 0xFF shall request all measurement blocks.</li> <li>A value between 0x1 and 0xFE, inclusively, shall request the measurement block at the index corresponding to that value.</li> </ul>
4	Nonce	NL=32 or NL=0	The Requester should choose a random value. This field is only present if Bit [0] of Param1 is 1. See <a href="#">Table 39 — GET_MEASUREMENTS request attributes</a> .
4 + NL	SlotIDParam	1	<p>This field is only present if Bit [0] of Param1 is 1.</p> <ul style="list-style-type: none"> <li>Bit [7:4]. Reserved.</li> <li>Bit [3:0]. SlotID. 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 0xF. See <a href="#">Table 39 — GET_MEASUREMENTS request attributes</a>.</li> </ul>

418 **Table 39 — GET\_MEASUREMENTS request attributes**

Bit offset	Field	Description
0	SignatureRequested	<p>If the Responder can generate a signature ( MEAS_CAP is 10b in the <a href="#">CAPABILITIES</a> response and either BaseAsymSel or ExtAsymSelCount is non-zero) a value of 1 indicates that a signature on the measurement log (defined in <a href="#">MEASUREMENTS signature generation</a>) is required. The Nonce 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 0 indicates that the Requester does not require a signature. The Responder shall not generate a signature in the response. The Nonce field shall be absent in the request.</p> <p>For Responders that cannot generate a signature ( MEAS_CAP is 01b in the <a href="#">CAPABILITIES</a> response or both BaseAsymSel and ExtAsymSelCount are zero) the Requester shall always use 0.</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 <a href="#">Table 42 — Measurement block format</a> 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 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.</p> <p>Value <code>0</code> shall request the Responder to return a hash version of the measurement. If the Responder cannot return hash of the measurement (for example, if the measurement represents a data structure where digest is not applicable), it shall return the corresponding raw bit stream.</p>
[7:2]	Reserved	Reserved.

419 **Measurement index assignments**

420 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 by a particular Responder and the indices to which they correspond. When measurements follow the DMTF measurement specification format that [Table 43 — DMTF measurement specification format](#) describes, a measurement with a `DMTFSpecMeasurementValueType[6:0]` equal to `0x04` is the measurement manifest. If a Requester specifies a measurement index that a Responder does not support then the Responder shall respond with an `ERROR` message of `ErrorCode=InvalidRequest`.

421 To aid interoperability, this specification reserves indices `0xF0` to `0xFE` inclusive for specific purposes. If a Responder supports a type of measurement that [Table 40 — Measurement index assigned range](#) defines, it shall always assign it to the corresponding index value. A Responder shall not assign indices `0xF0` to `0xFE` to measurements of types other than those that [Table 41 — Successful MEASUREMENTS response message format](#) defines.

422 **Table 40 — Measurement index assigned range**

Measurement Index	Measurement type	Description
<code>0xF0</code> – <code>0xFC</code>	Reserved	Reserved.
<code>0xFD</code>	Measurement manifest	Metadata on available measurements, as type <code>DMTFSpecMeasurementValueType[6:0] = 0x04</code> defines.

Measurement Index	Measurement type	Description
0xFE	Device mode	Structured device mode information, as type <code>DMTFSpecMeasurementValueType[6:0] = 0x05</code> defines.

423

**Table 41 — Successful MEASUREMENTS response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0x60</code> = MEASUREMENTS . See <a href="#">Table 5 — SPDM response codes</a> .
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 indicates if one or more <code>MeasurementRecord</code> fields of previous MEASUREMENTS 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 MEASUREMENTS 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 MEASUREMENTS 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 MEASUREMENTS responses in the measurement log being signed.</p> <p><code>11b</code> : reserved.</p> <p>Bit [3:0]. <code>SlotID</code> . If this message contains a signature, this field contains 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	Number of <code>measurement blocks</code> in the full <code>MeasurementRecord</code> . If <code>Param2</code> in the requested measurement operation is <code>0</code> , this field shall be <code>0</code> .
5	MeasurementRecordLength	3	Size of the full <code>MeasurementRecord</code> in bytes. If <code>Param2</code> in the requested measurement operation is <code>0</code> , this field shall be <code>0</code> .
8	MeasurementRecordData	$L = \text{MeasurementRecordLength}$	Concatenation of all measurement blocks that correspond to the requested Measurement operation. <code>Measurement block</code> 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	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.
$42 + L$	OpaqueData	<code>OpaqueDataLength</code>	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> .
$42 + L + \text{OpaqueDataLength}$	Signature	<code>SigLen</code>	<code>Signature</code> of the measurement log, 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 size of that asymmetric signing algorithm output. This field is conditional and only present in the <code>MEASUREMENTS</code> response corresponding to a <code>GET_MEASUREMENTS</code> request with <code>Param1[0]</code> set to 1.

**424 10.11.1 Measurement block**

425 Each measurement block that the `MEASUREMENTS` response message defines shall contain a four-byte descriptor, offsets 0 through 3, followed by the measurement data that correspond to a particular measurement index and measurement type. The blocks are ordered by `Index` .

426 [Table 42 — Measurement block format](#) shows the format for a measurement block:

427 **Table 42 — Measurement block format**

Byte offset	Field	Size (bytes)	Description
0	Index	1	Index. When <code>Param2</code> of <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 in the <code>ALGORITHMS</code> message. See <a href="#">Table 19 — Successful ALGORITHMS response message format</a> . Only one bit shall be set.  Bit 0: DMTF. If this bit is set, the format of data in the <code>Measurement</code> field is as specified in the DMTF measurement specification format that <a href="#">Table 43 — DMTF measurement specification format</a> describes.  All other bits are reserved.
2	MeasurementSize	2	Size of <code>Measurement</code> , in bytes.
4	Measurement	<code>MeasurementSize</code>	The <code>MeasurementSpecification</code> defines the format of this field.

#### 428 10.11.1.1 DMTF specification for the Measurement field of a measurement block

429 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 43 — DMTF measurement specification format](#) specifies this format.

430 The measurement manifest of `DMTFSpecMeasurementValueType` refers to a manifest that describes contents of other indexes. For example, the set of firmware modules running on the Responder can change at runtime. The measurement manifest tells the Requester which firmware modules' measurements are reported in this response and their indexes. The format of the measurement manifest is out of scope of this specification.

#### 431 Table 43 — DMTF measurement specification format

Byte offset	Field	Size (bytes)	Description
0	DMTFSpecMeasurementValueType	1	<p>Composed of:</p> <ul style="list-style-type: none"> <li>• Bit [7]. Indicates the representation in <code>DMTFSpecMeasurementValue</code>.</li> <li>• Bit [6:0]. Indicates what is being measured by <code>DMTFSpecMeasurementValue</code>.</li> </ul> <p>These values are set independently and are interpreted as follows:</p> <ul style="list-style-type: none"> <li>• [7]=0b . Digest.</li> <li>• [7]=1b . Raw bit stream. The Responder should ensure the raw bit stream does not contain secrets.</li> <li>• [6:0]=00h . Immutable ROM.</li> <li>• [6:0]=01h . Mutable firmware.</li> <li>• [6:0]=02h . Hardware configuration, such as straps.</li> <li>• [6:0]=03h . Firmware configuration, such as configurable firmware policy.</li> <li>• [6:0]=04h . Measurement manifest. When <code>DMTFSpecMeasurementValueType[6:0]=04h</code>, the Responder should support setting <code>DMTFSpecMeasurementValueType[7]</code> to either 0b or 1b.</li> <li>• [6:0]=05h . Structured representation of debug and device mode. See <a href="#">Device mode field of a measurement block</a>. When <code>DMTFSpecMeasurementValueType[6:0]=05h</code>, <code>DMTFSpecMeasurementValueType[7]</code> shall be set to 1b.</li> <li>• [6:0]=06h . Mutable firmware's version number. This specification does not mandate a format for firmware version number. When <code>DMTFSpecMeasurementValueType[6:0]=06h</code>, <code>DMTFSpecMeasurementValueType[7]</code> should be set to 1b.</li> <li>• [6:0]=07h . Mutable firmware's security version number, which should be formatted as an 8-byte unsigned integer. When <code>DMTFSpecMeasurementValueType[6:0]=07h</code>, <code>DMTFSpecMeasurementValueType[7]</code> should be set to 1b.</li> <li>• All other values reserved.</li> </ul>

Byte offset	Field	Size (bytes)	Description
1	DMTFSpecMeasurementValueSize	2	Size of <code>DMTFSpecMeasurementValue</code> , in bytes.  When <code>DMTFSpecMeasurementValueType[7]=0b</code> , the <code>DMTFSpecMeasurementValueSize</code> shall be derived from the measurement hash algorithm that the <code>ALGORITHM</code> response message returns.
3	DMTFSpecMeasurementValue	MS	Cryptographic hash or raw bit stream, as indicated in <code>DMTFSpecMeasurementValueType[7]</code> . For cryptographic hashes or digests, this field shall be in <a href="#">Hash byte order</a> . The vendor defines the byte order for raw bit streams.

### 432 10.11.1.2 Device mode field of a measurement block

Byte offset	Field	Size (bytes)	Description
0	OperationalModeCapabilities	4	Fields with bits set to 1 indicate support for reporting the associated state in <code>OperationalModeState</code> . <ul style="list-style-type: none"> <li>Bit [0]. Indicates support for reporting device in manufacturing mode.</li> <li>Bit [1]. Indicates support for reporting device in validation mode.</li> <li>Bit [2]. Indicates support for reporting device in normal operational mode.</li> <li>Bit [3]. Indicates support for reporting device in recovery mode.</li> <li>Bit [4]. Indicates support for reporting device in Return Merchandise Authorization (RMA) mode.</li> <li>Bit [5]. Indicates support for reporting device in decommissioned mode.</li> </ul> All other values reserved.
4	OperationalModeState	4	Fields with bits set to 1 indicate true for the reported state. <ul style="list-style-type: none"> <li>Bit [0]. Indicates the device is in manufacturing mode.</li> <li>Bit [1]. Indicates the device is in validation mode.</li> <li>Bit [2]. Indicates the device is in normal operational mode.</li> <li>Bit [3]. Indicates the device is in recovery mode.</li> <li>Bit [4]. Indicates the device is in RMA mode.</li> <li>Bit [5]. Indicates the device is in decommissioned mode.</li> <li>All other values reserved.</li> </ul>



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

**433 10.11.2 MEASUREMENTS signature generation**

434 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:

- 435 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)
```

436 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`.

437 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.

438 An error response with `ErrorCode=ResponseNotReady` or `ErrorCode=LargeResponse` 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` or `LargeResponse` shall re-initialize L1/L2 to null.

439 2. The Responder shall generate:

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

440 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.

### 441 10.11.3 MEASUREMENTS signature verification

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

443 1. The Requester shall perform:

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

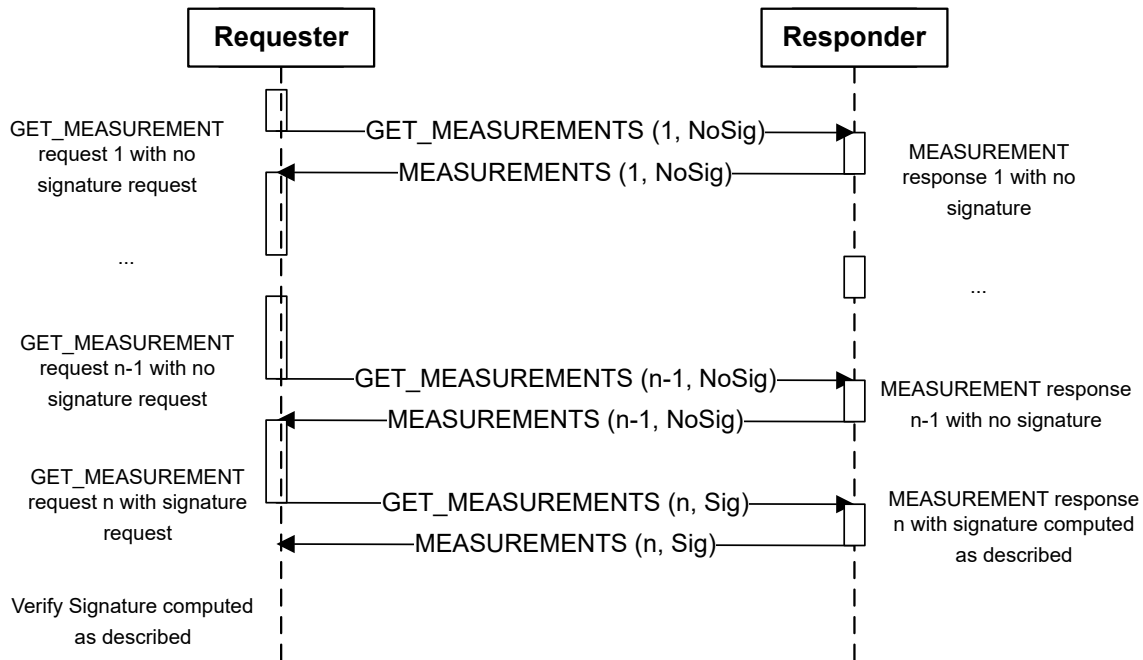
444 where:

- 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.

445 [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, followed by a single GET\_MEASUREMENTS request  $n$  with a signature.

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

447



448 **10.12 ERROR response message**

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

450 [Table 44 — ERROR response message format](#) shows the `ERROR` response format.

451 [Table 45 — Error code and error data](#) shows the detailed error code, error data, and extended error data.

452 [Table 46 — ResponseNotReady extended error data](#) shows the `ResponseNotReady` extended error data.

453 [Table 47 — Registry or standards body ID](#) shows the registry or standards body ID.

454 [Table 48 — ExtendedErrorData format for vendor or other standards-defined ERROR response message](#) shows the `ExtendedErrorData` format definition for vendor or other standards-defined `ERROR` response message.

455 **Table 44 — ERROR response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as <a href="#">SPDM version</a> describes.
1	RequestResponseCode	1	<code>0x7F</code> = <code>ERROR</code> . See <a href="#">Table 5 — SPDM response codes</a> .

Byte offset	Field	Size (bytes)	Description
2	Param1	1	Error code. See <a href="#">Table 45 — Error code and error data</a> .
3	Param2	1	Error data. See <a href="#">Table 45 — Error code and error data</a> .
4	ExtendedErrorData	0-32	Optional extended data. See <a href="#">Table 45 — Error code and error data</a> .

456 **Table 45 — Error code and error data**

Error code	Value	Description	Error data	ExtendedErrorData
Reserved	0x00	Reserved.	Reserved	Reserved
InvalidRequest	0x01	One or more request fields are invalid	0x00	No extended error data is provided.
Reserved	0x02	Reserved.	Reserved	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 <a href="#">RequestResponseCode</a> in the request message is unsupported.	<a href="#">RequestResponseCode</a> in the request message.	No extended error data is provided

Error code	Value	Description	Error data	ExtendedErrorData
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.
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 <code>ErrorCode</code> can be sent in response to the <code>GET_DIGESTS</code> , <code>GET_CERTIFICATE</code> , <code>GET_CSR</code> or <code>SET_CERTIFICATE</code> message.	0x00	No extended error data is provided.

Error code	Value	Description	Error data	ExtendedErrorData
ResponseTooLarge	0x0D	<p>Used in the following scenarios.</p> <ul style="list-style-type: none"> <li>The response is greater than the <code>MaxSPDMsgSize</code> of the requesting SPDM endpoint.</li> <li>The <code>CHUNK_CAP</code> of the requesting endpoint is <code>0</code> and the response is larger than the size of the transmit buffer of the responding SPDM endpoint</li> <li>The <code>CHUNK_CAP</code> of the requesting endpoint is <code>1</code>, the <code>CHUNK_CAP</code> of the responding endpoint is <code>0</code>, and the response is larger than the <code>DataTransferSize</code> of the requesting endpoint.</li> </ul>	Reserved	See <a href="#">Table 49</a> — <a href="#">ExtendedErrorData format for ResponseTooLarge</a> .
RequestTooLarge	0x0E	The request is greater than the <code>MaxSPDMsgSize</code> of the receiving SPDM endpoint.	Reserved	Reserved
LargeResponse	0x0F	The response is greater than <code>DataTransferSize</code> and less than or equal to <code>MaxSPDMsgSize</code> of the requesting SPDM endpoint, or greater than the transmit buffer size of the responding SPDM endpoint.	Reserved	See <a href="#">Table 50</a> — <a href="#">ExtendedErrorData format for LargeResponse</a> .

Error code	Value	Description	Error data	ExtendedErrorData
MessageLost	0x10	The SPDM message is lost. For example, this error code can be used to indicate a Large Request, Large Response or the request in a <code>ResponseNotReady</code> has been lost.	Reserved	Reserved
Reserved	0x11 - 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 <a href="#">RESPOND_IF_READY request message format</a> .	0x00	See <a href="#">Table 46 — ResponseNotReady extended error data</a> .
RequestResynch	0x43	Responder is requesting Requester to reissue <code>GET_VERSION</code> to re-synchronize. An example is following a firmware update.	0x00	No extended error data is provided.
Reserved	0x44 - 0xFE	Reserved.	Reserved	Reserved
Vendor or Standard Defined	0xFF	Vendor or standard defined	Shall indicate the registry or standard body using one of the values in the <b>ID</b> column in <a href="#">Table 47 — Registry or standards body ID</a> .	See <a href="#">Table 48 — ExtendedErrorData format for vendor or other standards-defined ERROR response message</a> for format definition.

457 **Table 46 — ResponseNotReady extended error data**



Byte offset	Field	Size (bytes)	Description
0	RDTEExponent	1	<p>Exponent expressed in logarithmic (base 2 scale) to calculate <code>RDT</code> time in <math>\mu\text{s}</math> 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 <math>2^8=256 \mu\text{s}</math>.</p> <p>Requester should use <code>RDT</code> to avoid continuous pinging and issue the <code>RESPOND_IF_READY</code> request message, as <a href="#">Table 52 — RESPOND_IF_READY request message format</a> shows, after <code>RDT</code> time.</p> <p>For timing requirement details, see <a href="#">Table 7 — Timing specification for SPDM messages</a>.</p>
1	RequestCode	1	The request code that triggered this response.
2	Token	1	The opaque handle that the Requester shall pass in with the <code>RESPOND_IF_READY</code> request message, as <a href="#">Table 52 — RESPOND_IF_READY request message format</a> shows. The Responder can use the value in this field to provide the correct response when the Requester issues a <code>RESPOND_IF_READY</code> request.
3	RDTM	1	<p>Multiplier used to compute <code>WT<sub>Max</sub></code> in <math>\mu\text{s}</math> 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 <a href="#">Table 7 — Timing specification for SPDM messages</a>.</p>

458 **Table 47 — Registry or standards body ID**

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

ID	Vendor ID length (bytes)	Registry or standards body name	Description
0x0	0	<a href="#">DMTF</a>	DMTF does not have a Vendor ID registry.
0x1	2	<a href="#">TCG</a>	<code>VendorID</code> is identified by using <a href="#">TCG Vendor ID Registry</a> . For extended algorithms, see <a href="#">TCG Algorithm Registry</a> .

ID	Vendor ID length (bytes)	Registry or standards body name	Description
0x2	2	USB	VendorID is identified by using the vendor ID assigned by USB.
0x3	2	PCI-SIG	VendorID is identified using PCI-SIG Vendor ID.
0x4	4	IANA	The Private Enterprise Number (PEN) assigned by the Internet Assigned Numbers Authority (IANA) identifies the vendor.
0x5	4	HDBaseT	VendorID is identified by using HDBaseT HDCD entity.
0x6	2	MIPI	The Manufacturer ID assigned by MIPI identifies the vendor.
0x7	2	CXL	VendorID is identified by using CXL vendor ID.
0x8	2	JEDEC	VendorID 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.

460 **Table 48 — ExtendedErrorData format for vendor or other standards-defined ERROR response message**

Byte offset	Field	Size (bytes)	Description
0	Len	1	<p>Length of the VendorID field.</p> <p>If the vendor defines the error, the value of this field shall equal the Vendor ID Len, as <a href="#">Table 47 — Registry or standards body ID</a> describes, of the corresponding registry or standard body name.</p> <p>If a registry or a standard defines the error, this field shall be zero ( 0 ), which also indicates that the VendorID field is not present.</p> <p>The Error Data field in the ERROR message indicates the registry or standards body name (that is, Param2 ), and is one of the values in the ID column in <a href="#">Table 47 — Registry or standards body ID</a>.</p>
1	Vendor ID	Len	<p>The value of this field shall indicate the Vendor ID, as assigned by the registry or standards body. <a href="#">Table 47 — Registry or standards body ID</a> describes the length of this field. Shall be in little endian format.</p> <p>The registry or standards body name in the ERROR is indicated in the Error Data field (that is, Param2 ), and is one of the values in the ID column in <a href="#">Table 47 — Registry or standards body ID</a>.</p>
1 + Len	OpaqueErrorData	Variable	The vendor or standards define this value.

461 **Table 49 — ExtendedErrorData format for ResponseTooLarge**

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

462 **Table 50 — ExtendedErrorData format for LargeResponse**

Byte offset	Field	Size (bytes)	Description
0	Handle	1	<p>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.</p> <p>The value of this field should either entirely monotonically increase or entirely monotonically 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. See <a href="#">CHUNK_GET request and CHUNK_RESPONSE response message</a>.</p>

### 463 10.12.1 Standard body or vendor-defined header

464 This specification uses the format that [Table 51 — Standard 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. Note, if the payload format in question is defined by a standards body, the SVH header does not require the use of the `VendorID` field. Instead, the `ID` field would be set to the ID of the standards body, `VendorIDLen` would be set to `0`, and `VendorID` would be absent. A standards body, registry, or vendor that defines a payload format should also define the values to use in the SVH header.

465 **Table 51 — Standard 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 <a href="#">Table 47 — Registry or standards body ID</a> .
1	VendorIDLen	1	Length in bytes of the <code>VendorID</code> field.  If the format of the given payload is specified by a standards body or registry itself, this field shall be 0.  Otherwise, if the format of the given payload is specified by an organization that is identified on the vendor ID list indicated in the <code>ID</code> field, this field shall be the length indicated in the “Vendor ID length” column of <a href="#">Table 47 — Registry or standards body ID</a> for the respective <code>ID</code> .
2	VendorID	VendorIDLen	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.

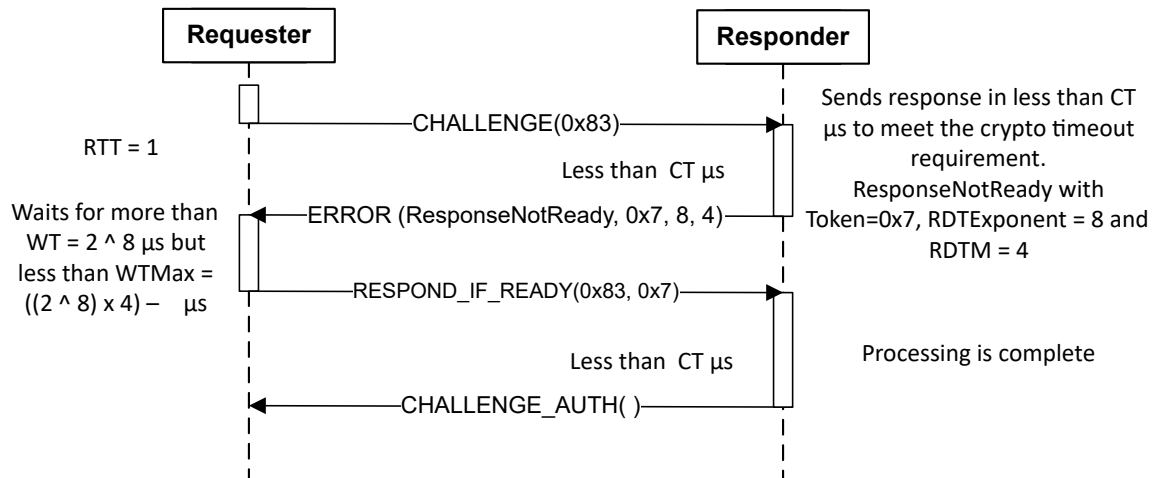
### 466 10.13 RESPOND\_IF\_READY request message format

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

468 [Figure 14 — RESPOND\\_IF\\_READY flow leading to completion](#) shows the `RESPOND_IF_READY` flow:

469 **Figure 14 — RESPOND\_IF\_READY flow leading to completion**

470



471 [Table 52 — RESPOND\\_IF\\_READY request message format](#) shows the RESPOND\_IF\_READY request message format.

472 **Table 52 — 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	0xFF = RESPOND_IF_READY . See Table 4 — SPDM request codes.
2	Param1	1	The original request code that triggered the ResponseNotReady error code response. Shall match the request code returned as part of the ResponseNotReady extended error data.
3	Param2	1	The token that was returned as part of the ResponseNotReady extended error data.

### 473 10.14 VENDOR\_DEFINED\_REQUEST request message

474 A Requester intending to define a unique request to meet its need can use this request message. [Table 53 — VENDOR\\_DEFINED\\_REQUEST request message format](#) defines the format.

475 The Requester should send this request message only after sending GET\_VERSION , GET\_CAPABILITIES , and NEGOTIATE\_ALGORITHMS request sequence.

476 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.

477 [Table 53 — VENDOR\\_DEFINED\\_REQUEST request message format](#) shows the `VENDOR_DEFINED_REQUEST` request message format.

478 **Table 53 — VENDOR\_DEFINED\_REQUEST request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0xFE</code> = <code>VENDOR_DEFINED_REQUEST</code> . See <a href="#">Table 4 — SPDM request codes</a> .
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 <b>ID</b> column in <a href="#">Table 47 — Registry or standards body ID</a> .
6	Len	1	Length of the <code>VendorID</code> field. If the <code>VendorDefinedRequest</code> is standard defined, <code>Len</code> shall be <code>0</code> . If the <code>VendorDefinedRequest</code> is vendor-defined, <code>Len</code> shall equal <code>VendorIDLen</code> , as <a href="#">Table 47 — Registry or standards body ID</a> describes.
7	VendorID	Len	Vendor ID, as assigned by the registry or standards body. Shall be in little endian format.
7 + Len	ReqLength	2	Length of the <code>VendorDefinedReqPayload</code> .
7 + Len + 2	VendorDefinedReqPayload	ReqLength	The standard or vendor shall use this field to send the request payload.

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

## 480 10.15 VENDOR\_DEFINED\_RESPONSE response message

481 A Responder can use this response message in response to `VENDOR_DEFINED_REQUEST`. [Table 54 — VENDOR\\_DEFINED\\_RESPONSE response message format](#) defines the format.

482 **Table 54 — VENDOR\_DEFINED\_RESPONSE response message format**

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

Byte offset	Field	Size (bytes)	Description
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	StandardID	2	Shall indicate the registry or standard body using one of the values in the <b>ID</b> column in <a href="#">Table 47 — Registry or standards body ID</a> .
6	Len	1	Length of the VendorID field. If the VendorDefinedRequest is standards-defined, length shall be 0. If the VendorDefinedRequest is vendor-defined, length shall equal VendorIDLen, as <a href="#">Table 47 — Registry or standards body ID</a> describes.
7	VendorID	Len	Shall indicate the Vendor ID, as assigned by the registry or standards body. Shall be in little endian format.
7 + Len	RespLength	2	Length of the VendorDefinedRespPayload
7 + Len + 2	VendorDefinedRespPayload	ReqLength	Standard or vendor shall use this value to send the response payload.

**483 10.15.1 VendorDefinedReqPayload and VendorDefinedRespPayload defined by DMTF specifications**

484 Other DMTF specifications may define `VENDOR_DEFINED_REQUEST` and `VENDOR_DEFINED_RESPONSE` messages with `StandardID` set to 0 (“DMTF”, as defined in [Table 47 — 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 55 — Format of VendorDefinedReqPayload and VendorDefinedRespPayload when StandardID is DMTF](#).

485 **Table 55 — Format of VendorDefinedReqPayload and VendorDefinedRespPayload when StandardID is DMTF**

Byte offset	Field	Size (bytes)	Description
0	DSPNumber	2	Shall be the DMTF specification's DSP number in a 16-bit integer. For example, DSP0287 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 <a href="#">Table 10 — VersionNumberEntry definition</a> .

Byte offset	Field	Size (bytes)	Description
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.

## 486 10.16 KEY\_EXCHANGE request and KEY\_EXCHANGE\_RSP response messages

487 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 56 — KEY\\_EXCHANGE request message format](#) shows the `KEY_EXCHANGE` request message format and [Table 58 — 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, and depends on the tight coupling between the two request/response message pairs.

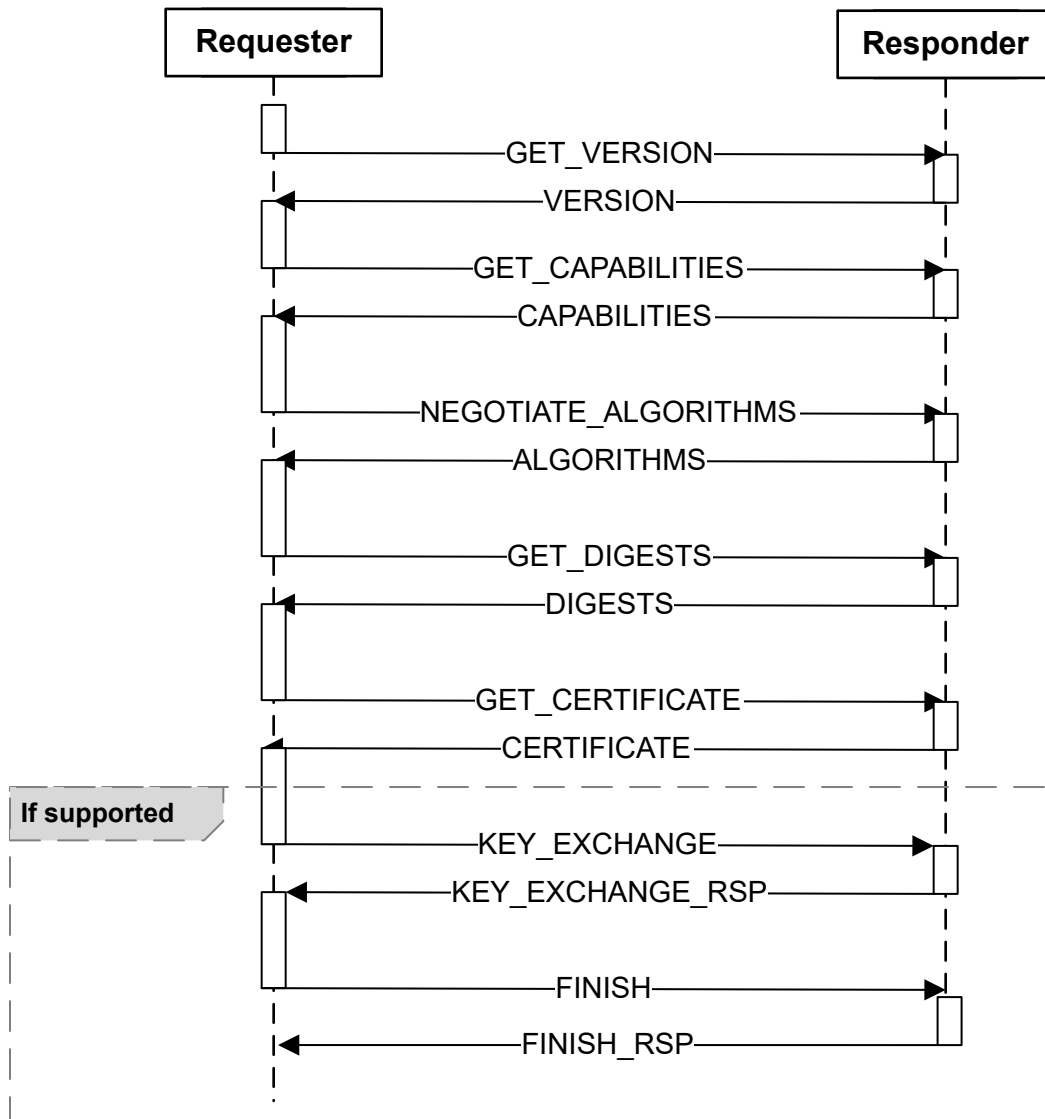
488 The Requester and Responder pair can support two modes of handshakes. If `HANDSHAKE_IN_THE_CLEAR_CAP` is set in both the Requester and the Responder all SPDM messages exchanged during the Session Handshake Phase are sent in the clear (outside of a secure session). Otherwise both the Requester and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

489 [Figure 15 — Responder authentication key exchange example](#) shows an example of a Responder authentication key exchange:

490 **Figure 15 — Responder authentication key exchange example**



491

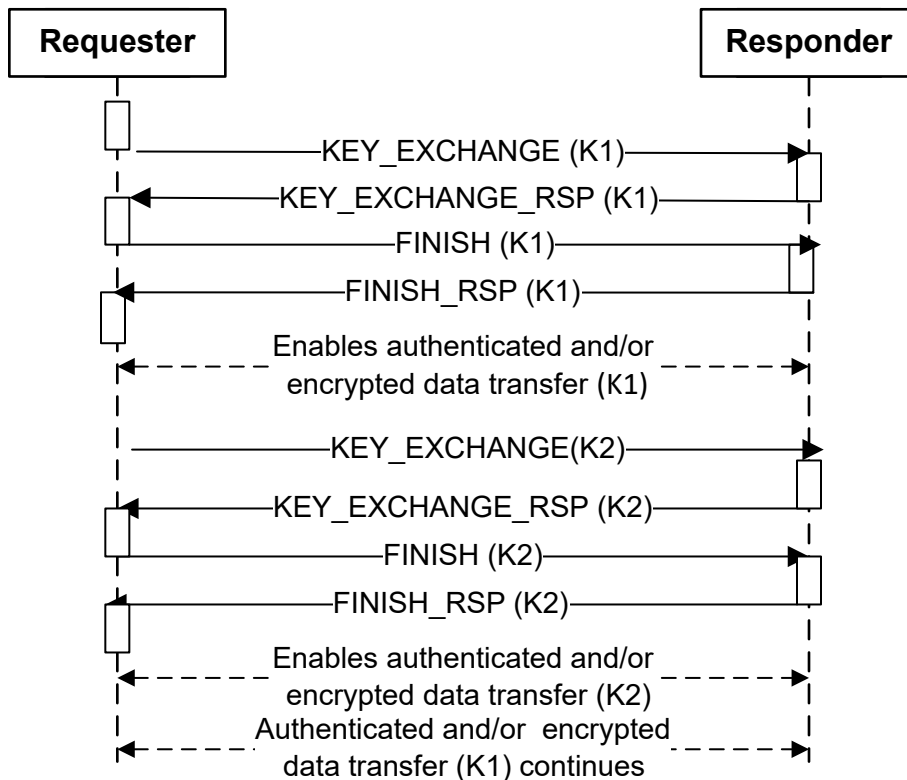


492 **Figure 16 — Responder authentication multiple key exchange example** shows an example of multiple sessions using two independent sets of root session keys that coexist at the same time. When `HANDSHAKE_IN_THE_CLEAR_CAP = 0` for both Requester and Responder, the specification does not require a 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. If the handshake is performed in the clear (that is, if `HANDSHAKE_IN_THE_CLEAR_CAP = 1` for both Requester and Responder), a Requester shall not send a second `KEY_EXCHANGE` request message until the first `FINISH_RSP` response message is received. A

Responder shall respond with an `ERROR` message of `ErrorCode=UnexpectedRequest` if it receives a second `KEY_EXCHANGE` request message before the first `FINISH` request is received.

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

494



495 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 [RFC8446](#). Assuming that the public keys were received correctly, both the Requester and Responder generate identical DHE secrets from which session secrets are generated.

496 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 [RFC8446](#). 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 15 — 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 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 15 — DHE structure](#)).

497 For SM2\_P256 key exchange, an additional identifier,  $ID_A$  and  $ID_B$ , that the [GB/T 32918.3-2016](#) specification defines, is 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 shall be the concatenation of “Responder-KEP-dmtf-spdm-v” and `SPDMversionString`.

498 A Requester should generate a fresh DHE key pair for each `KEY_EXCHANGE` request message that the Requester sends. A Responder should generate a fresh DHE key pair for each `KEY_EXCHANGE_RSP` response message that the Responder sends.

499 **Table 56 — KEY\_EXCHANGE request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0xE4</code> = <code>KEY_EXCHANGE</code> . See <a href="#">Table 4 — SPDM request codes</a> .
2	Param1	1	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 <code>CAPABILITIES</code> response), the Requester shall set this value to <code>0x0</code> .
3	Param2	1	<code>SlotID</code> . Slot number of the Responder certificate chain that shall be used for authentication. The value in this field shall be between 0 and 7 inclusive. It shall be <code>0xFF</code> if the public key of the Responder was provisioned to the Requester previously.
