



1

Document Identifier: DSP0274

2

Date: 2024-08-21

3

Version: 1.1.4

4

# Security Protocol and Data Model (SPDM) Specification

5

Supersedes: 1.1.3

6

Document Class: Normative

7

Document Status: Published

8

Document Language: en-US

Copyright Notice

Copyright © 2019–2024 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patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a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	6
1.1 Acknowledgments	6
2 Introduction	7
2.1 Conventions	7
2.1.1 Document conventions	7
2.1.2 Reserved and unassigned values	7
2.1.3 Byte ordering	7
2.1.4 Sizes and lengths	7
2.1.5 SPDM data types	7
2.1.6 Version encoding	8
2.1.7 Notations	8
2.1.8 Other conventions	9
3 Scope	10
4 Normative references	11
5 Terms and definitions	13
6 Symbols and abbreviated terms	17
7 SPDM message exchanges	18
7.1 Security capability discovery and negotiation	18
7.2 Identity authentication	18
7.2.1 Identity provisioning	19
7.2.2 Runtime authentication	19
7.3 Firmware and configuration measurement	19
7.4 Secure sessions	19
7.5 Mutual authentication overview	20
8 SPDM messaging protocol	21
8.1 SPDM bits-to-bytes mapping	23
8.2 Generic SPDM message format	23
8.3 SPDM request codes	24
8.4 SPDM response codes	26
8.5 SPDM request and response code issuance allowance	27
8.6 Concurrent SPDM message processing	28
8.7 Requirements for Requesters	28
8.8 Requirements for Responders	29
9 Timing requirements	30
9.1 Timing measurements	30
9.2 Timing specification table	30
10 SPDM messages	35
10.1 Capability discovery and negotiation	35
10.2 GET_VERSION request and VERSION response messages	35
10.3 GET_CAPABILITIES request and CAPABILITIES response messages	38
10.4 NEGOTIATE_ALGORITHMS request and ALGORITHMS response messages	45

10.4.1 Behavior after VERSION, CAPABILITIES and ALGORITHMS . . . . .	65
10.5 Responder identity authentication . . . . .	66
10.6 Requester identity authentication . . . . .	68
10.6.1 Certificates and certificate chains . . . . .	68
10.7 GET_DIGESTS request and DIGESTS response messages . . . . .	69
10.8 GET_CERTIFICATE request and CERTIFICATE response messages. . . . .	70
10.8.1 Mutual authentication requirements for GET_CERTIFICATE and CERTIFICATE messages. 73	
10.8.2 Leaf certificate . . . . .	73
10.9 CHALLENGE request and CHALLENGE_AUTH response messages . . . . .	75
10.9.1 CHALLENGE_AUTH signature generation . . . . .	78
10.9.2 CHALLENGE_AUTH signature verification . . . . .	80
10.9.3 Basic mutual authentication . . . . .	82
10.10 Firmware and other measurements. . . . .	85
10.11 GET_MEASUREMENTS request and MEASUREMENTS response messages. . . . .	86
10.11.1 Measurement block . . . . .	90
10.11.2 MEASUREMENTS signature generation . . . . .	92
10.11.3 MEASUREMENTS signature verification . . . . .	95
10.12 ERROR response message . . . . .	96
10.13 RESPOND_IF_READY request message format . . . . .	102
10.14 VENDOR_DEFINED_REQUEST request message . . . . .	103
10.15 VENDOR_DEFINED_RESPONSE response message . . . . .	105
10.15.1 VendorDefinedReqPayload and VendorDefinedRespPayload defined by DMTF specifications . . . . .	106
10.16 KEY_EXCHANGE request and KEY_EXCHANGE_RSP response messages . . . . .	106
10.16.1 Mutual authentication . . . . .	115
10.16.2 Specifying Requester certificate for mutual authentication . . . . .	115
10.17 FINISH request and FINISH_RSP response messages . . . . .	116
10.17.1 Transcript hash calculation rules. . . . .	119
10.18 PSK_EXCHANGE request and PSK_EXCHANGE_RSP response messages . . . . .	122
10.19 PSK_FINISH request and PSK_FINISH_RSP response messages . . . . .	128
10.20 HEARTBEAT request and HEARTBEAT_ACK response messages. . . . .	130
10.20.1 Heartbeat additional information . . . . .	131
10.21 KEY_UPDATE request and KEY_UPDATE_ACK response messages . . . . .	131
10.21.1 Session key update synchronization. . . . .	132
10.21.2 KEY_UPDATE transport allowances. . . . .	135
10.22 GET_ENCAPSULATED_REQUEST request and ENCAPSULATED_REQUEST response messages. . . . .	138
10.22.1 Encapsulated request flow . . . . .	138
10.22.2 Optimized encapsulated request flow . . . . .	138
10.22.3 Triggering GET_ENCAPSULATED_REQUEST . . . . .	142
10.22.4 Additional constraints . . . . .	142
10.23 DELIVER_ENCAPSULATED_RESPONSE request and ENCAPSULATED_RESPONSE_ACK response messages. . . . .	142