4	ReqSessionID	2	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	SessionPolicy	1	See <a href="#">Table 57 — Session policy</a> .

Byte offset	Field	Size (bytes)	Description
7	Reserved	1	Reserved.
8	RandomData	32	Requester-provided random data.
40	ExchangeData	D	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.
40 + D	OpaqueDataLength	2	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	<code>OpaqueDataLength</code>	If present, <code>OpaqueData</code> sent by the Requester. Used to indicate any parameters that 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> .

500 **Table 57 — Session policy**

Bit offset	Field	Description
0	TerminationPolicy	This field specifies the behavior of the Responder when the Responder completes a runtime code or configuration update which 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.  To terminate a session, the Responder shall respond with <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.
[7:1]	Reserved	Reserved

501 **Table 58 — 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 <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0x64 = KEY_EXCHANGE_RSP</code> . See <a href="#">Table 5 — SPDM response codes</a> .
2	Param1	1	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, but means that a heartbeat is not desired on this session.
3	Param2	1	Reserved.
4	RspSessionID	2	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 (<a href="#">Session-based mutual authentication</a>) 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 GET_DIGESTS 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 <a href="#">GET_ENCAPSULATED_REQUEST request and ENCAPSULATED_REQUEST response messages</a> clause.</p>

Byte offset	Field	Size (bytes)	Description
7	SlotIDParam	1	<p>Bit [7:4]. Reserved.</p> <p>Bit [3:0]. SlotID. Slot number of the Requester certificate chain that shall be used for mutual authentication, if MutAuthRequested Bit 0 is set. The value in this field shall be between 0 and 7 inclusive, or 0xF if the public key of the Requester was provisioned to the Responder through other means. All other values Reserved.</p> <p>For any other value of MutAuthRequested this field shall be set to 0 and ignored by the Requester.</p>
8	RandomData	32	Responder-provided random data.
40	ExchangeData	D	<p>DHE public information generated by the Responder. If the DHE group selected in the most recent ALGORITHMS response is finite-field-based (FFDHE), the ExchangeData represents the computed public value. If the selected DHE group is elliptic curve-based (ECDHE), the ExchangeData 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.</p>

Byte offset	Field	Size (bytes)	Description
40 + D	MeasurementSummaryHash	H	<p>If the Responder does not support measurements ( <code>MEAS_CAP=00b</code> in <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(Concatenation(MeasurementBlock[0], MeasurementBlock[1], ...))</code> where <code>MeasurementBlock[x]</code> denotes a measurement of an element in the TCB. Measurements are concatenated in ascending order based on their measurement index as <a href="#">Table 42 — Measurement block format</a> describes.</p> <p>When the requested <code>Param1 = 0x1</code> and 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(Concatenation(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 <a href="#">Measurement index assignments</a> clause.</p> <p>If the Responder supports both raw bit stream and digest representations for a given measurement index, then the Responder shall use the digest form.</p> <p>This field shall be in <a href="#">Hash byte order</a>.</p>
40 + D + H	OpaqueDataLength	2	<p>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.</p>
42 + D + H	OpaqueData	<code>OpaqueDataLength</code>	<p>If present, <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>.</p>

Byte offset	Field	Size (bytes)	Description
42 + D + H + OpaqueDataLength	Signature	SigLen	Signature over the transcript. SigLen is the size of the asymmetric signing algorithm output the Responder selected via the last ALGORITHMS response message to the Requester. The <a href="#">Transcript for KEY_EXCHANGE_RSP signature</a> defines the construction of the transcript.
42 + D + H + OpaqueDataLength + SigLen	ResponderVerifyData	H	<p>Conditional field.</p> <p>If the Session Handshake Phase is encrypted and/or message authenticated, then this field shall be of length H and it shall equal the HMAC of the transcript hash, using finished_key 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 KEY_EXCHANGE_RSP HMAC as <a href="#">Transcript for KEY_EXCHANGE_RSP HMAC</a> shows. The finished_key shall be derived from the Response Direction Handshake Secret and is described in the <a href="#">finished_key derivation</a> clause. HMAC is described in <a href="#">RFC2104</a>.</p> <p>If both the Requester and Responder set HANDSHAKE_IN_THE_CLEAR_CAP to 1, then this field shall be absent.</p>

## 502 10.16.1 Session-based mutual authentication

503 Mutual authentication for KEY\_EXCHANGE occurs in the session handshake phase of a session.

504 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](#).

505 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 using ErrorCode=UnexpectedRequest and shall terminate the session.

506 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.



**507 10.16.1.1 Specify Requester certificate for session-based mutual authentication**

508 The SPDM key exchange protocol is optimized to perform key exchange with the least number of messages exchanged. When Responder-only authentication, or mutual authentication where the Responder has obtained the certificate chains of the Requester in a previous interaction is performed, key exchange is carried out with two request/response message pairs ( KEY\_EXCHANGE , 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 authenticating a Requester whose certificates it has not obtained in a previous interaction, using a slot other than the default (Slot 0), has no way of knowing in advance which SlotID value to use.

509 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 a ENCAPSULATED\_RESPONSE\_ACK request with Param2 = 0x02 and an 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.

510 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 .

**511 10.17 FINISH request and FINISH\_RSP response messages**

512 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. [Table 59 — FINISH request message format](#) shows the FINISH request message format and [Table 60 — Successful FINISH\\_RSP response message format](#) shows the FINISH\_RSP response message format.

**513 Table 59 — FINISH request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	0xE5 = FINISH . See <a href="#">Table 4 — SPDM request codes</a> .
2	Param1	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.

Byte offset	Field	Size (bytes)	Description
3	Param2	1	SlotID . Only valid if Param1 = 0x01 , otherwise reserved. Slot number of the Requester certificate chain that shall be authenticated in Signature field. The value in this field shall be between 0 and 7 inclusive. It shall be 0xFF if the public key of the Requester was provisioned to the Responder through other means.
4	Signature	SigLen	Signature over the transcript. SigLen is the size of the asymmetric signing algorithm ( ReqBaseAsymAlg ) output the Responder selected via the last ALGORITHMS response message to the Requester. If Param1 = 0x00 , SigLen is zero and this field shall not be present. 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 finished_key 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 FINISH HMAC, mutual authentication as Transcript for FINISH HMAC, mutual authentication shows. Otherwise, it shall be the hash of the transcript for FINISH HMAC, Responder-only authentication as Transcript for FINISH HMAC, Responder-only authentication shows. The finished_key shall be derived from Request Direction Handshake Secret and is described in the finished_key derivation clauses. HMAC is described in RFC2104.

514 The following clause applies when the handshake is performed in the clear, that is, when both Requester and Responder have set HANDSHAKE\_IN\_THE\_CLEAR\_CAP to 1:

515 If KEY\_EXCHANGE\_RSP.MutAuthRequested equals either 0x02 or 0x04 , 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 .

516 **Table 60 — Successful 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	0x65 = FINISH_RSP . See Table 5 — SPDM response codes.
2	Param1	1	Reserved.

Byte offset	Field	Size (bytes)	Description
3	Param2	1	Reserved.
4	ResponderVerifyData	H	<p>Conditional field.</p> <p>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.</p> <p>If both the Requester and Responder support <code>HANDSHAKE_IN_THE_CLEAR_CAP</code> field, this field shall be of length H and it 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, mutual authentication as <a href="#">Transcript for FINISH_RSP HMAC, mutual authentication</a> shows. Otherwise, the transcript hash shall be the hash of the transcript for <code>FINISH_RSP</code> HMAC, Responder-only authentication as <a href="#">Transcript for FINISH_RSP HMAC, Responder-only authentication</a> shows. The <code>finished_key</code> shall be derived from Response Direction Handshake Secret and is described in the <a href="#">finished_key derivation</a> clause. HMAC is described in <a href="#">RFC2104</a>.</p>

517 **10.17.1 Transcript and transcript hash calculation rules**

518 The transcript is a concatenation of the prescribed full messages or message fields in order. In the case where a message is transferred in chunks, only the complete message that is built by the concatenation of chunk payloads shall be added to the transcript. Consequently, the transcript hash is the hash of the transcript using the negotiated hash algorithm (that is, `BaseHashSel` or `ExtHashSel` of `ALGORITHMS`). In general, the transcript is used in signature fields and the transcript hash is used in calculating the HMAC in `KEY_EXCHANGE`, `FINISH` and their corresponding responses. For messages that are encrypted, the plaintext messages are used in the transcript. Where this clause 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 26 — Certificate chain format](#) describes.

519 The notation `[${message_name}] . ${field_name}` is used, where:

- `${message_name}` is the name of the request or response message.
- `${field_name}` is the name of the field in the request or response message. The asterisk (`*`) means all fields in that message, except from any conditional fields that are empty (for example `KEY_EXCHANGE . OpaqueData`).

520 [Transcript for KEY\\_EXCHANGE\\_RSP signature](#) shows the transcript for the `KEY_EXCHANGE_RSP` signature:

521 **Transcript for KEY\_EXCHANGE\_RSP signature**

1. VCA
2. 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.
3. [KEY\_EXCHANGE] . \*
4. [KEY\_EXCHANGE\_RSP] . \* except the Signature and ResponderVerifyData fields.

522 The Responder shall generate the KEY\_EXCHANGE\_RSP signature from:

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

523 where

- SPDMsign is described by the [Signature generation](#) clause.
- PrivKey shall be the private key of the leaf certificate of the Responder.
- transcript shall be the concatenation of the messages for a KEY\_EXCHANGE\_RSP signature.

524 The leaf certificate of the Responder shall be the one indicated by SlotID in Param2 of KEY\_EXCHANGE request.

525 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 the 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.

526 [Transcript for KEY\\_EXCHANGE\\_RSP HMAC](#) shows the transcript for KEY\_EXCHANGE\_RSP HMAC:

527 **Transcript for KEY\_EXCHANGE\_RSP HMAC**

1. VCA
2. 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.
3. [KEY\_EXCHANGE] . \*
4. [KEY\_EXCHANGE\_RSP] . \* except the ResponderVerifyData field.

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

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

1. VCA
2. 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.
3. [KEY\_EXCHANGE] . \*
4. [KEY\_EXCHANGE\_RSP] . \*
5. 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.

#### 6. [FINISH] . SPDM Header Fields

- 530 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](#).
- 531 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 FINISH request. `SPDMsignatureVerify` is described in [Signature verification](#). A successful verification is when `SPDMsignatureVerify` returns success.
- 532 [Transcript for FINISH HMAC, Responder-only authentication](#) shows the transcript for FINISH HMAC with Responder-only authentication:
- 533 **Transcript for FINISH HMAC, Responder-only authentication**
1. VCA
  2. 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.
  3. `[KEY_EXCHANGE] . *`
  4. `[KEY_EXCHANGE_RSP] . *`
  5. `[FINISH] . SPDM Header Fields`
- 534 [Transcript for FINISH HMAC, mutual authentication](#) shows the transcript for FINISH HMAC with mutual authentication:
- 535 **Transcript for FINISH HMAC, mutual authentication**
1. VCA
  2. 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.
  3. `[KEY_EXCHANGE] . *`
  4. `[KEY_EXCHANGE_RSP] . *`
  5. 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.
  6. `[FINISH] . SPDM Header Fields`
  7. `[FINISH] . Signature`
- 536 [Transcript for FINISH\\_RSP HMAC, Responder-only authentication](#) shows the transcript for FINISH\_RSP HMAC with Responder-only authentication:
- 537 **Transcript for FINISH\_RSP HMAC, Responder-only authentication**
1. VCA
  2. 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.

3. [KEY\_EXCHANGE] . \*
4. [KEY\_EXCHANGE\_RSP] . \*
5. [FINISH] . \*
6. [FINISH\_RSP] . SPDM Header fields

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

539 **Transcript for FINISH\_RSP HMAC, mutual authentication**

1. VCA
2. 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.
3. [KEY\_EXCHANGE] . \*
4. [KEY\_EXCHANGE\_RSP] . \*
5. 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.
6. [FINISH] . \*
7. [FINISH\_RSP] . SPDM Header fields

540 When multiple session keys are being established between the same Requester and Responder pair, Signature over the transcript during FINISH request is computed using only the corresponding KEY\_EXCHANGE , KEY\_EXCHANGE\_RSP and FINISH request parameters.

541 For additional rules, see [General ordering rules](#).

## 542 10.18 PSK\_EXCHANGE request and PSK\_EXCHANGE\_RSP response messages

---

543 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 the session key establishment, even if asymmetric-key cryptography is supported.

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

545 A Requester can be paired with multiple Responders. Likewise, a Responder can be paired with multiple Requesters. A Requester and 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.

546 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 out of scope of this specification. The size of the provisioned PSK is determined by the requirement of security strength of the application, but should be at least 128 bits and recommended to be 256 bits or larger, to resist dictionary attacks especially when the Requester and Responder cannot both contribute sufficient entropy during the exchange.

547 Two message pairs are defined for this option:

- PSK\_EXCHANGE / PSK\_EXCHANGE\_RSP
- PSK\_FINISH / PSK\_FINISH\_RSP

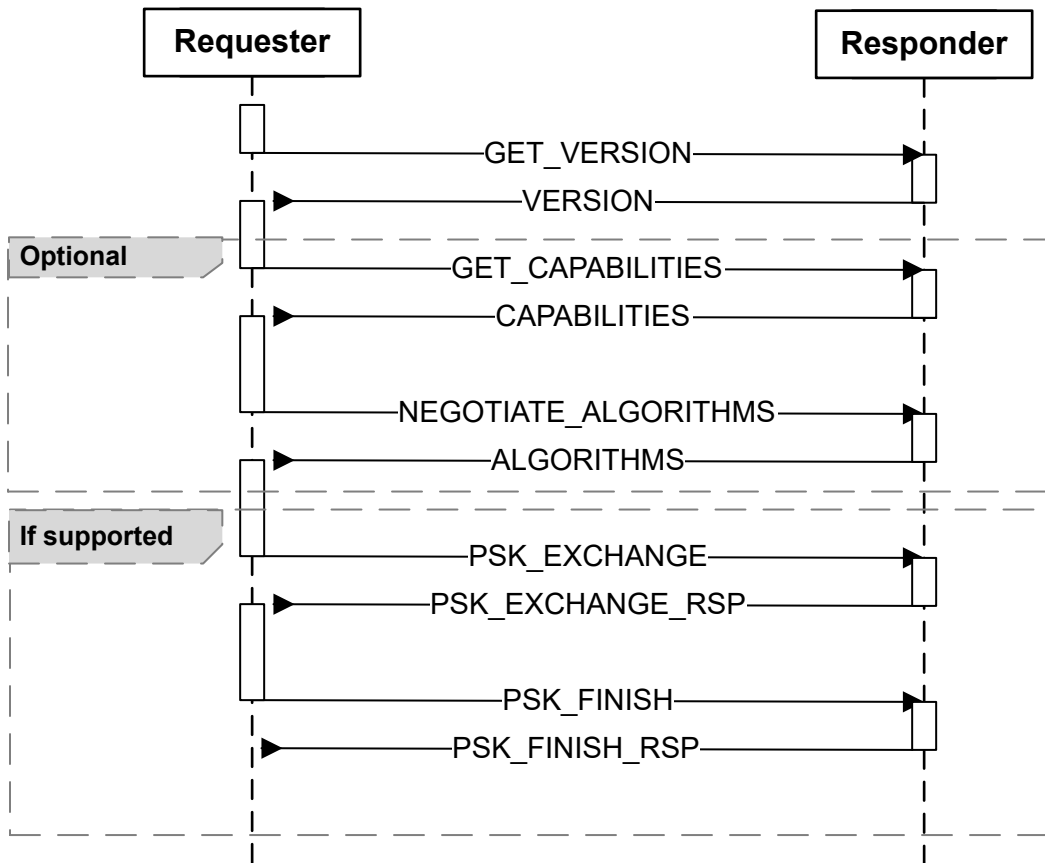
548 The PSK\_EXCHANGE message carries three responsibilities:

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.

549 [Figure 17 — PSK\\_EXCHANGE: Example](#) shows an example of the PSK\_EXCHANGE message:

550 **Figure 17 — PSK\_EXCHANGE: Example**

551



552 **Table 61 — PSK\_EXCHANGE request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	0xE6 = PSK_EXCHANGE . See <a href="#">Table 4 — SPDM request codes</a> .



Byte offset	Field	Size (bytes)	Description
2	Param1	1	Type of measurement summary hash requested: 0x0 : No measurement summary hash requested. 0x1 : TCB measurements only. 0xFF : All measurements. All other values reserved. If a Responder does not support measurements ( MEAS_CAP=00b in CAPABILITIES response), the Requester shall set this value to 0x0 .
3	Param2	1	Session policy. See <a href="#">Table 57 — Session policy</a> .
4	ReqSessionID	2	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	Length of PSKHint in bytes.
8	R	2	Length of RequesterContext in bytes.
10	OpaqueDataLength	2	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.
12	PSKHint	P	Information required by the Responder to retrieve the PSK. Optional.
12 + P	RequesterContext	R	The context of the Requester. Shall include a nonce (random number or monotonic counter) of at least 32 bytes and optionally relevant information contributed by the Requester.
12 + P + R	OpaqueData	OpaqueDataLength	Optional. If present, the OpaqueData sent by the Requester is used to indicate any parameters that 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 .

553 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.

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

555 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 [Key schedule](#).
4. Constructs `PSK_EXCHANGE_RSP` response message and sends to the Requester.

556 **Table 62 — PSK\_EXCHANGE\_RSP response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	0x66 = <code>PSK_EXCHANGE_RSP</code> . See <a href="#">Table 5 — SPDM response codes</a> .
2	Param1	1	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, but means that a heartbeat is not desired on this session.
3	Param2	1	Reserved.
4	RspSessionID	2	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	Reserved	2	Reserved.
8	Q	2	Length of <code>ResponderContext</code> in bytes.
10	OpaqueDataLength	2	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.

Byte offset	Field	Size (bytes)	Description
12	MeasurementSummaryHash	H	<p>If the Responder does not support measurements ( <code>MEAS_CAP=00b</code> in <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(Concatenation(MeasurementBlock[0], MeasurementBlock[1], ...))</code> where <code>MeasurementBlock[x]</code> denotes a measurement of an element in the TCB. Measurements are concatenated in ascending order based on their measurement index as <a href="#">Table 42 — Measurement block format</a> describes.</p> <p>When the requested <code>Param1 = 0x1</code> and 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(Concatenation(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 <a href="#">Measurement index assignments</a> clause.</p> <p>If the Responder supports both raw bit stream and digest representations for a given measurement index, then the Responder shall use the digest form.</p> <p>This field shall be in <a href="#">Hash byte order</a>.</p>
12 + H	ResponderContext	Q	Context of the Responder. Optional. If present, shall include a nonce and/or information contributed by the Responder.
12 + H + Q	OpaqueData	OpaqueDataLength	Optional. If present, the <code>OpaqueData</code> sent by the Responder is used to indicate any parameters that 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 + H + Q + OpaqueDataLength	ResponderVerifyData	H	Data to be verified by the Requester using the <code>finished_key</code> of the Responder.