10.23.1 Additional information . . . . .	145
10.24 END_SESSION request and END_SESSION_ACK response messages . . . . .	146
11 Session. . . . .	148
11.1 Session handshake phase. . . . .	148
11.2 Application phase. . . . .	149
11.3 Session termination phase. . . . .	149
11.4 Simultaneous active sessions . . . . .	149
11.5 Records and session ID. . . . .	150
12 Key schedule . . . . .	151
12.1 Transcript hash in key derivation . . . . .	153
12.2 TH1 definition . . . . .	153
12.3 TH2 definition . . . . .	154
12.4 Key schedule major secrets . . . . .	155
12.4.1 Request-direction handshake secret. . . . .	155
12.4.2 Response-direction handshake secret . . . . .	155
12.4.3 Requester-direction data secret . . . . .	155
12.4.4 Responder-direction data secret . . . . .	155
12.5 Encryption key and IV derivation . . . . .	156
12.6 finished_key derivation . . . . .	156
12.7 Deriving additional keys from the Export Master Secret . . . . .	157
12.8 Major secrets update . . . . .	157
13 Application data . . . . .	158
13.1 Nonce derivation . . . . .	158
14 ANNEX A (informative) TLS 1.3. . . . .	159
15 ANNEX B (normative) Leaf certificate example. . . . .	160
16 ANNEX C (informative) Change log. . . . .	162
16.1 Version 1.0.0 (2019-10-16) . . . . .	162
16.2 Version 1.1.0 (2020-07-15) . . . . .	162
16.3 Version 1.1.1 (2021-05-12) . . . . .	162
16.4 Version 1.1.2 (2022-03-09) . . . . .	163
16.5 Version 1.1.3 (2023-10-08) . . . . .	163
16.6 Version 1.1.4 (2024-08-19) . . . . .	164
17 Bibliography . . . . .	165

# 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 **1.1 Acknowledgments**

---

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

### 18 **Contributors:**

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

## 19 2 Introduction

---

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

### 21 2.1 Conventions

---

22 The following conventions apply to all SPDM specifications.

#### 23 2.1.1 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.

#### 24 2.1.2 Reserved and unassigned values

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

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

#### 27 2.1.3 Byte ordering

28 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).

#### 29 2.1.4 Sizes and lengths

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

#### 31 2.1.5 SPDM data types

32 The [SPDM data types](#) table lists the abbreviations and descriptions for common data types that SPDM message fields and data structure definitions use. These definitions follow [DSP0240](#).

33 **SPDM data types**

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

34 **2.1.6 Version encoding**

35 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.

36 EXAMPLE:

37 Version 3.7 → 0x37

38 Version 1.0 → 0x10

39 Version 1.2 → 0x12

40 An [endpoint](#) that supports Version 1.2 can interoperate with an older endpoint that supports Version 1.0 only, but the available functionality is limited to what SPDM specification Version 1.0 defines.

41 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`.

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

43 **2.1.7 Notations**

44 SPDM specifications use the following notations:



Notation	Description
M:N	<p>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> (<math>M \leq N</math>).</p> <p>The lowest offset is on the left. The highest offset is on the right.</p>
[4]	<p>Square brackets around a number typically indicate a bit offset.</p> <p>Bit offsets are zero-based values. That is, the least significant bit (<code>[LSb]</code>) offset = 0.</p>
[M:N]	<p>A range of bit offsets where M is greater than or equal to N.</p> <p>The most significant bit is on the left, and the least significant bit is on the right.</p>
1b	<p>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.</p>
0x12A	<p>Hexadecimal, indicated by the leading <code>0x</code> .</p>
N+	<p>Variable-length byte range that starts at byte offset N.</p>
{ Payload }	<p>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.</p>
{ Payload }::[[S <sub>x</sub> ]]	<p>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 X.</p> <p>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> .</p>

**45 2.1.8 Other conventions**

46 Unless otherwise specified, all figures are informative.

## 47 **3 Scope**

---

48 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.