- 557 The `ResponderContext` is the contribution of the Responder to session key derivation. It should contain a nonce (random number or monotonic counter) and other information of the Responder. If a monotonic counter is used as the nonce, the monotonic 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.
- 558 It should be noted 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 - a man-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 or session keys of the prior replayed session are not compromised.
- 559 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.
- 560 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

- 561 Note that, even if `CERTIFICATE`, `CHALLENGE_AUTH`, and/or `MEASUREMENTS` were issued, these messages would not be included in the data for calculating `ResponderVerifyData`. In other words, the identity of the signer of `CHALLENGE_AUTH` and/or `MEASUREMENTS` is not bound to identity of the sender of `PSK_EXCHANGE_RSP`. Therefore, to mitigate Responder identity impersonation, the Requester should not issue `PSK_EXCHANGE` if it has received `CHALLENGE_AUTH` and/or `MEASUREMENTS` with a signature from the Responder.

- 562 Upon receiving `PSK_EXCHANGE_RSP`, the Requester:

1. Derives the `finished_key` of the Responder by following [Key schedule](#).
2. Verify `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 `PSK_CAP` in the `CAPABILITIES` of the Responder is `10b`, construct `PSK_FINISH` request and send to the Responder.

563 **10.19 PSK\_FINISH request and PSK\_FINISH\_RSP response messages**

564 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 .

565 [Table 63 — PSK\\_FINISH request message format](#) describes the PSK\_FINISH request message format:

566 **Table 63 — PSK\_FINISH request message format**

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

567 To calculate RequesterVerifyData , the Requester calculates an HMAC. The key is the finished\_key of the Requester, as described in [Key schedule](#). 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

568 For additional rules, see [General ordering rules](#).

569 Upon receiving PSK\_FINISH request, the Responder derives the finished\_key of the Requester and calculates the HMAC independently in the same manner and verifies the result matches RequesterVerifyData . If verified, the Responder constructs PSK\_FINISH\_RSP response and sends to the Requester. Otherwise, the Responder sends ERROR response with error code InvalidRequest to the Requester.

570 [Table 64 — Successful PSK\\_FINISH\\_RSP response message format](#) describes the successful PSK\_FINISH\_RSP response message format:

571 **Table 64 — Successful PSK\_FINISH\_RSP response message format**

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

572 **10.20 HEARTBEAT request and HEARTBEAT\_ACK response messages**

573 This request shall keep a session alive if `HEARTBEAT` is supported by both the Requester and Responder. The `HEARTBEAT` request shall be sent periodically as indicated in `HeartbeatPeriod` in either `KEY_EXCHANGE_RSP` or `PSK_EXCHANGE_RSP` response messages. The Responder shall terminate the session if session traffic is not received in twice `HeartbeatPeriod` . Likewise, the Requester shall terminate the session if session traffic, including `ERROR` response, is not received in twice `HeartbeatPeriod` . Session traffic includes encrypted data at the transport layer. How SPDM is informed of encrypted data at the transport layer is outside the scope of this specification. The Requester can retry `HEARTBEAT` requests.

574 The timer for the Heartbeat period shall start at the transmission, for Responders, or reception, for Requester, of the appropriate `FINISH_RSP` , `PSK_FINISH_RSP` ( `PSK_CAP` of Responder is `10b` ), or `PSK_EXCHANGE_RSP` ( `PSK_CAP` of Responder is `01b` ) response messages. When determining the value of `HeartbeatPeriod`, the Responder should ensure this value is sufficiently greater than `T1` .

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

576 [Table 65 — HEARTBEAT request message format](#) describes the message format.

577 **Table 65 — HEARTBEAT request message format**

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

578 [Table 66 — HEARTBEAT\\_ACK response message format](#) describes the format for the Heartbeat Response.

579 **Table 66 — HEARTBEAT\_ACK response message format**

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

**580 10.20.1 Heartbeat additional information**

581 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.

**582 10.21 KEY\_UPDATE request and KEY\_UPDATE\_ACK response messages**

583 To update session keys, this request shall be used. There are many reasons for doing this but an important one is when the per-record nonce will soon reach its maximum value and rollover. The KEY\_UPDATE request can be issued by the Responder as well using the GET\_ENCAPSULATED\_REQUEST mechanism. A KEY\_UPDATE request shall perform the operation given in Param1 and defined in [Table 69 — 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. 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.

584 [Table 67 — KEY\\_UPDATE request message format](#) describes the KEY\_UPDATE request message format:

**585 Table 67 — KEY\_UPDATE request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0xE9</code> = KEY_UPDATE Request. See <a href="#">Table 4 — SPDM request codes</a> .
2	Param1	1	Key operation. See <a href="#">Table 69 — KEY_UPDATE operations</a> .
3	Param2	1	Tag. This field shall contain a unique number to aid the Responder in differentiating between the original and retry request. A retry request shall contain the same tag number as the original.

586 [Table 68 — KEY\\_UPDATE\\_ACK response message format](#) describes the KEY\_UPDATE\_ACK response message format:

587 **Table 68 — KEY\_UPDATE\_ACK response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	0x69 = <code>KEY_UPDATE_ACK</code> response. See <a href="#">Table 5 — SPDM response codes</a> .
2	Param1	1	Key Operation. This field shall reflect the Key Operation field of the request. See <a href="#">Table 69 — KEY_UPDATE operations</a>
3	Param2	1	Tag. This field shall reflect the Tag number in the <code>KEY_UPDATE</code> request.

588 [Table 69 — KEY\\_UPDATE operations](#) describes the `KEY_UPDATE` operations:589 **Table 69 — 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 keys for both directions.
3	VerifyNewKey	Ensure the key update is successful and the old keys can be safely discarded.
4 - 255	Reserved	Reserved.

590 **10.21.1 Session key update synchronization**

591 For clarity, in the key update process, 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.

592 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.

593 Both the sender and the receiver shall derive the new keys as detailed in [Major secrets update](#).

594 The sender shall not use the new transmit key until after reception of the `KEY_UPDATE_ACK` response.

595 The sender and receiver shall use the new key on the `KEY_UPDATE` request with `VerifyNewKey` command and all subsequent commands until another key update is performed.

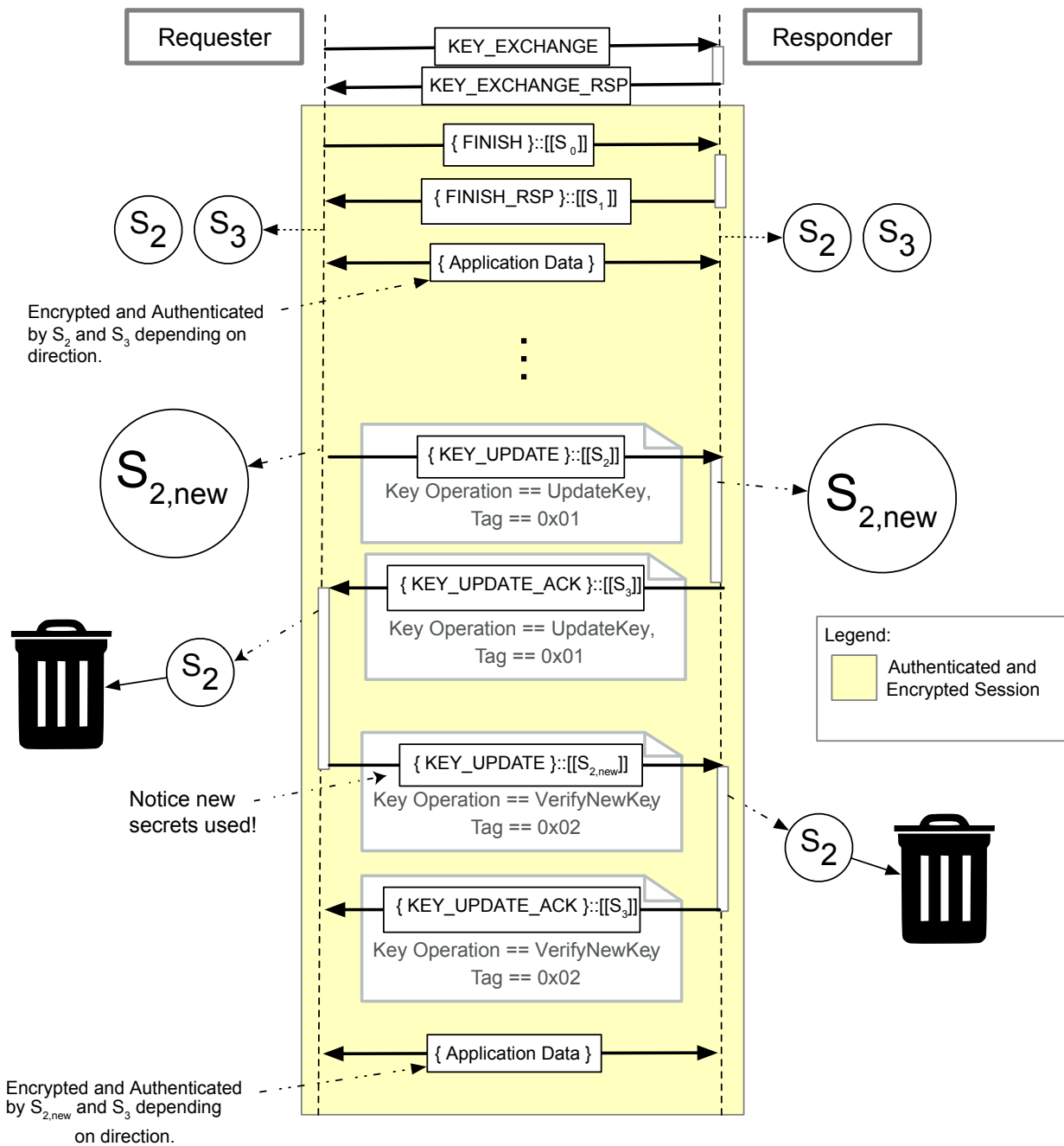
596 In the case of `KEY_UPDATE` request with `UpdateAllKeys`, the receiver shall use the new transmit key for the



`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.

- 597 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`.
- 598 Upon the successful reception of the `KEY_UPDATE_ACK`, the sender shall transmit a `KEY_UPDATE` request with `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.
- 599 For `UpdateKey`, upon successful reception and verification of the `KEY_UPDATE` with `VerifyNewKey` operation, the receiver can discard the old session keys. For `UpdateAllKeys`, upon successful reception and verification of the `KEY_UPDATE_ACK` with `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 `VerifyNewKey` operation, the receiver can discard the old session keys that protect sender-sent messages.
- 600 In certain scenarios, the receiver might need additional time to process the `KEY_UPDATE` request such as processing data already in its buffer. Thus, the receiver can reply with an `ERROR` message with `ErrorCode=Busy`. The sender should retry the request after a reasonable period of time with a reasonable amount of retries to prevent premature session termination.
- 601 Finally, it bears repeating that a key update in one direction can happen simultaneously with a key update in the opposite direction. Still, the aforementioned synchronization process occurs independently but simultaneously for each direction.
- 602 [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.
- 603 **Figure 18 — KEY\_UPDATE protocol example flow**

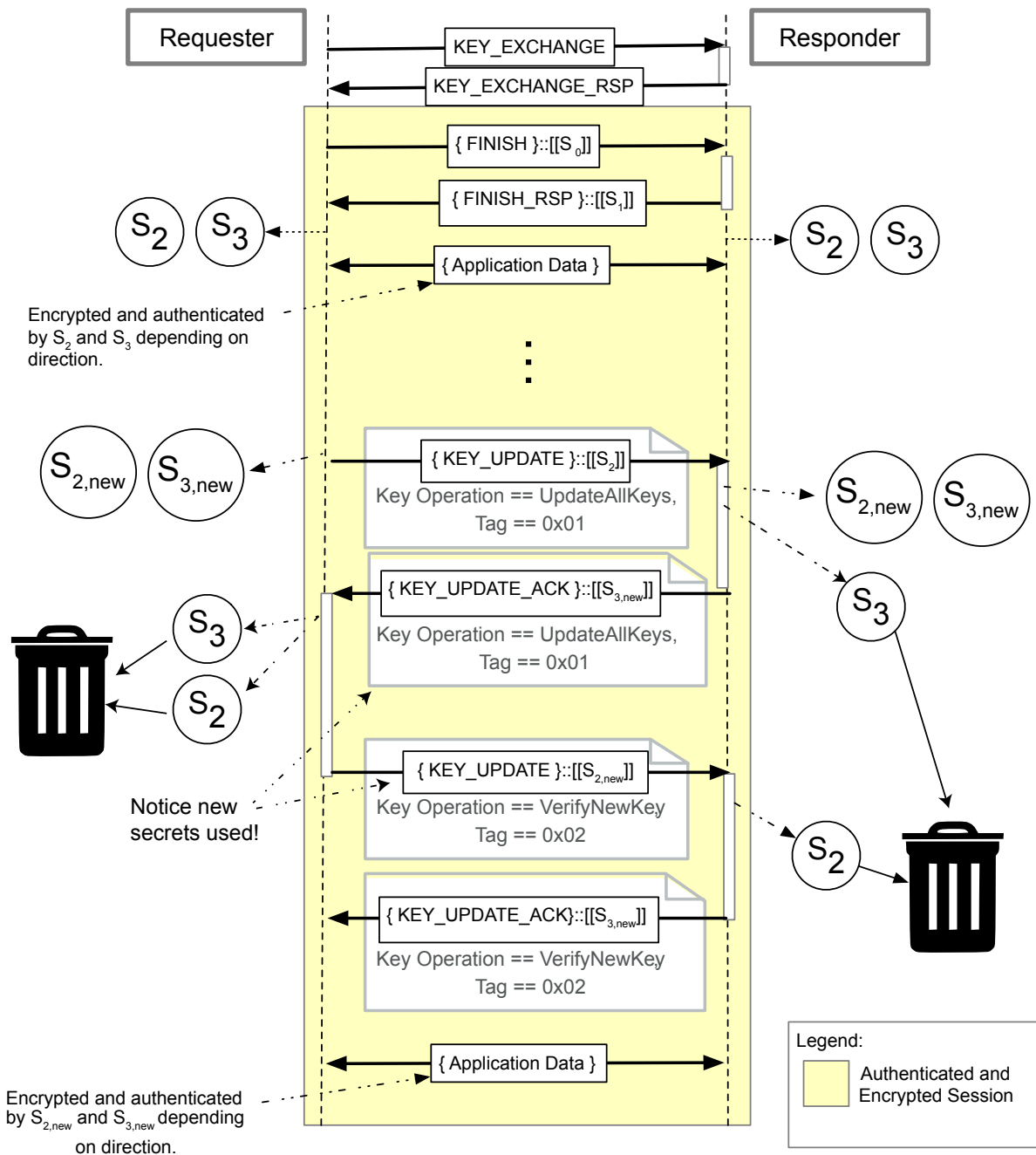
604



605 **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.

606 **Figure 19 — KEY\_UPDATE protocol change all keys example flow**

607



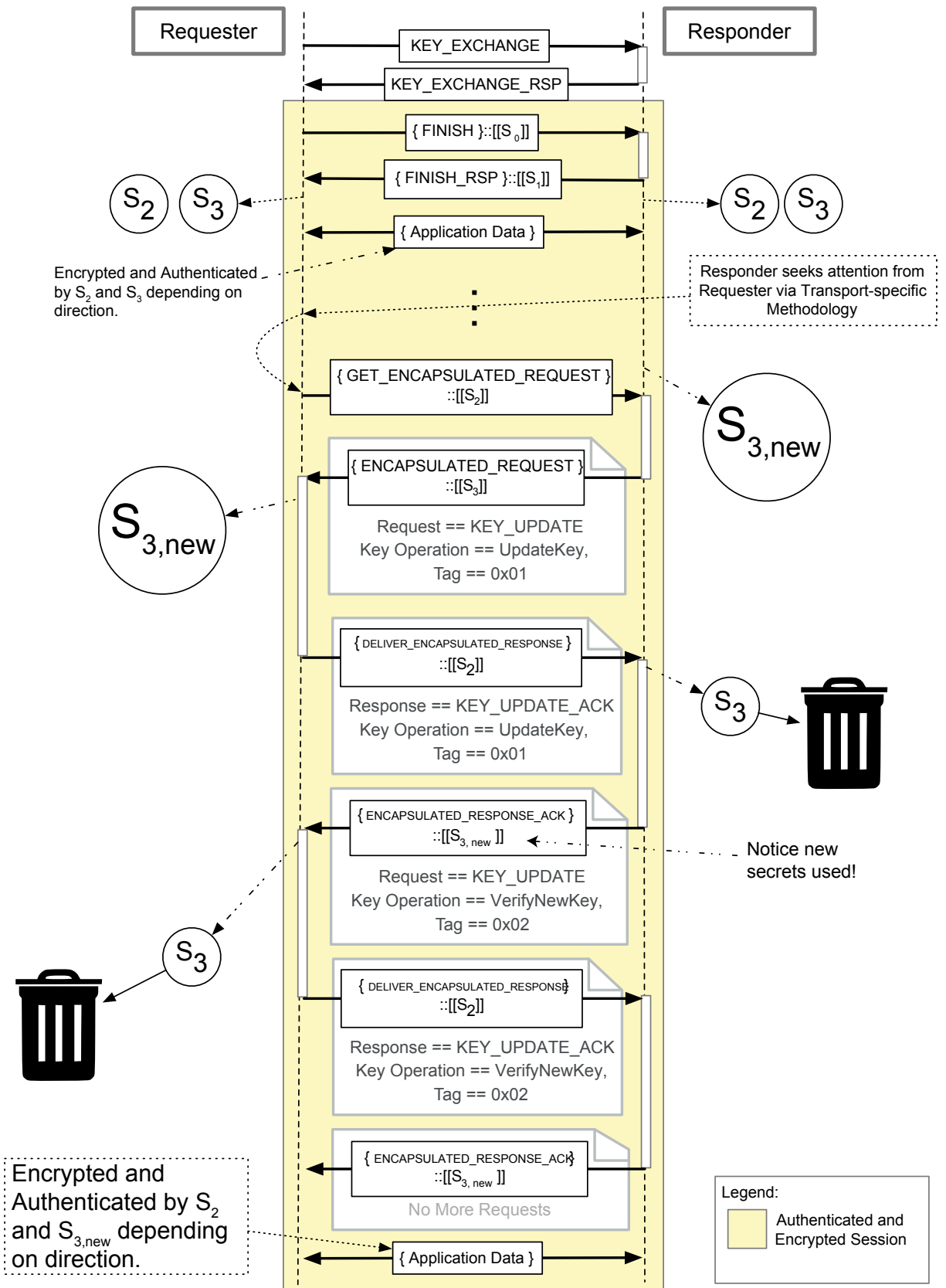
**608 10.21.2 KEY\_UPDATE transport allowances**

609 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. That definition is outside the scope of this specification.

610 [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 master) 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. Also, in this same example, the Requester and Responder are both capable of encryption and message authentication.

611 **Figure 20 — KEY\_UPDATE protocol example flow 2**

612



## 613 **10.22 GET\_ENCAPSULATED\_REQUEST request and ENCAPSULATED\_REQUEST response messages**

---

614 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. [Figure 21 — Session-based mutual authentication example](#) and [Figure 22 — Optimized session-based mutual authentication example](#) illustrate the use of this scheme.

615 A Requester issues a `GET_ENCAPSULATED_REQUEST` request message to retrieve an encapsulated SPDM request message from the Responder. The response to this message (`ENCAPSULATED_REQUEST`) encapsulates the SPDM request message as if the Responder was acting as a Requester. [Table 70 — GET\\_ENCAPSULATED\\_REQUEST request message format](#) describes the request message format. The Responder shall use the same SPDM version the Requester used.

### 616 **10.22.1 Encapsulated request flow**

617 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](#).

618 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, with the exception of `GET_VERSION`, `RESPOND_IF_READY` and `DELIVER_ENCAPSULATED_RESPONSE`, the Requester shall not issue any other message. If a Responder receives a request other than `DELIVER_ENCAPSULATED_RESPONSE`, `RESPOND_IF_READY` or `GET_VERSION`, the Responder should respond with `ErrorCode=RequestInFlight`.

### 619 **10.22.2 Optimized encapsulated request flow**

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

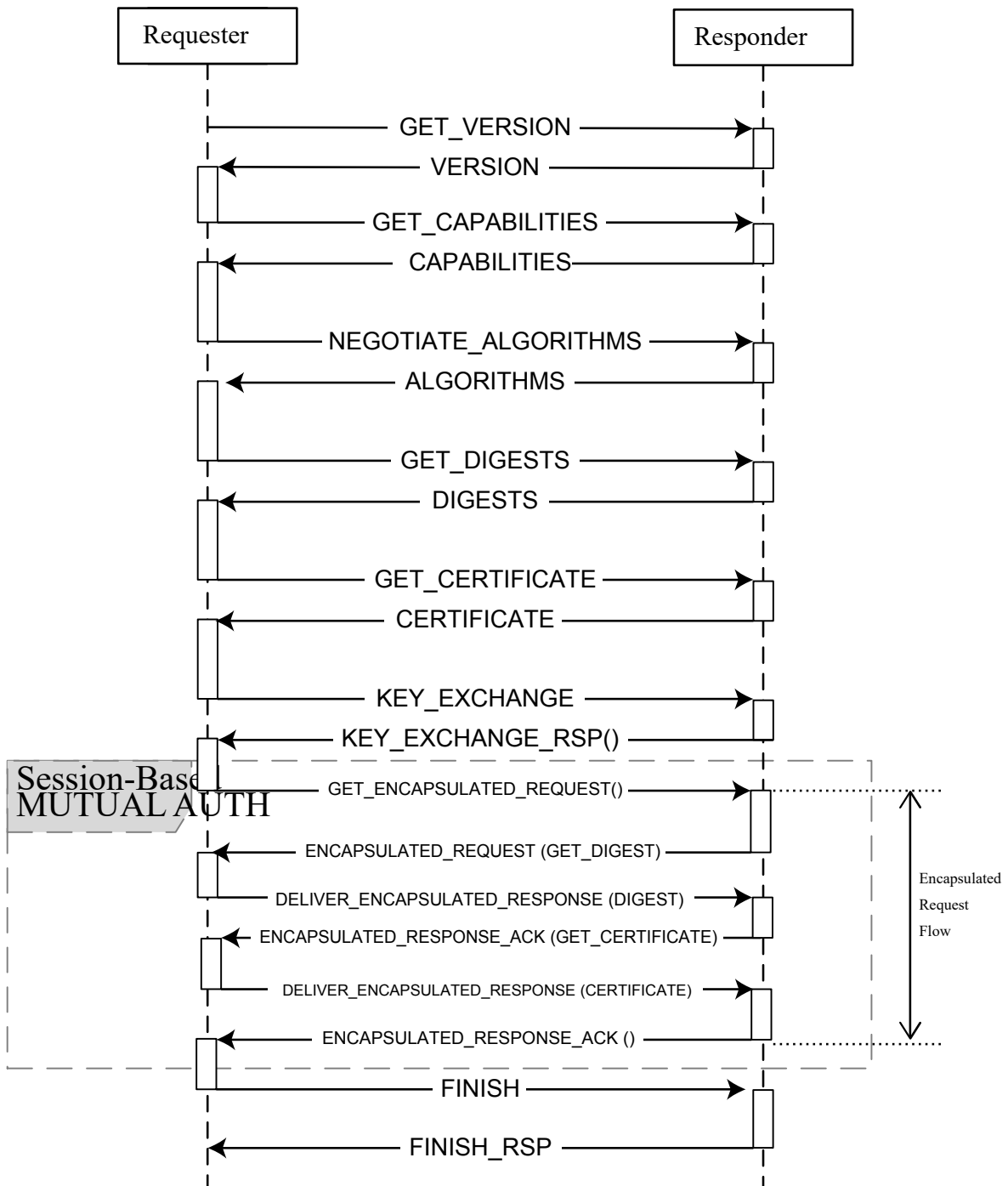
621 When the Requester successfully receives a `Session-Secrets-Exchange` response with an included encapsulated request, the Requester shall send a `DELIVER_ENCAPSULATED_RESPONSE` after processing the encapsulated request. The Requester shall not issue any other requests except for `DELIVER_ENCAPSULATED_RESPONSE`, `RESPOND_IF_READY` and `GET_VERSION`. If a Responder receives a request other than `DELIVER_ENCAPSULATED_RESPONSE`, `RESPOND_IF_READY` or `GET_VERSION`, then the Responder should respond with `ErrorCode=RequestInFlight`.

---

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

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

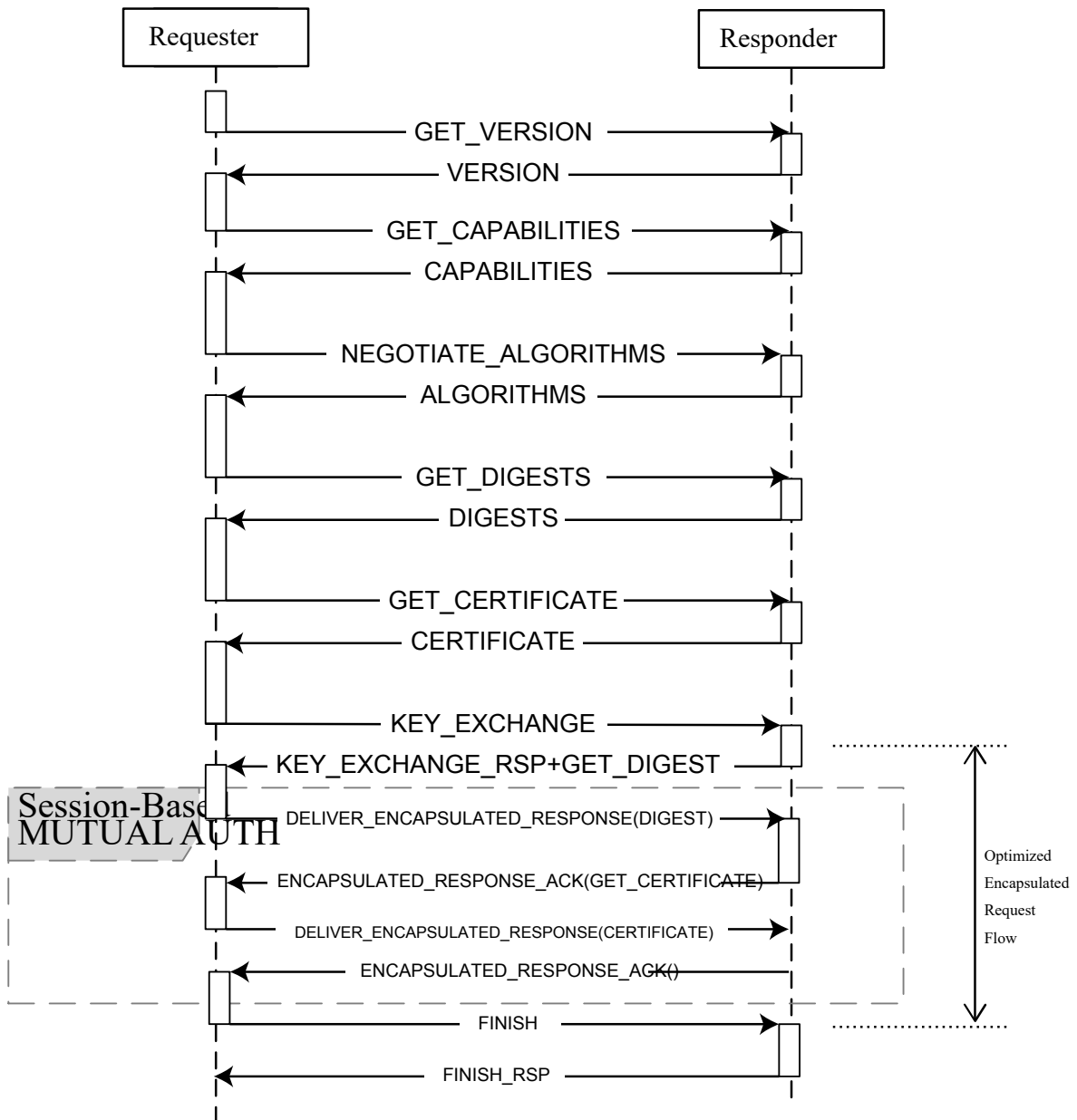
624



625 [Figure 22 — Optimized session-based mutual authentication example](#) shows an example of optimized session-based mutual authentication:

626 **Figure 22 — Optimized session-based mutual authentication example**

627



628 **Table 70 — GET\_ENCAPSULATED\_REQUEST request message format**



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

629 [Table 71 — ENCAPSULATED\\_REQUEST response message format](#) describes the format this response.

630 **Table 71 — ENCAPSULATED\_REQUEST response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0x6A</code> = <code>ENCAPSULATED_REQUEST</code> response. See <a href="#">Table 5 — SPDM response codes</a> .
2	Param1	1	Request ID.  This field should be unique to help the Responder match response to request.
3	Param2	1	Reserved.
4	EncapsulatedRequest	Variable	SPDM Request Message.  The value of this field shall represent a valid SPDM request message. The length of this field is dependent on the SPDM Request message. The field shall start with the <code>SPDMVersion</code> field. The <code>SPDMVersion</code> field of the <code>Encapsulated Request</code> shall be the same as <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 <code>ErrorCode=UnexpectedRequest</code> if these requests are encapsulated.

631 **10.22.3 Triggering `GET_ENCAPSULATED_REQUEST`**

632 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.

### 633 10.22.4 Additional constraints

634 The `GET_ENCAPSULATED_REQUEST` and `ENCAPSULATED_REQUEST` messages shall only be allowed to encapsulate certain requests in certain scenarios. For constraint details, see the [Session, Basic mutual authentication](#), and [KEY\\_UPDATE request and KEY\\_UPDATE\\_ACK response messages](#) clauses.

## 635 10.23 DELIVER\_ENCAPSULATED\_RESPONSE request and ENCAPSULATED\_RESPONSE\_ACK response messages

636 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.

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

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

639 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.

640 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](#).

641 [Table 72 — DELIVER\\_ENCAPSULATED\\_RESPONSE request message format](#) describes the request message format.

642 **Table 72 — DELIVER\_ENCAPSULATED\_RESPONSE request message format**

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

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

643 [Table 73 — ENCAPSULATED\\_RESPONSE\\_ACK response message format](#) describes the response message format.

644 **Table 73 — ENCAPSULATED\_RESPONSE\_ACK response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the SPDMVersion as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	0x6B = ENCAPSULATED_RESPONSE_ACK. See <a href="#">Table 5 — SPDM response codes</a> .
2	Param1	1	Request ID.  If EncapsulatedRequest is present and Param2 = 0x01, then this field should contain a unique, non-zero number to help the Responder match response to request. Otherwise, this field shall be 0x00.

Byte offset	Field	Size (bytes)	Description
3	Param2	1	<p>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>
4	AckRequestID	1	<p>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 a retry.</p>
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 SPDMVersion of the ENCAPSULATED_REQUEST response. Both GET_ENCAPSULATED_REQUEST and DELIVER_ENCAPSULATED_RESPONSE shall be invalid requests and the Requester shall respond with ErrorCode=UnexpectedRequest if these requests are encapsulated.</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>

### 645 10.23.1 Additional information

646 Using a unique request ID is highly recommended to aid the Responder in avoiding confusion between a retry and a new DELIVER\_ENCAPSULATED\_RESPONSE message. For example, if the Responder sent the ENCAPSULATED\_RESPONSE\_ACK with a new encapsulated request and that 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 when, in fact, it was a retry. This mistake might cause additional mistakes to occur.

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

648 The `DELIVER_ENCAPSULATED_RESPONSE` and `ENCAPSULATED_RESPONSE_ACK` messages shall only be allowed to encapsulate certain requests in certain scenarios. For constraint details, see [Session, Basic mutual authentication](#), and [KEY\\_UPDATE request and KEY\\_UPDATE\\_ACK response messages](#) clauses.

### 649 **10.23.2 Allowance for encapsulated requests**

650 Only certain requests can be encapsulated in any encapsulated request flow. Their corresponding response, including `ERROR`, can be encapsulated too. Additionally, these requests are only allowed in certain flows, such as [Basic Mutual Authentication](#), and are described in various parts of this specification. The consolidated list of requests allowed to be encapsulated shall be these requests:

- `CHALLENGE`
- `GET_CERTIFICATE`
- `GET_DIGESTS`
- `KEY_UPDATE`

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

### 652 **10.23.3 Certain error handling in encapsulated flows**

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

#### 654 **10.23.3.1 Response not ready**

655 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 with a `ResponseNotReady` error code, 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.

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

657 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, the Requester can drop the original response.

**658 10.23.3.2 Timeouts**

659 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.

660 Both Responder and Request should comply with the timing requirements laid forth in [Timing requirements](#).

**661 10.24 END\_SESSION request and END\_SESSION\_ACK response messages**

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

663 [Table 74 — END\\_SESSION request message format](#) and [Table 75 — End session request attributes](#) describe this format.

664 **Table 74 — END\_SESSION request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0xEC = END_SESSION</code> . See <a href="#">Table 4 — SPDM request codes</a> .
2	Param1	1	See <a href="#">Table 75 — End session request attributes</a> .
3	Param2	1	Reserved.

665 **Table 75 — End session request attributes**

Bit offset	Value	Field	Description
0	0	Negotiated State Clearing Indicator	If the Responder supports Negotiated State caching ( <code>CACHE_CAP=1</code> ), the Responder shall preserve the cached Negotiated State. Otherwise, this field shall be ignored.
0	1	Negotiated State Clearing Indicator	If the Responder supports Negotiated State caching ( <code>CACHE_CAP=1</code> ), the Responder shall also clear the cached Negotiated State as part of session termination. If there is no cached 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.

666 [Table 76 — END\\_SESSION\\_ACK response message format](#) describes the response message.

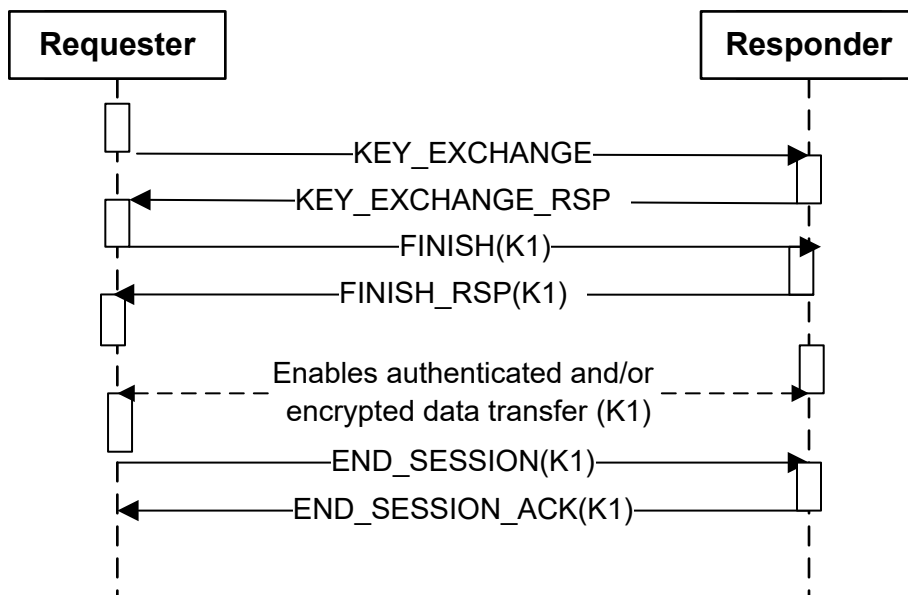
667 **Table 76 — END\_SESSION\_ACK response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0x6C = END_SESSION_ACK</code> . See <a href="#">Table 5 — SPDM response codes</a> .
2	Param1	1	Reserved.
3	Param2	1	Reserved.

668 [Figure 23 — END\\_SESSION protocol flow](#) shows the `END_SESSION` protocol flow:

669 **Figure 23 — END\_SESSION protocol flow**

670



671 **10.25 Certificate provisioning**

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

### 673 10.25.1 GET\_CSR request and CSR response messages

674 The GET\_CSR request message shall retrieve a Certificate Signing Request (CSR) from the Responder.

675 A Responder shall only process a GET\_CSR request if it already possesses an appropriate asymmetric key pair for each of the signature suites (algorithms and associated parameters) it supports. If more than one signature suite is supported, selection of the appropriate signature suite (and thus 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 others contributed by the Responder itself. RequesterInfo format shall comply to the PKCS #10 specification in RFC2986, specifically the CertificationRequestInfo format. Vendor-defined extensions shall be encoded using the Attributes type. The Responder may alter the value of requested CertificationRequestInfo fields in RequesterInfo when generating CSRdata. The Responder shall return an ERROR message with error code InvalidRequest if it cannot support a field included in the RequesterInfo, or if the value of a requested field is not supported and the Responder cannot alter the value of the field. If the Responder receives a GET\_CSR request while another GET\_CSR request is outstanding, the Responder shall overwrite the existing request and process the new GET\_CSR request.

676 If the device requires a reset to complete the GET\_CSR request, then the device shall respond with an ErrorCode=ResetRequired response. If the device requires a reset to complete the GET\_CSR request, the device shall persist information through the reset to allow it to identify whether a GET\_CSR received after the reset matches the one from before the reset. After the Responder has processed the reset, the Requester sends the same GET\_CSR request that was sent before the reset, which signals to the Responder to send the CSR response. If the Requester sends a different GET\_CSR request than the request that was sent before the reset, then the Responder shall treat it as a new request or reject the different GET\_CSR request with an ErrorCode=UnexpectedRequest response. On receipt of a new GET\_CSR request, the Responder can discard any existing CSR responses.

677 The attributes of the resulting CSR and their values shall comply with the clauses presented in the SPDM certificate requirements and recommendations section. If the device conforms to the DeviceCert model (ALIAS\_CERT\_CAP=0b in CAPABILITIES response), the resulting CSR shall be for a Device Certificate. If the device conforms to the AliasCert model (ALIAS\_CERT\_CAP=1b in CAPABILITIES response), the resulting CSR shall be for a Device Certificate CA. See Identity provisioning for more details.

678 Table 77 — GET\_CSR request message format shows the GET\_CSR request message format.

679 Table 78 — CSR response message format shows the CSR response message format.

680 Fields from the resulting CSR contained in a successful CSR response will have to be assembled into a certificate and 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.

681 **Table 77 — GET\_CSR request 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	0xED = GET_CSR . See Table 4 — SPDM request codes.
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	RequesterInfoLength	2	Length of RequesterInfo field in bytes provided by the Requester. This field can be 0.
6	OpaqueDataLength	2	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.
8	RequesterInfo	RequesterInfoLength	Optional information provided by the Requester. This field shall be DER-encoded.
8 + RequesterInfoLength	OpaqueData	OpaqueDataLength	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 OtherParamsSelection .

682 **Table 78 — CSR response message format**

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

683 The CSRdata format shall comply to the PKCS #10 specification in RFC2986, specifically the CertificationRequest format.

684 **10.25.2 SET\_CERTIFICATE request and SET\_CERTIFICATE\_RSP response messages**

685 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 is done outside of a trusted environment for slots 1-7, the Requester shall issue `SET_CERTIFICATE` inside a secure session. Mutual authentication and/or other means for authorizing 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 `ErrorCode=ResetRequired` response. When `ResetRequired` is pending and the device receives a new `SET_CERTIFICATE` request for the same slot number, the device shall overwrite the existing `CertChain` and process the new `SET_CERTIFICATE` request. If the device temporarily cannot write to a slot, including in a case when it receives overlapping `SET_CERTIFICATE` requests from different Requesters, then the device shall respond with an `ErrorCode=Busy` response.

686 When a reset is required for a pending previous `SET_CERTIFICATE` request and the device receives a `GET_CERTIFICATE` request for a pending slot or a `GET_DIGESTS` request, the device shall respond with an `ErrorCode=ResetRequired` response.

687 [Table 79 — SET\\_CERTIFICATE request message format](#) shows the `SET_CERTIFICATE` request message format.

688 [Table 80 — Successful SET\\_CERTIFICATE\\_RSP response message format](#) shows the `SET_CERTIFICATE_RSP` response message format.

689 **Table 79 — SET\_CERTIFICATE request message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0xEE</code> = <code>SET_CERTIFICATE</code> . See <a href="#">Table 4 — SPDM request codes</a> .
2	Param1	1	Bit [7:4]. Reserved. Bit [3:0]. <code>SlotID</code> where the new certificate is written. The value in this field shall be between 0 and 7 inclusive.
3	Param2	1	Reserved.
4	CertChain	Variable	Shall be the contents of the target certificate chain as specified in <a href="#">Certificates and certificate chains</a> 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 <code>CAPABILITIES</code> response), this field shall contain a partial certificate chain from the root CA to the Device Certificate CA.

690 **Table 80 — 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 <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0x6E</code> = <code>SET_CERTIFICATE_RSP</code> . See <a href="#">Table 5 — SPDM response codes</a> .
2	Param1	1	Bit [7:4]. Reserved. Bit [3:0]. <code>SlotID</code> where the new certificate is written. The value in this field shall be the same as <code>SlotID</code> in corresponding <code>SET_CERTIFICATE</code> Request. The value in this field shall be between 0 and 7 inclusive.
3	Param2	1	Reserved.

## 691 10.26 Large SPDM message transfer mechanism

692 A large SPDM message is an SPDM message whose size is 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 when the size of an SPDM message exceeds 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.

693 This transfer mechanism divides a large SPDM message into smaller fragments called chunks. The chunks shall be numbered and shall transfer in sequence. The chunks and transfer sequence are as such:

- 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 until the last chunk. These numeric values are 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 until 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 in a monotonically increasing chunk sequence number until the last chunk.
- The chunked transfer shall not be interrupted by any commands that are not part of the chunk transfer sequence, with the exception of `GET_VERSION`. The Responder shall return the error `ErrorCode=UnexpectedRequest` if an unexpected command is received during the chunked transfer. If `CHUNK_GET` is invalid or corrupted, the Requester may receive corresponding error codes ( `ErrorCode=InvalidRequest` , `ErrorCode=VersionMismatch` , etc.). These error codes shall not interrupt the chunk transfer sequence, with exception of the error code `ErrorCode=DecryptError` .

- `CHUNK_SEND` , `CHUNK_GET` and their corresponding Responses shall be used to transfer these chunks.

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

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

## 696 10.26.1 `CHUNK_SEND` request and `CHUNK_SEND_ACK` response message

697 `CHUNK_SEND` request and `CHUNK_SEND_ACK` response shall be used to send a request to an SPDM endpoint when the size of the request is greater than the `DataTransferSize` of the receiving SPDM endpoint or the transmit buffer size of the sending SPDM endpoint.

698 [Table 81 — `CHUNK\_SEND` request format table](#) describes the format for the request.

699 [Table 82 — Chunk sender attributes](#) describes the chunk sender attributes:

### 700 Table 81 — `CHUNK_SEND` request format table

Byte offset	Field	Size (bytes)	Description
0	<code>SPDMVersion</code>	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	<code>RequestResponseCode</code>	1	<code>0x85</code> = <code>CHUNK_SEND</code> request. See <a href="#">Table 4 — SPDM request codes</a> .
2	<code>Param1</code>	1	Request Attributes. See <a href="#">Table 82 — Chunk sender attributes</a> .
3	<code>Param2</code>	1	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 entirely monotonically increase or entirely monotonically 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	<code>ChunkSeqNo</code>	2	Shall identify the chunk number associated with <code>SPDMChunk</code> .
6	Reserved	2	Reserved.
8	<code>ChunkSize</code>	4	Shall indicate the size of <code>SPDMchunk</code> . See <a href="#">Additional chunk transfer requirements</a> .
12	<code>LargeMessageSize</code>	<code>L0 = 0 or 4</code>	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.

Byte offset	Field	Size (bytes)	Description
12 + L0	SPDMchunk	Variable	Shall contain the chunk of the large SPDM request message associated with <code>ChunkSeqNo</code> .

701 **Table 82 — 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.

702 [Table 83 — CHUNK\\_SEND\\_ACK response message format](#) describes the format for the response.

703 **Table 83 — CHUNK\_SEND\_ACK response message format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	0x05 = CHUNK_SEND_ACK response. See <a href="#">Table 5 — SPDM response codes</a> .
2	Param1	1	Response attributes. See <a href="#">Table 84 — Chunk receiver attributes</a> .
3	Param2	1	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 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.

704 [Table 84 — Chunk receiver attributes](#) describes the chunk receiver attributes:

705 **Table 84 — 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.

706 [Table 83 — CHUNK\\_SEND\\_ACK response message format](#) describes the format for the response.

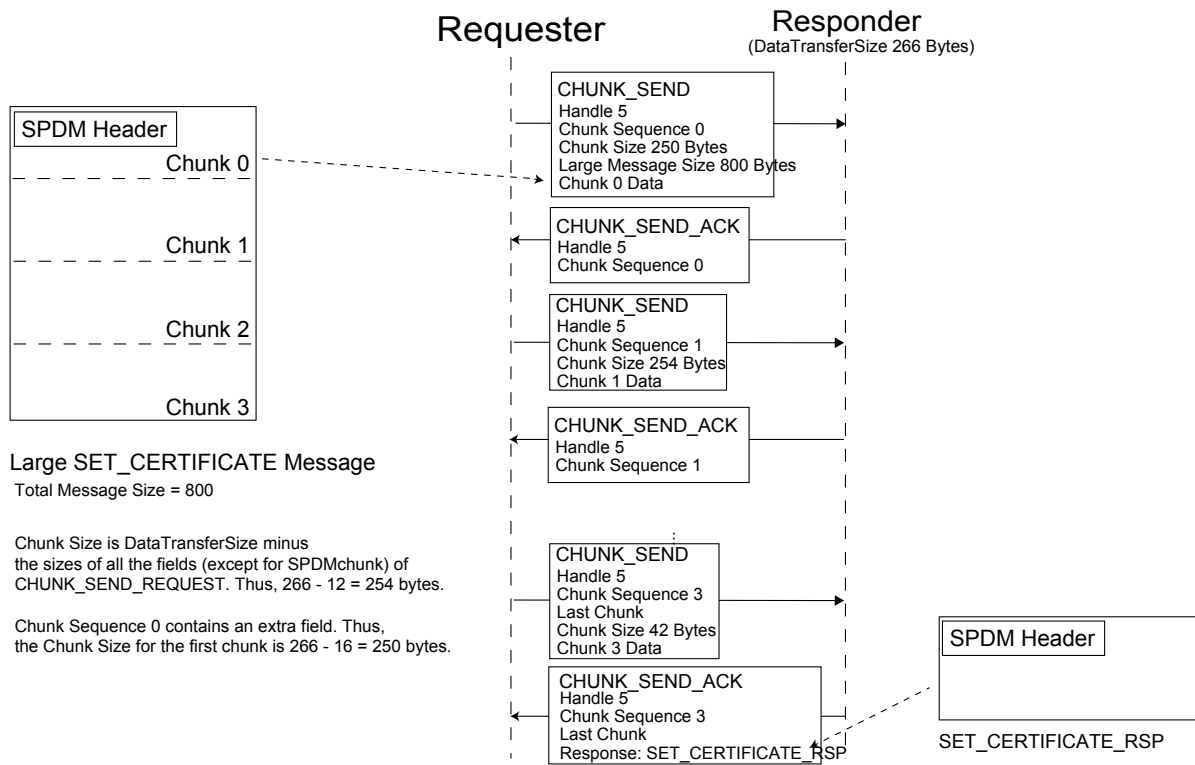
707 Upon reception of the last chunk, the receiving SPDM endpoint shall respond with the response corresponding to the large SPDM request message in `ResponseToLargeRequest`. If placing the response in `ResponseToLargeRequest` causes the size of the `CHUNK_SEND_ACK` to exceed `DataTransferSize`, the receiving end point shall, instead, respond to `CHUNK_SEND` with an `ERROR` message using `ErrorCode=LargeResponse`. An `ERROR` message with an `ErrorCode=LargeResponse` shall not be allowed in `ResponseToLargeRequest`. An `ERROR` messages with other `ErrorCodes` 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.

708 In the case where the size of the `CHUNK_SEND_ACK` message is greater than `DataTransferSize` but the size of `ResponseToLargeRequest` is less than `DataTransferSize` the Responder will chunk a message whose size is less than `DataTransferSize`.

709 [Figure 24 — Large SET\\_CERTIFICATE example](#) illustrates the sending of a large SPDM request message to a Responder.

710 **Figure 24 — Large SET\_CERTIFICATE example**

711



**712 10.26.2 CHUNK\_GET request and CHUNK\_RESPONSE response message**

713 `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 or the transmit buffer size of the SPDM endpoint sending the Response.

714 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 using `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 the `ERROR` message with `ErrorCode=LargeResponse`.

715 [Table 85 — `CHUNK\_GET` request format](#) describes the format for the request.

716 **Table 85 — `CHUNK_GET` request format**

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	0x86 = <code>CHUNK_GET</code> request. See <a href="#">Table 4 — SPDM request codes</a> .
2	Param1	1	Reserved.

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

717 [Table 86 — CHUNK\\_RESPONSE response format](#) describes the format for the response.

718 **Table 86 — CHUNK\_RESPONSE response format**

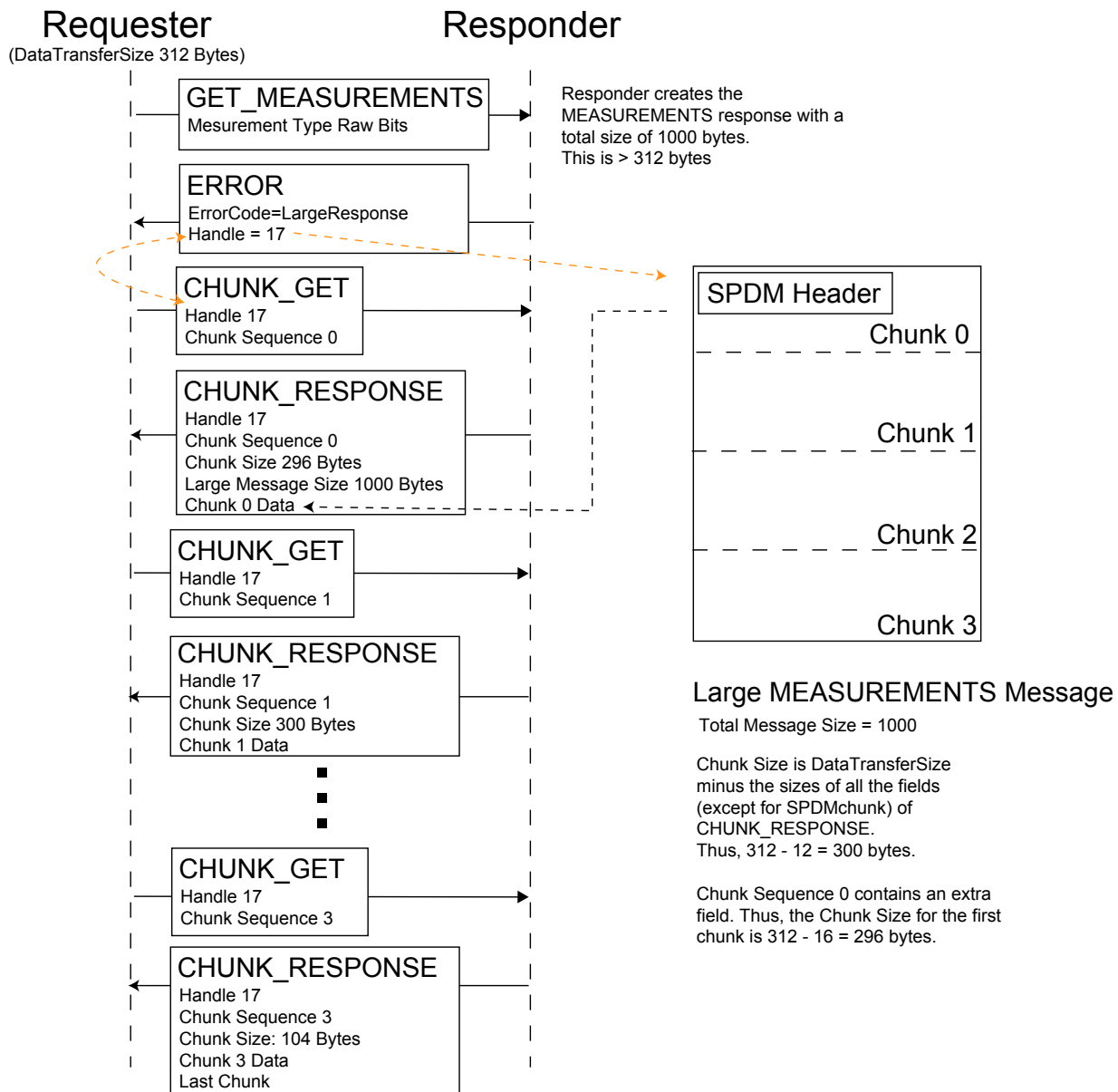
Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be the <code>SPDMVersion</code> as described in <a href="#">SPDM version</a> .
1	RequestResponseCode	1	<code>0x06 = CHUNK_RESPONSE</code> response. See <a href="#">Table 5 — SPDM response codes</a> .
2	Param1	1	Response attributes. See <a href="#">Table 82 — Chunk sender attributes</a> .
3	Param2	1	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 <a href="#">Additional chunk transfer requirements</a> .
12	LargeMessageSize	$L0 = 0$ 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.
$12 + L0$	SPDMchunk	Variable	Shall contain the chunk of the large SPDM request message associated with <code>ChunkSeqNo</code> .

719 [Figure 25 — Large MEASUREMENT example](#) illustrates the sending and retrieval of a Large SPDM Response to a Requester that issued a `GET_MEASUREMENT` request.

720 **Figure 25 — Large MEASUREMENT example**



721



**722 10.26.3 Additional chunk transfer requirements**

723 When transferring a large SPDM message, an SPDM endpoint shall be prohibited from transferring a chunk sequence number (that is, `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 silently discard the request or respond with an `ERROR` message with `ErrorCode = InvalidRequest`.

- 724 In general, 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 between `MinDataTransferSize` and 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.
- 725 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 reliability of this transfer mechanism and general interoperability, these messages shall be prohibited from being transferred in chunks using this transfer mechanism:
- `GET_VERSION`
  - `VERSION`
  - `GET_CAPABILITIES`
  - `CAPABILITIES`
  - `ERROR`
    - An `ERROR` message with `ErrorCodes` other than `LargeResponse` can be placed in `ResponseToLargeRequest` of `CHUNK_SEND_ACK` response.
- 726 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 computation operate on the large SPDM messages, themselves, which other parts of this specification define.
- 727 The response to a `CHUNK_SEND` or `CHUNK_GET` request, themselves, shall not be `ErrorCode=ResponseNotReady`. However, the `ResponseToLargeRequest` can contain an `ERROR` message with `ErrorCode=ResponseNotReady`.
- 728 While a large SPDM message is being transferred in chunks, the 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, the large SPDM request message is considered to be outstanding.

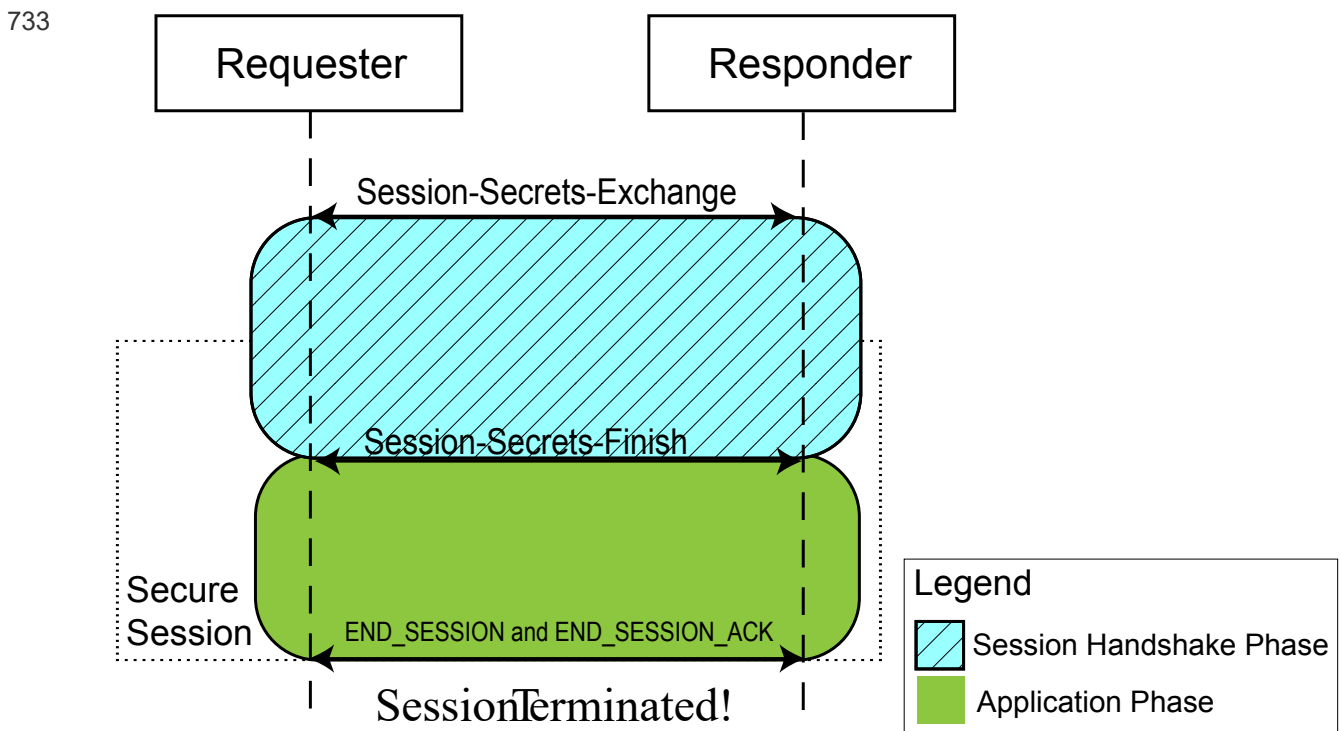
## 729 11 Session

730 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 or both of encryption or message authentication.

731 A session has three phases, as [Figure 26 — Session phases](#) shows:

- The handshake
- The application
- Termination

732 **Figure 26 — Session phases**



### 734 11.1 Session handshake phase

735 The session handshake phase begins with either `KEY_EXCHANGE` or `PSK_EXCHANGE`. This phase also allows for authentication of the Requester if the Responder indicated that earlier in `ALGORITHMS` response. Furthermore, this phase of the session uses the handshake secrets to secure the communication as described in the [Key schedule](#).

736 The purpose of this phase is to build trust between the Responder and Requester, first, before either side can send

application data. Additionally, it also ensures the integrity of the handshake and to a certain degree, synchronicity with the derived handshake secrets.

737 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 `ALGORITHMS` message. 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.

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

739 A successful handshake ends with either `FINISH_RSP` or `PSK_FINISH_RSP` and the application phase begins.

## 740 11.2 Application phase

---

741 Once the handshake completes and all validation passes, the session reaches the application phase where either the Responder and Requester can send application data.

742 During this phase, a Requester can send SPDM messages such as `GET_MEASUREMENT`. These messages might involve transcript calculations and 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.

743 The application phase ends when either the `HEARTBEAT` requirements fail, `END_SESSION` or an `ERROR` message with `ErrorCode = DecryptError`. The next phase is the session termination phase.

## 744 11.3 Session termination phase

---

745 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, outside the scope of this specification.

746 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.

747 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.

## 748 11.4 Simultaneous active sessions

---

749 At least one session per connection shall be supported if both Requester and Responder advertise `KEY_EXCHANGE` or `PSK_EXCHANGE` capabilities in this connection. If the `KEY_EXCHANGE` or `PSK_EXCHANGE` request will exceed the

maximum number of simultaneous active sessions of the Responder, the Responder shall respond with an `Errorcode = SessionLimitExceeded`.

750 This specification does not prohibit concurrent sessions in which the same Requester and Responder reverses role. 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 their portion of the session IDs are unique at 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.

## 751 **11.5 Records and session ID**

---

752 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 or both encrypted or authenticated. 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.

753 The actual format and other details of a record is outside the scope of this specification. It is generally assumed that the transport protocol will define the format and other details of the record.

## 754 12 Key schedule

755 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 [RFC5869](#) describes. SPDM defines the following additional functions:

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

756 where

- `BinConcat` shall be the concatenation of binary data, in the order that [Table 87 — BinConcat details](#) shows:

757 **Table 87 — 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	<a href="#">Hash byte order</a>	Hash.Length

758 If `Context` is null, `BinConcat` is the concatenation of the first three components only.

759 [Table 88 — Version details](#) describes the version details.

760 **Table 88 — Version details**

SPDM version	Version text
SPDM 1.1	"spdm1.1"
SPDM 1.2	"spdm1.2"

761 The `HKDF-Expand` function prototype, as used by the default SPDM key schedule, is as follows:

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

762 The `HKDF-Extract` function prototype is described as follows:

```
HKDF-Extract(salt, IKM);
```

763 where

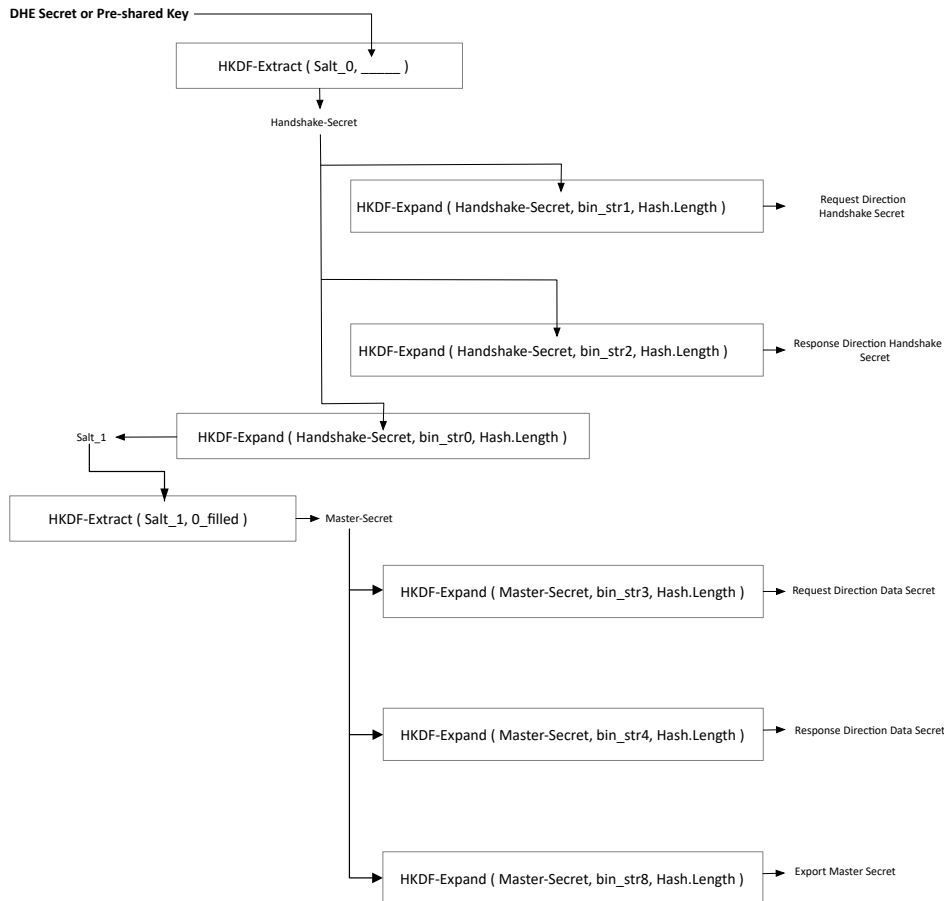
- IKM is the Input Keying Material.

764 For HKDF-Expand and HKDF-Extract, the hash function shall be the selected hash function in ALGORITHMS response. Hash.Length shall be the length of the output of the hash function selected by the ALGORITHMS response.

765 Both Responder and Requester shall use the key schedule that Figure 27 — Key schedule shows.

766 **Figure 27 — Key schedule**

767



768 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 a name for clarity.

769 Table 89 — Key schedule accompanies the figure to complete the Key Schedule. The Responder and Requester shall also adhere to the definition of this table.

770 **Table 89 — Key schedule**

Variable	Definition
Salt_0	A zero-filled array of Hash.Length length.
0_filled	A zero-filled array of Hash.Length length.
bin_str0	BinConcat(Hash.Length, Version, "derived", NULL)
bin_str1	BinConcat(Hash.Length, Version, "req hs data", TH1)
bin_str2	BinConcat(Hash.Length, Version, "rsp hs data", TH1)
bin_str3	BinConcat(Hash.Length, Version, "req app data", TH2)
bin_str4	BinConcat(Hash.Length, Version, "rsp app data", TH2)
DHE Secret	This shall be the secret derived from KEY_EXCHANGE/KEY_EXCHANGE_RSP
Pre-shared Key	PSK

771 **Note:** With common hash functions, any label longer than 12 characters requires an additional iteration of the hash function to compute. As in [RFC8446](#), the previously defined labels have all been chosen to fit within this limit.

## 772 12.1 DHE secret computation

773 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.

774 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 [RFC8446](#) defines.

775 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 to the same secret value.  $K_A$  and  $K_B$  are the results of a KDF. This specification shall use the KDF as defined by the [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 `BashHashSel` or `ExtHashSel`.

## 776 12.2 Transcript hash in key derivation

777 The key schedule uses two transcript hashes:

- TH1
- TH2



## 778 12.3 TH1 definition

---

779 If the Requester and Responder used `KEY_EXCHANGE / KEY_EXCHANGE_RSP` to exchange initial keying information, then **TH1** shall be the output of applying the negotiated hash function to the concatenation of the following:

1. `VCA`
2. 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.
3. `[KEY_EXCHANGE] . *`
4. `[KEY_EXCHANGE_RSP] . *` except the `ResponderVerifyData` field

780 If the Requester and Responder used `PSK_EXCHANGE / PSK_EXCHANGE_RSP` to exchange initial keying information, then **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 the `ResponderVerifyData` field

## 781 12.4 TH2 definition

---

782 If the Requester and Responder used `KEY_EXCHANGE / KEY_EXCHANGE_RSP` to exchange initial keying information, then **TH2** shall be the output of applying the negotiated hash function to the concatenation of the following:

1. `VCA`
2. 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.
3. `[KEY_EXCHANGE] . *`
4. `[KEY_EXCHANGE_RSP] . *`
5. 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)
6. `[FINISH] . *`
7. `[FINISH_RSP] . *`

783 If the Requester and Responder used `PSK_EXCHANGE / PSK_EXCHANGE_RSP` to exchange initial keying information, then **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)

## 784 12.5 Key schedule major secrets

---

785 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 ( $S_3$ )

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

### 787 12.5.1 Request-direction handshake secret

788 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`.

### 789 12.5.2 Response-direction handshake secret

790 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`.

### 791 12.5.3 Request-direction data secret

792 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.

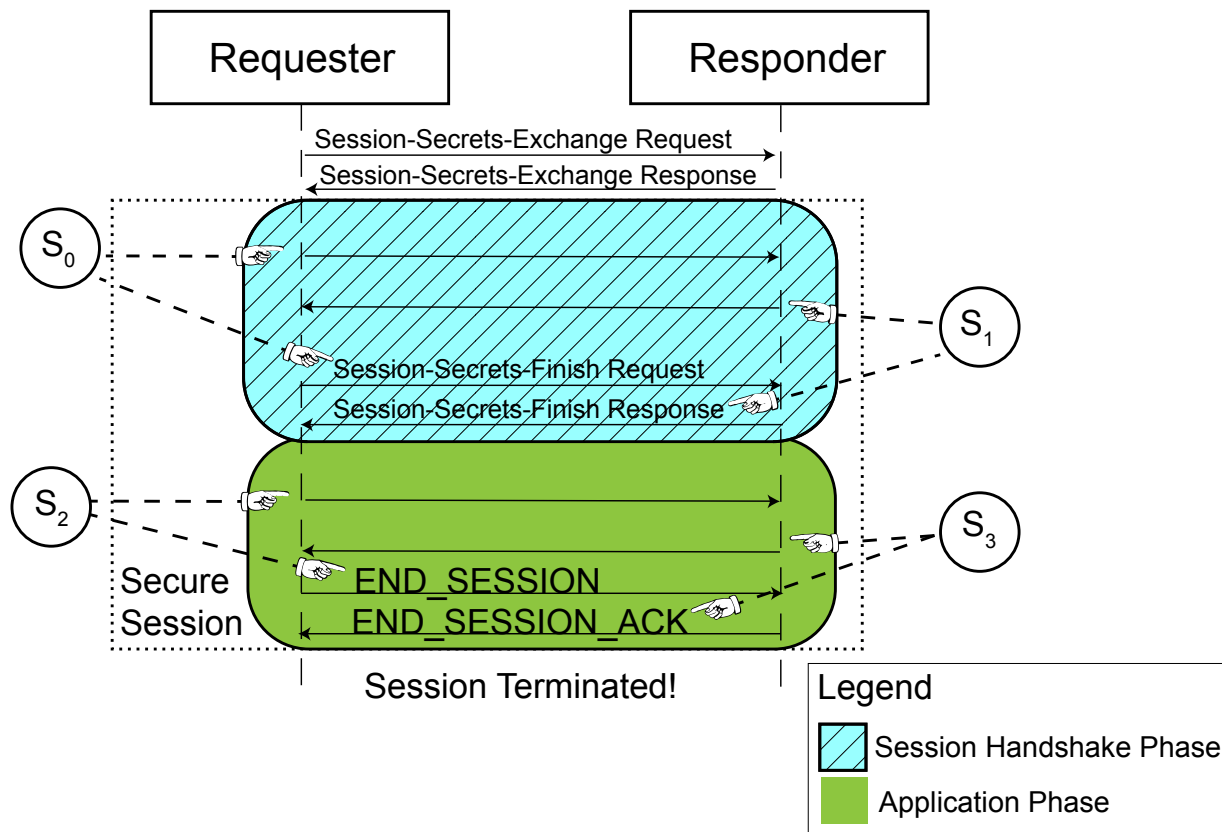
### 793 12.5.4 Response-direction data secret

794 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.

795 [Figure 28 — Secrets usage](#) illustrates where each of the major secrets are used, as described previously.

796 **Figure 28 — Secrets usage**

797



798 **12.6 Encryption key and IV derivation**

799 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);
    
```