49 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.

## 50 4 Normative references

---

51 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 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 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 01.27*, February 7, 2018
- 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
- **ASN.1 — ISO-822-1-4, DER — ISO-8825-1**
  - [ITU-T X.680, X.681, X.682, X.683, X.690](#), 08/2015

- **X.509 — ISO-9594-8**
  - [ITU-T X.509](#), 10/2012
- **ECDSA**
  - Section 6, The Elliptic Curve Digital Signature Algorithm (ECDSA) in [FIPS PUB 186-4 Digital Signature Standard \(DSS\)](#)
  - Appendix D: Recommended Elliptic Curves for Federal Government Use in [FIPS PUB 186-4 Digital Signature Standard \(DSS\)](#)
- **RSA**
  - Table 3 in [TCG Algorithm Registry Family “2.0” Level 00 Revision 01.22](#), February 9, 2015
- **SHA2-256, SHA2-384, and SHA2-512**
  - [FIPS PUB 180-4 Secure Hash Standard \(SHS\)](#)
- **SHA3-256, SHA3-384, and SHA3-512**
  - [FIPS PUB 202 SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions](#)
- **Transport Layer Security 1.3**
  - [TLS 1.3 RFC 8446](#)

## 52 5 Terms and definitions

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

54 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.

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

56 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.

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

58 This specification uses these terms:

Term	Definition
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	Process of determining whether an entity is who or what it claims to be.
authentication initiator	Endpoint that initiates the authentication process by challenging another endpoint.
byte	Eight-bit quantity. Also known as an <i>octet</i> .
certificate	Digital form of identification that provides information about an entity and certifies ownership of a particular asymmetric key-pair.
certificate authority (CA)	Trusted entity that issues certificates.
certificate chain	Series of two or more certificates. Each certificate is signed by the preceding certificate in the chain.
component	Physical entity similar to the PCI Express specification's definition.
device	Physical entity such as a network controller or a fan.

Term	Definition
DMTF	Formerly known as the Distributed Management Task Force, DMTF creates open manageability standards that span diverse emerging and traditional information technology (IT) infrastructures, including cloud, virtualization, network, servers, and storage. Member companies and alliance partners worldwide collaborate on standards to improve the interoperable management of IT.
endpoint	Logical entity that communicates with other endpoints over one or more transport protocols.
intermediate certificate	Certificate that is neither a root certificate nor a leaf certificate.
leaf certificate	Last certificate in a certificate chain.
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 originator	Original transmitter, or source, of an SPDM message.
message transcript	The concatenation of a sequence of messages in the order in which they are sent and received by an endpoint. The final message included in the message transcript may be truncated to allow inclusion of a signature in that message which is computed over the message transcript. 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 <i>byte</i> in a number consisting of multiple bytes.
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.
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 of the scope of this specification. The semantics and usage of this data are implementation specific and also outside of the scope of this specification.

Term	Definition
payload	Information-bearing fields of a message. These fields are separate from the fields and elements, such as address fields, framing bits, checksums, and so on, that transport the message from one point to another. In some instances, a field can be both a payload field and a transport field.
physical transport binding	Specifications that define how a base messaging protocol is implemented on a particular physical transport type and medium, such as SMBus/I <sup>2</sup> C, PCI Express™ Vendor Defined Messaging, and so on.
record	A record is 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.
Responder	Ultimate receiver, or destination, of an SPDM request message. It is also the original transmitter, or source of an SPDM response message.
root certificate	First certificate in a certificate chain, which is self-signed.
<a href="#">Security Protocols and Data Models (SPDM)</a>	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	Session Keys are any secrets, derived cryptographic keys, or any cryptographic information bound to the session.
Session-Secrets-Exchange	This term denotes 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> .
secure session	A secure session is a session that provides either or both of encryption or message authentication for communicating data over a transport.
SPDM message	Unit of communication in SPDM communications.
SPDM message payload	Portion of the message body of an SPDM message. This portion of the message is separate from those fields and elements that identify the SPDM version, the SPDM request and response codes, and the two parameters.
SPDM request message	Message that is sent to an endpoint to request a specific SPDM operation. A corresponding SPDM response message acknowledges receipt of an SPDM request message.
SPDM response message	Message that is sent in response to a specific SPDM request message. This message includes a <code>Response Code</code> field that indicates whether the request completed normally.

Term	Definition
trusted computing base (TCB)	<p>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.</p> <p>Reference: <a href="https://en.wikipedia.org/wiki/Trusted_computing_base">https://en.wikipedia.org/wiki/Trusted_computing_base</a></p>



## 59 6 Symbols and abbreviated terms

---

60 The abbreviations defined in [DSP0004](#), [DSP0223](#), and [DSP1001](#) apply to this document.

61 The following additional abbreviations are used in this document.

Abbreviation	Definition
AEAD	Authenticated Encryption with Associated Data
CA	<i>certificate authority</i>
DMTF	Formerly the <i>Distributed Management Task Force</i>
MAC	Message Authentication Code
MSB	<i>most significant byte</i>
SPDM	<i>Security Protocol and Data Model</i>
TCB	<i>trusted computing base</i>

## 62 7 SPDM message exchanges

---

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

64 The specification-defined message exchanges enable Requesters to:

- Discover and negotiate the security capabilities of a Responder.
- Authenticate the 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.

65 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.

### 66 7.1 Security capability discovery and negotiation

---

67 This specification defines a mechanism for a Requester to discover the security capabilities of a Responder. For example, an endpoint could support multiple cryptographic hash functions that are defined in this specification. Furthermore, the specification defines a mechanism for a Requester and Responder to select a common set of cryptographic algorithms to use for all subsequent message exchanges before another negotiation is initiated by the Requester, if an overlapping set of cryptographic algorithms exists that both endpoints support.

### 68 7.2 Identity authentication

---

69 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.

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

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

## 71 **7.2.1 Identity provisioning**

72 Identity provisioning is the process that device vendors follow during or after hardware manufacturing. A trusted root *certificate authority (CA)* generates a *root certificate* ( `RootCert` ) that is provisioned to the *authentication initiator*. The authentication initiator uses this certificate to verify the validity of certificate chains. A device carries a *certificate chain*, which has the `RootCert` as the root of the certificate chain and a device certificate ( `DeviceCert` ) as the *leaf certificate* of the certificate chain. The device certificate contains the public key that corresponds to the device private key.

73 Through the certificate chain, the root CA indirectly endorses the per-device public/private key pair in the `DeviceCert` , where the private key is provisioned to or generated by the endpoint.

74 Alternatively to certificate chains, the vendor may 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.

75 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](#).

## 76 **7.2.2 Runtime authentication**

77 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 to sign the challenge. The authentication initiator verifies the signature by using the public key of the Responder, and any intermediate public keys within the certificate chain by using the root certificate as the trusted anchor.

78 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.

## 79 **7.3 Firmware and configuration measurement**

---

80 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.

## 81 **7.4 Secure sessions**

---

82 Many devices exchange data with other devices that may 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. This protocol of the application data usually allows for encrypted and/or authenticated data transfer.

83 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](#).

84 Lastly, but not least, many SPDM requests and their corresponding responses can also be afforded the same protection. See the [SPDM request and response messages validity](#) table and [SPDM request and response code issuance allowance](#) clause for more details.

85 The [SPDM messaging protocol flow](#) gives a very high-level view of when the secure session actually starts.

## 86 **7.5 Mutual authentication overview**

---

87 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 authentication](#) discusses identity in regard to the Responder but the details apply to the Requester as well.

88 In general, when this specification places 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 will provide more details.