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

801 **12.7 finished\_key derivation**

802 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);
```

803 The handshake-secret shall either be request-direction handshake secret or response-direction handshake secret.

## 804 12.8 Deriving additional keys from the Export Master Secret

---

805 After a successful SPDM key exchange, additional keys can be derived from the Export Master Secret. How keys are derived 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);
```

## 806 12.9 Major secrets update

---

807 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.

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

```
new_secret = HKDF-Expand(current_secret, bin_str9, Hash.Length);  
bin_str9 = BinConcat(Hash.Length, Version, "traffic upd", NULL);
```

809 In computing the new secret, `current_secret` shall either be the current Request-Direction Data Secret or Response-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.

## 810 13 Application data

---

811 SPDM utilizes authenticated encryption with associated data (AEAD) cipher algorithms in much the same way that TLS 1.3 does to protect both the confidentiality and integrity of data that shall remain secret, as well as the integrity of data that need to be transmitted in the clear, such as protocol headers, but shall be protected from manipulation. AEAD algorithms provide both encryption and message authentication. Each algorithm specifies the details such as the size of the nonce, the position and length of the MAC and many other factors to ensure a strong cryptographic algorithm.

812 AEAD functions shall provide the following functions and comply with the requirements defined in [RFC5116](#):

```
AEAD_Encrypt(encryption_key, nonce, associated_data, plaintext);
AEAD_Decrypt(encryption_key, nonce, associated_data, ciphertext);
```

813 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.

### 814 13.1 Nonce derivation

---

815 Certain AEAD ciphers have specific requirements on 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 [RFC5116](#) for nonce derivation.

## 816 14 General opaque data format

817 The general opaque data format allows for a mixture of vendors, standard organizations or transport-specific data to accompany an SPDM message without namespace collisions.

818 If the `OpaqueDataFmt1` bit is selected in `OtherParamsSelection` of `ALGORITHMS`, then all opaque data fields in SPDM messages shall use the format that [Table 90 — General opaque data format](#) defines.

819 **Table 90 — 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 <a href="#">Table 91 — Opaque element</a> .

820 [Table 91 — Opaque element](#) defines the format for each element in `OpaqueList`.

821 **Table 91 — Opaque element**

Byte offset	Field	Size (bytes)	Description
0	ID	1	Shall be one of the values in the <code>ID</code> column of <a href="#">Table 47 — Registry or standards body ID</a> .
1	VendorIDLen	1	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 <code>VendorID</code> column of <a href="#">Table 47 — Registry or standards body ID</a> for the respective <code>ID</code> .
2	VendorID	VendorIDLen	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.

Byte offset	Field	Size (bytes)	Description
$2 + \text{VendorIDLen}$	OpaqueElementDataLen	2	Shall be the length of OpaqueElementData .
$4 + \text{VendorIDLen}$	OpaqueElementData	OpaqueElementDataLen	Shall be the data defined by the vendor or standards body.
$4 + \text{VendorIDLen} + \text{OpaqueElementDataLen}$	AlignPadding	AlignPaddingSize = 0, 1, 2, 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 $4 + \text{VendorIDLen} + \text{OpaqueElementDataLen} + \text{AlignPaddingSize}$ 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.

## 822 15 Signature generation

823 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 `ALGORITHMS` message.

824 The signature generation function takes this form:

```
signature = SPDMsign(PrivKey, data_to_be_signed, context);
```

825 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

826 The function shall output a signature using `PrivKey` and a selected cryptographic signing algorithm.

827 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, then `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`.

828 Here is an example, named Example 1:

829 If the version of this specification is 1.4.3, the Responder is generating a signature and `context` is “my example context”. 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”.

830 Next, form `combined_spdm_prefix`. The `combined_spdm_prefix` shall be the concatenation of `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 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.

831 Continuing Example 1, [Table 92 — 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 the ASCII value of that number is used. See [Text or string encoding](#) for encoding rules.

832 **Table 92 — 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												

833 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.

834 If the Responder is generating the signature, the selected cryptographic signing algorithm is indicated in exactly one of `BaseAsymSel` or `ExtAsymSel` in `ALGORITHMS` message. If the Requester is generating the signature, the selected cryptographic signing algorithm is indicated in `ReqBaseAsymAlg` of `RespAlgStruct` in `ALGORITHMS` message.

835 Because each cryptographic signing algorithm is vastly different, these clauses define the binding of `SPDMsign` to those algorithms.

## 836 15.1 Signing algorithms in extensions

837 If an algorithm is selected in either the `ExtAsymSel` or `AlgExternal` of `ReqBaseAsymAlg` of `RespAlgStruct` in `ALGORITHMS` response, its binding is out of scope of this specification.

## 838 15.2 RSA and ECDSA signing algorithms

839 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`.

840 The private key, defined by the specification for these algorithms, shall be `PrivKey`.

841 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`.

842 RSA and ECDSA algorithms are described in [Signature algorithm references](#).

## 843 15.3 EdDSA signing algorithms

---

844 These algorithms are described in [RFC8032](#).

845 The private key, defined by RFC8032, shall be `PrivKey` .

846 In the specification for these algorithms, the letter `M` denotes the message to be signed.

### 847 15.3.1 Ed25519 sign

848 This specification only defines Ed25519 usage and not its variants.

849 `M` shall be the concatenation of `combined_spdm_prefix` and `message_hash` .

### 850 15.3.2 Ed448 sign

851 This specification only defines Ed448 usage and not its variants.

852 `M` shall be the concatenation of `combined_spdm_prefix` and `message_hash` .

853 Ed448 defines a context string, `C` . `C` shall be the `spdm_context` .

## 854 15.4 SM2 signing algorithm

---

855 This algorithm is described in [GB/T 32918.2-2016](#). GB/T 32918.2-2016 also defines the variable `M` and `IDA`.

856 The private key, defined by GB/T 32918.2-2016, shall be `PrivKey` .

857 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` .

858 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` .

859 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.

## 860 15.5 Signature algorithm references

---

861 These clauses provide basic information about each asymmetric algorithms SPDM supports as [Table 93 — 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 the [Signature byte order](#).

862 **Table 93 — SPDM Asymmetric Signature Reference Information**

---

Algorithm Name	Key Size (bits)	References
TPM_ALG_RSASSA_2048	2048	Section 8.2 of <a href="#">IETF RFC 8017</a>
TPM_ALG_RSASSA_3072	3072	Section 8.2 of <a href="#">IETF RFC 8017</a>
TPM_ALG_RSASSA_4096	4096	Section 8.2 of <a href="#">IETF RFC 8017</a>
TPM_ALG_RSAPSS_2048	2048	Section 8.1 of <a href="#">IETF RFC 8017</a>
TPM_ALG_RSAPSS_3072	3072	Section 8.1 of <a href="#">IETF RFC 8017</a>
TPM_ALG_RSAPSS_4096	4096	Section 8.1 of <a href="#">IETF RFC 8017</a>
TPM_ALG_ECDSA_ECC_NIST_P256	256	Section 6 of <a href="#">FIPS PUB 186-4</a> using TPM_ECC_NIST_P256 curve parameters as <a href="#">TCG Algorithm Registry</a> defines.
TPM_ALG_ECDSA_ECC_NIST_P384	384	Section 6 of <a href="#">FIPS PUB 186-4</a> using TPM_ECC_NIST_P384 curve parameters as <a href="#">TCG Algorithm Registry</a> defines.
TPM_ALG_ECDSA_ECC_NIST_P521	521	Section 6 of <a href="#">FIPS PUB 186-4</a> using TPM_ECC_NIST_P521 curve parameters as <a href="#">TCG Algorithm Registry</a> defines.
TPM_ALG_SM2_ECC_SM2_P256	256	Section 6 of <a href="#">GB/T 32918.2-2016</a> using TPM_ECC_SM2_P256 curve parameters as defined in <a href="#">TCG Algorithm Registry</a> .
EdDSA ed25519	256	<a href="#">IETF RFC8032</a>
EdDSA ed448	456	<a href="#">IETF RFC8032</a>

## 863 16 Signature verification

---

864 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 `ALGORITHMS` message.

865 The signature verification function takes this form:

```
SPDMsignatureVerify(PubKey, signature, unverified_data, context);
```

866 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

867 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.

868 The verifier of the signature shall create `spdm_prefix`, `spdm_context` and `combined_spdm_context` as described in [Signature generation](#).

869 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`.

870 If the Responder generated the signature, the selected cryptographic signature verification algorithm is indicated in exactly one of `BaseAsymSel` or `ExtAsymSel` in `ALGORITHMS` message. If the Requester generated the signature, the selected cryptographic signature verification algorithm is indicated in `ReqBaseAsymAlg` of `RespAlgStruct` in `ALGORITHMS` message.

871 Because each cryptographic signature verification algorithm is vastly different, these clauses define the binding of `SPDMsignatureVerify` to those algorithms.

### 872 16.1 Signature verification algorithms in extensions

---

873 If an algorithm is selected in either the `ExtAsymSel` or `AlgExternal` of `ReqBaseAsymAlg` of `RespAlgStruct` in `ALGORITHMS` response, its binding is out of scope of this specification.

## 874 16.2 RSA and ECDSA signature verification algorithms

---

- 875 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`.
- 876 The public key, defined in the specification for these algorithms, shall be `PubKey`.
- 877 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`.
- 878 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.
- 879 For ECDSA algorithms, `SPDMSignatureVerify` shall return success when the output of "Verification with the Public Key" as defined in [ANSI X9.62-2005](#) is "valid". Otherwise, `SPDMSignatureVerify` shall return failure.
- 880 RSA and ECDSA algorithms are described in [Signature algorithm references](#).