## 89 **8 SPDM messaging protocol**

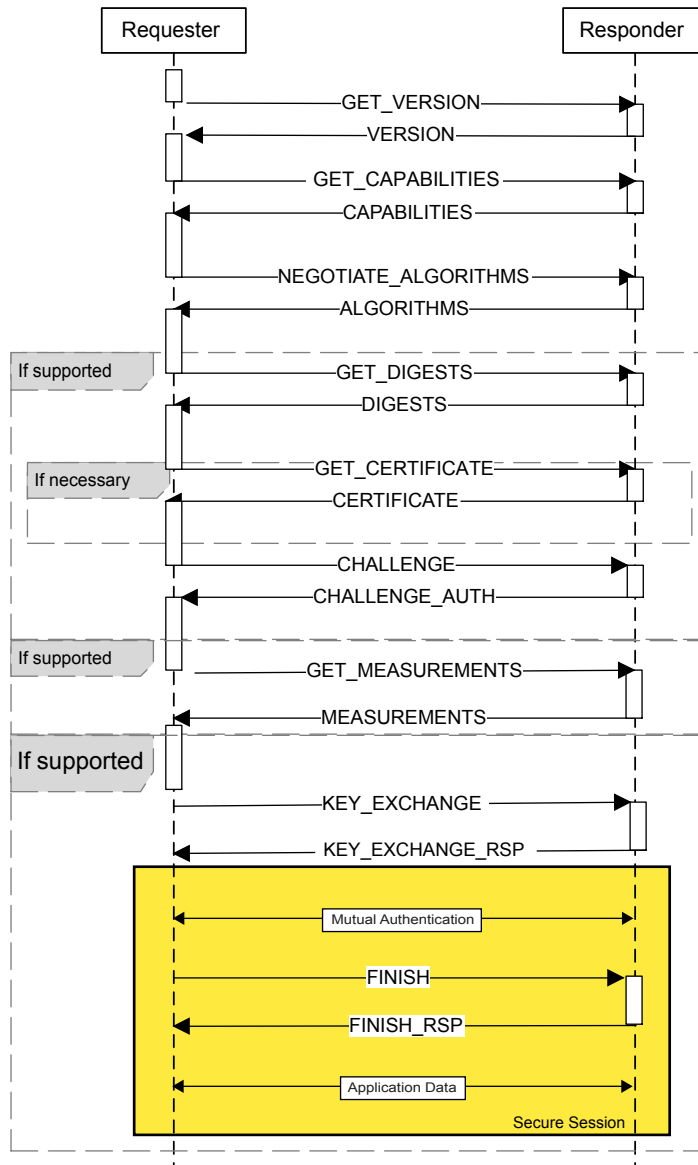
---

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

91 The [SPDM messaging protocol flow](#) is an example of a high-level request-response flow diagram for SPDM. An endpoint that acts as the *Requester* sends an SPDM request message to another endpoint that acts as the *Responder*, and the Responder returns an SPDM response message to the Requester.

92 **SPDM messaging protocol flow**

93



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

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

96 **8.1 SPDM bits-to-bytes mapping**

97 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.

98 **One-byte field bit map**

99 Example: A One-Byte Field

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

100 **Two-byte field bit map**

101 Example: A Two-Byte Field

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

102 **8.2 Generic SPDM message format**

103 The [Generic SPDM message field definitions](#) table defines the fields that constitute a generic SPDM message, including the message header and payload.

104 **Generic SPDM message field definitions**

Byte	Bits	Length (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> .

Byte	Bits	Length (bits)	Field	Description
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> .
1	[7:0]	8	Request Response Code	The request message code or response code, which <a href="#">Table 4</a> and <a href="#">Table 5</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	SPDM message payload	Zero or more bytes that are specific to the Request Response Code .

## 105 8.3 SPDM request codes

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

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

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



## 109 SPDM request codes

Request	Code value	Implementation requirement	Message format
GET_DIGESTS	0x81	Optional	GET_DIGESTS request message format
GET_CERTIFICATE	0x82	Optional	GET_CERTIFICATE request message format
CHALLENGE	0x83	Optional	CHALLENGE request message format
GET_VERSION	0x84	Required	GET_VERSION request message format
GET_MEASUREMENTS	0xE0	Optional	GET_MEASUREMENTS request message format
GET_CAPABILITIES	0xE1	Required	GET_CAPABILITIES request message format
NEGOTIATE_ALGORITHMS	0xE3	Required	NEGOTIATE_ALGORITHMS request message format
KEY_EXCHANGE	0xE4	Optional	KEY_EXCHANGE request message format
FINISH	0xE5	Optional	FINISH request message format
PSK_EXCHANGE	0xE6	Optional	PSK_EXCHANGE request message format
PSK_FINISH	0xE7	Optional	PSK_FINISH request message format
HEARTBEAT	0xE8	Optional	HEARTBEAT request message format
KEY_UPDATE	0xE9	Optional	KEY_UPDATE request message format
GET_ENCAPSULATED_REQUEST	0xEA	Optional	GET_ENCAPSULATED_REQUEST request message format
DELIVER_ENCAPSULATED_RESPONSE	0xEB	Optional	DELIVER_ENCAPSULATED_RESPONSE request message format
END_SESSION	0xEC	Optional	END_SESSION request message format
RESPOND_IF_READY	0xFF	Required	RESPOND_IF_READY request message format
VENDOR_DEFINED_REQUEST	0xFE	Optional	VENDOR_DEFINED_REQUEST request message format
Reserved	0x80 , 0x85 - 0xDF , 0xE2 , 0xED - 0xFD		SPDM implementations compatible with this version shall not use the reserved request codes.