## 881 16.3 EdDSA signature verification algorithms

---

- 882 [RFC8032](#) describes these algorithms. RFC8032, also, defines the `M`, `PH`, and `C` variables.
- 883 The public key, also defined in RFC8032, shall be `PubKey`.
- 884 In the specification for these algorithms, the letter `M` denotes the message to be signed.

### 885 16.3.1 Ed25519 verify

- 886 `M` shall be the concatenation of `combined_spdm_prefix` and `unverified_message_hash`.
- 887 `SPDMSignatureVerify` shall return success when step 1 does not result in an invalid signature and the constraints of the group equation in step 3 are met as described in [RFC8032](#) section 5.1.7. Otherwise, `SPDMSignatureVerify` shall return failure.

### 888 16.3.2 Ed448 verify

- 889 `M` shall be the concatenation of `combined_spdm_prefix` and `unverified_message_hash`.
- 890 Ed448 defines a context string, `C`. `C` shall be the `spdm_context`.
- 891 `SPDMSignatureVerify` shall return success when step 1 does not result in an invalid signature and the constraints of the group equation in step 3 are met as described in [RFC8032](#) section 5.2.7. Otherwise, `SPDMSignatureVerify` shall return failure.

## 892 **16.4 SM2 signature verification algorithm**

---

893 This algorithm is described in [GB/T 32918.2-2016](#), which also defines the variable `M` and `IDA`.

894 The public key, also defined in GB/T 32918.2-2016, shall be `PubKey` .

895 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` .

896 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` .

897 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`.

898 `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.

## 899 **17 General ordering rules**

---

- 900 These general ordering rules apply to SPDM messages that form a transcript that eventually gets signed.
- 901 When requests are received out of order, the Responder can silently discard all requests or return an `ERROR` message with `ErrorCode=RequestResynch` (with the exception of `GET_VERSION` ) until the Requester issues a `GET_VERSION` . Out of order requests shall nullify the transcript.
- 902 A Requester can retry messages. The retries shall be identical to the first, excluding transport variances. However, if the Responder sees two or more non-identical `GET_CAPABILITIES` or `NEGOTIATE_ALGORITHMS` , the Responder shall return an `ERROR` message with `ErrorCode=UnexpectedRequest` or silently discard non-identical `GET_CAPABILITIES` or `NEGOTIATE_ALGORITHMS` requests. Because a retried message is identical to the first, a retried message shall not be used in transcript hash calculations.
- 903 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.

## 904 **18 ANNEX A (informative) TLS 1.3**

---

905 This specification heavily models TLS 1.3. TLS 1.3 and consequently this specification assumes the transport layers provide these capabilities or attributes:

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

906 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](#), which at the time of this specification is still a draft.



## 907 19 ANNEX B (informative) Device certificate example

908 [Device certificate example](#) shows an example device certificate:

### 909 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-----

```
MIIC4jCCAoigAwIBAgIBCDABggqhkJOPQDAjBcMQswCQYDVQQGEwJDQTELMaKGA1UECAwCTkxDTALBgNVBAcMBGNpdHkxDTALBgNVBAoMBEFDTUUxFTATBgNVBAsMDEFDTUUgRGV2aWNLczELMAKGA1UEAwwCQ0EwIBcNNzAwMTAxMDAwMDAwWhgPOTk5OTEyMzEyMzU5NTlzMh0xZCZAJBgNVBAYTALVMTMQswCQYDVQQIDAJQZEMCAGA1UECgwZQUNNRSBXaWRnZXQgTWfudWZhY3R1cmLuZzEnMCUGA1UECwweQUNNRSBXaWRnZXQgTWfudWZhY3R1cmLuZyBvbm0MRQwEgYDVQQDDAt3MDEyMzU5NTUj40TCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALpnR3J42iiB2YGb24gD4RckkbhI7WtwP0yiaKk7X3j8rkrRHGN2VKhAMSZ//z7gv5VcSrRvEVbKyBFTI+Edonq18CLYsvtD2t29UmvmpT8P02C4dNtWCNnuoDBKAYEe7mCt5AB6bmsyHCh+n0jDVNtj/R/RiCe74CIAF8l289DRsYfUBL/mCOE0IhHXVG0EWJvDyh3pyA083QngnCNllsbuxDn1WL1N0u6IE48yRiyzUtYcKuiVPYv7S9Ikr5azFxeq0qdY0j4hX3ADS9qCHTRL+hePbc1ph3SpgSZ05BDZjXhdBCoLztJBVFhB8YIARzcqF+eMJeoGGz5sSxW6GcCAwEAAnNMEswCQYDVR0TBAlwADALBgNVHQ8EBAMCBeAwMQYDVR0RBCowKKAmBgorBgEEAYMCGhIBoBgMFkFDUU6V0LER0VU0jAxMjM0NTY3ODkwCgYIKoZIZj0EAWIDSAARQIGhLqm7Vy2K/WeIiic78eq2xyHg0jBUMslBKvJbnz1awECIQDaSNRjpwVcLIP8BQBmSj48Mtjty6H28hjWGwhkXpoLQ==
```

-----END CERTIFICATE-----

## 910 20 ANNEX C (informative) OID reference

911 [Table 94 — Object identifiers \(OIDs\)](#) lists all object identifiers (OIDs) that this specification defines:

912 **Table 94 — 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	<a href="#">SPDM certificate requirements and recommendations</a>	Certificate device information.
{ id-DMTF-spdm 2 }	id-DMTF-hardware-identity	<a href="#">Identity provisioning</a>	Hardware certificate identifier.
{ id-DMTF-spdm 3 }	id-DMTF-eku-responder-auth	<a href="#">Extended Key Usage authentication OIDs</a>	Certificate Extended Key Usage - SPDM Responder Authentication.
{ id-DMTF-spdm 4 }	id-DMTF-eku-requester-auth	<a href="#">Extended Key Usage authentication OIDs</a>	Certificate Extended Key Usage - SPDM Requester Authentication.
{ id-DMTF-spdm 5 }	id-DMTF-mutable-certificate	<a href="#">Identity provisioning</a>	Mutable certificate identifier.
{ id-DMTF-spdm 6 }	id-DMTF-SPDM-extension	<a href="#">SPDM Non-Critical Certificate OID</a>	To contain other OIDs in a certificate extension.

## 913 21 ANNEX D (informative) variable name reference

914 Throughout this document, various sizes and offsets are referred to by a variable. [Table 95 — 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.

915 **Table 95 — Variables**

Symbol	Definition	Set location
A	Number of Requester-supported extended asymmetric key signature algorithms.	<a href="#">Table 13 — NEGOTIATE_ALGORITHMS request message format</a>
A'	Number of extended asymmetric key signature algorithms selected by the Requester.	<a href="#">Table 19 — Successful ALGORITHMS response message format</a>
D	The size of D (and C for ECDHE) is derived from the selected DHE group.	See the <code>KEY_EXCHANGE</code> request message format in <a href="#">Table 56 — KEY_EXCHANGE request message format</a> .
E	Number of Requester-supported extended hashing algorithms.	<a href="#">Table 13 — NEGOTIATE_ALGORITHMS request message format</a>
E'	The number of extended hashing algorithms selected requested by the Requester.	<a href="#">Table 19 — Successful ALGORITHMS response message format</a>
H	The output size, in bytes, of the hash algorithm agreed upon in <code>NEGOTIATE_ALGORITHMS</code> .	<a href="#">Table 19 — Successful ALGORITHMS response message format</a>
MS	The length of the cryptographic hash or raw bit stream, as indicated in <code>DMTFSpecMeasurementValueType[7]</code> .	<a href="#">Table 43 — DMTF measurement specification format</a>
NL	The length of the Nonce field in the <code>GET_MEASUREMENTS</code> request and the <code>MEASUREMENTS</code> response.	<a href="#">GET_MEASUREMENTS request attributes</a>
n	Number of version entries in the <code>VERSION</code> response message.	<a href="#">Table 9 — Successful VERSION response message format</a>
Q	Length of the <code>ResponderContext</code> .	<a href="#">Table 62 — PSK_EXCHANGE_RSP response message format</a>
P	Length of the <code>PSKHint</code> .	<a href="#">Table 61 — PSK_EXCHANGE request message format</a>
R	Length of the <code>RequesterContext</code> .	<a href="#">Table 61 — PSK_EXCHANGE request message format</a>
<code>SigLen</code>	The size of the asymmetric-signing algorithm output, in bytes, that the Responder selected through the last <code>ALGORITHMS</code> response message.	<a href="#">Table 19 — Successful ALGORITHMS response message format</a>

## 916 22 ANNEX E (informative) change log

---

### 917 22.1 Version 1.0.0 (2019-10-16)

---

- Initial Release

### 918 22.2 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 are 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.

### 919 22.3 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 occurs 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 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 43 — 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 backwards 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 `DeviceCerts` 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.

## 920 22.4 Version 1.2.1 (2022-05-11)

---

- Fix minor typographical errors.
- Correct indication that Identity Provisioning OIDs are in the certificate Extended Key Usage, and add SPDM Non-Critical Certificate Extension OID to [Table 32 — Optional fields](#).
- Added [Signature Algorithm References](#) clauses to clarify basic information about asymmetric algorithms.
- Added recommended `ErrorCode` for the case when the Responder detects overlapping `SET_CERTIFICATE` commands.
- Clarified rules around when the old key can be discarded during `KEY_UPDATE`.
- Clarified `Offset` and `Length` fields in `GET_CERTIFICATE` message.
- Clarified `DataTransferSize` and `MaxSPDMmsgSize` in `GET_CAPABILITIES` and `CAPABILITIES` messages.
- 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.
- Clarified measurement specification related fields in `NEGOTIATE_ALGORITHMS` and `ALGORITHMS`.
- Allow the sender to utilize the [Large SPDM message transfer mechanism](#) when the transmit buffer size of the sender that is less than the `DataBufferSize` of the receiving SPDM endpoint.
- Clarified measurement method for various timing parameters in [Timing specification table](#).
- Clarified the definition of Session-Secret-Exchange and removed the duplicate definition of it.
- Clarified that `VERSION` is not to be chunked.
- Edit Figure 22 so that a Secure Session does not encompass Session-Secrets-Exchange.
- Clarified retries from the perspective Responder and Requester in [Timing requirements](#).
- Clarified effects on out of order message on the transcript and other clarifications in [General ordering rules](#).
- Added [Signature byte order](#) and [Octet string byte order](#) clauses.
- Clarified the use of 'OtherParamsSelection'.
- Clarified that `CSRdata` references RFC2986 `CertificationRequest`.
- Clarified DER encoding for `RequesterInfo`
- Clarified the use of `OtherParamsSelection`.
- Clarified conditions for `LargeResponse` error.
- Added `SET_CERT_CAP`, `CSR_CAP` and `CERT_INSTALL_RESET_CAP` capabilities bits.
- Clarified measurement signing capabilities in `SignatureRequested` field of `GET_MEASUREMENTS`.
- Clarify that Responder [Timing measurements](#) are measured under the assumption that the Responder can access the bus.
- Clarified that `AlgCount` field in Algorithm request and response structures shall be a value of 2.



- Clarified the process to handle a `GET_CSR` request after a reset.
- Added `SET_CERTIFICATE` to the list of commands in [Table 7 — Timing specification for SPDM messages](#) that include a cryptographic operation.
- Clarify how retried messages affect transcript hash in [General ordering rules](#).
- Required root certificate to always be included in `SET_CERTIFICATE`.
- Correct [Figure 1 — SPDM certificate chain models](#) to show AliasCert model.
- Clarify that if a `GET_CSR` interrupts processing of another `GET_CSR`, then the existing request is dropped and the new request is processed.
- Clarified updating keys in `KEY_UPDATE`.
- Clarified that extended algorithms are external to this specification.
- Allowed `GET_DIGESTS` and `GET_CERTIFICATE` in session. Updated Table 6.
- Changed instances of “secure environment” to “trusted environment”.
- Clarified (A1, B4, C1) message flow is permitted.
- Added [Table 93 — SPDM Asymmetric Signature Reference Information](#)
- Added definition of [opaque data](#).
- Changed “cancel” to “invalidate state and data associated with” in `GET_VERSION` and `VERSION` response messages.
- Fixed typo in the `ExchangeData` field of table “Successful `KEY_EXCHANGE_RSP` response message format”.
- Clarified the `Request ID` for the first message in an optimized encapsulated request flow.
- Clarified Session-Secrets-Exchange in Optimized encapsulated request flow
- Fixed typo in Table 77.
- Fixed OID value for id-DMTF-device-info to match previous releases.
- Fixed typos and removed redundant grammar in Table 41.
- Updated BinConcat Version to 1.2 in Table 90.
- Clarified definition of `DecryptError`.
- Renamed “VendorLen” to “VendorIDLen”.
- Punctuation and grammar changes as needed.
- Clarified slots 1-7 certificate provisioning.
- Clarified exclusion of signature in `CHALLENGE_AUTH` and usage of concatenation in [Table 36](#)
- Corrected the signing algorithm in the `FINISH` request’s Signature field.
- Changed ANNEX B from “normative” to “informative”.
- Clarified restrictions on Bit 0 through 2 of the `MutAuthRequested` field of `KEY_EXCHANGE_RSP`.
- Removed potentially confusing statements on Slot provisioning for `GET_CSR`.
- Fixed typo in Table 79.

## 921 **22.5 Version 1.2.2 (2023-10-08)**

- Correct values in Field and Size columns of Table 51.

- Update [Table 7 — Timing specification for SPDM messages](#) to clarify that Responders can exceed `ST1` and `CT` using `ErrorCode=ResponseNotReady`.
- Make the layout of tables 52 and 53 match the layout of the other table in the specification.
- Clarified the presence of the `SlotIDParam` field in `GET_MEASUREMENTS`.
- Clarified that if endpoint does not support chunking then it must set `MaxSPDMmsgSize` equal to `DataTransferSize`.
- Replaced wording of “internal buffer” in `GET_CERTIFICATE` with `DataTransferSize` and “transmit buffer”.
- Added `LargeResponse` error to description of chunking certificates.
- Clarified requirements for chunking the `CERTIFICATE` response.
- Clarified `VCA` for the case where capabilities and algorithms were provisioned alongside PSK.
- Removed normative error statement from the `BasicMutAuthReq` field of `CHALLENGE_AUTH`.
- Added the VESA standards body to [Registry or standards body ID](#).
- Replaced links to ITU-T X.509 with RFC5280 and removed ITU-T X.509 from the Normative references section.
- Removed “after Reset” from M1/M2 ordering.
- Changed “or” to “and” in Large SPDM message transfer mechanism section.
- Added size of the transmit buffer as a condition for `CHUNK_SEND`.
- Renamed instances of OEM to vendor-defined.
- Removed normative text that prohibited reuse of session IDs.
- Deprecated the `CHAL_CAP` capability of the Requester.
- Specified Responder’s response to invalid measurement index.
- Changed endianness of the Context field of `BinConcat` from little to hash byte order.
- Renamed the `HMAC-Hash` to `HKDF-Extract`.
- Expanded `ResponseTooLarge` to include the transmit buffer size and `DataTransferSize`.
- Clarified that size of `ResponseToLargeRequest` may be smaller than `DataTransferSize`.
- Added clause that sizes and lengths are in units of bytes.
- Added evaluation of the Responder’s transmit buffer size to `LargeResponse`.
- Clarified that the Negotiated State Preservation Indicator applies to the cached Negotiated State.
- Clarified that `LargeResponse` shall not re-initialize L1/L2 to null.
- Changed statement about graceful error handling during chunking from normative to non-normative.
- Clarified that SPDM messages sent outside of a session do not contribute to in-session transcripts.
- Added missing `MaxSPDMmsgSize` to `GET_CAPABILITIES` request and `CAPABILITIES` response messages.
- Clarified that Responders can alter requested CSR fields.
- Changed “should” to “shall” in the `LargeMessageSize` field of `CHUNK_SEND`.
- Clarified that `KEY_EX_CAP` only applies to Requester’s request message and Responder’s response message.
- Changed instances of “Requester-direction” to “Request-direction” and instances of “Responder-direction” to “Response-direction”.
- Removed encapsulation requirements from `MUT_AUTH_CAP` definition.
- Clarified that start of the Heartbeat timer can include `PSK_EXCHANGE_RSP`.

- Removed deprecation status from ENCAP\_CAP.
- Clarified that `WTMax` includes the wait time from the most recently received `ResponseNotReady`.
- Clarified that if either Requester or Responder do not support Heartbeat then the value of `HeartbeatPeriod` would be `0`.
- Removed informative statement that chunks are equal in size.
- Clarified that non-encapsulated requests are prohibited during the session handshake phase.
- Clarified the assumption that version entries are not duplicated when calculating `MinDataTransferSize`.
- Clarified that `MeasurementHashAlgo` should be zero if `MeasurementSpecificationSel` is zero.
- Clarified runtime measurement change detection (Param2 of Table 43).
- Clarified that some devices may need to persist `GET_CSR` related information across resets.
- Corrected `PK` to `PubKey` in [CHALLENGE\\_AUTH signature verification](#).
- 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`.
- Clarify the definition of errata versions and that they may contain behavioral changes to fix security issues or defects.
- Clarified that `ERROR` is only allowed in response to `GET_VERSION` in cases explicitly defined in this specification.
- The Responder shall return error `ResponseTooLarge` and shall not discard the request that caused this error.
- Add missing `ffdhe3072` in DHE secret computation section.
- Clarify allowed session phases for `GET_CSR`, `SET_CERTIFICATE`, `GET_DIGESTS`, and `GET_CERTIFICATE` in [Table 6 — SPDM request and response messages validity](#).
- Clarified [Table 51 — Standard body or vendor-defined header \(SVH\)](#) when the payload is standards body or registry organization defined.
- Move statement that DMTF does not define extended algorithms to [Table 25 — Extended Algorithm field format](#).
- Added normative information in [Table 11 — Flag fields definitions for the Requester](#) and [Table 12 — Flag fields definitions for the Responder](#).
- Clarify the device behavior when a reset is required for a pending previous `SET_CERTIFICATE` request.
- Clarified in [Table 47 — Registry or standards body ID](#) that the registry specifies the value used for the `VendorID` field.
- Renamed “Negotiated State Preservation Indicator” field of `END_SESSION` request to “Negotiated State Clearing Indicator”
- Clarified sessions can be established one at a time when `HANDSHAKE_IN_THE_CLEAR_CAP` is set.
- Clarify that `CERTIFICATE` response shall not return a partial certificate chain in case of chunking enabled and the Requester asking for a complete certificate chain.
- New:
  - Added [custom environments](#) clauses.
  - Added [VendorDefinedReqPayload and VendorDefinedRespPayload defined by DMTF specifications](#) clauses.
- Remove text that `ENCRYPT_CAP` and `MAC_CAP` apply to all phases of a secure session.
- Clarified that the minimum number of supported sessions shall be one per connection.

- Clarified the chunked transfer including transcript update and interruption of the chunk transfer sequence.
- Removed text that prohibited error response codes for `GET_CAPABILITIES` and `NEGOTIATE_ALGORITHMS`.
- Added explanation as to how the `RDT` value is measured at the Responder.
- Clarified the definition of RDT as the additional time needed by the responder and not as a delay.
- Clarified Responder's support for retry.
- Clarified that `SlotID` field value in `SET_CERTIFICATE_RSP` Response shall be the same as `SlotID` value in `SET_CERTIFICATE` Request.

## 922 **23 Bibliography**

---

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