## 110 8.4 SPDM response codes

111 The `Request Response Code` field in the SPDM response message shall specify the appropriate response code for a request. All SPDM-compatible implementations shall use the following [SPDM response codes](#).

112 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.

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

### 114 SPDM response codes

Response	Value	Implementation requirement	Message format
DIGESTS	0x01	Optional	Successful DIGESTS response message format
CERTIFICATE	0x02	Optional	Successful CERTIFICATE response message format
CHALLENGE_AUTH	0x03	Optional	Successful CHALLENGE_AUTH response message format
VERSION	0x04	Required	Successful VERSION response message format
MEASUREMENTS	0x60	optional	Successful MEASUREMENTS response message format
CAPABILITIES	0x61	Required	Successful CAPABILITIES response message format
ALGORITHMS	0x63	Required	Successful ALGORITHMS response message format
KEY_EXCHANGE_RSP	0x64	Optional	Successful KEY_EXCHANGE_RSP response message format
FINISH_RSP	0x65	Optional	Successful FINISH_RSP response message format
PSK_EXCHANGE_RSP	0x66	Optional	PSK_EXCHANGE_RSP response message format
PSK_FINISH_RSP	0x67	Optional	Successful PSK_FINISH_RSP response message format
HEARTBEAT_ACK	0x68	Optional	HEARTBEAT_ACK response message format

Response	Value	Implementation requirement	Message format
KEY_UPDATE_ACK	0x69	Optional	KEY_UPDATE_ACK response message format
ENCAPSULATED_REQUEST	0x6A	Optional	ENCAPSULATED_REQUEST response message format
ENCAPSULATED_RESPONSE_ACK	0x6B	Optional	ENCAPSULATED_RESPONSE_ACK response message format
END_SESSION_ACK	0x6C	Optional	END_SESSION_ACK response message format
VENDOR_DEFINED_RESPONSE	0x7E	Optional	VENDOR_DEFINED_RESPONSE response message format
ERROR	0x7F		ERROR response message format
Reserved	0x00 , 0x05 - 0x5F , 0x62 , 0x6D - 0x7D	SPDM implementations compatible with this version shall not use the reserved response codes.	

## 115 8.5 SPDM request and response code issuance allowance

- 116 The [SPDM request and response messages validity](#) table describes the conditions under which a request and response can be issued.
- 117 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.
- 118 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.
- 119 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 may have additional restrictions. See the respective request and response clause for further details.
- 120 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 of the SPDM specification. The session ID is implicit from the session used to transmit the commands.

121 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.

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

### 123 **SPDM request and response messages validity**

Request	Response	Session	Outside of a session	Session phases
GET_MEASUREMENT	MEASUREMENT	Allowed	Allowed	Application Phase
FINISH	FINISH_RSP	Allowed	Conditional (*)	Session Handshake
PSK_FINISH	PSK_FINISH_RSP	Allowed	Allowed	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
All others	All others	Prohibited	Allowed	Not Applicable

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

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

## 126 **8.6 Concurrent SPDM message processing**

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

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

## 129 **8.7 Requirements for Requesters**

130 A Requester shall not have multiple outstanding requests to the same Responder, with the following exception: as addressed in [GET\\_VERSION request and VERSION response messages](#), a Requester may issue a `GET_VERSION` to

a Responder to restart the protocol due to an internal error or reset, even if the Requester has existing outstanding requests to the same Responder.

131 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.

132 A Requester may send simultaneous request messages to different Responders.

## 133 8.8 Requirements for Responders

---

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

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

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

137 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.

## 138 9 Timing requirements

---

139 The [Timing specification for SPDM messages](#) table shows the timing specifications for Requesters and Responders.

140 If the Requester does not receive a response within **T1** or **T2** time accordingly, the Requester may retry a request message. A retry of a request message shall be a complete retransmission of the original SPDM request message. Because a retried message is identical to the first, a retried message shall not be used in transcript hash calculations.

141 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.

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

### 143 9.1 Timing measurements

---

144 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 **RD<sub>T</sub>**, 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.

### 145 9.2 Timing specification table

---

146 The **Ownership** column in the [Timing specification for SPDM messages](#) table specifies whether the timing parameter applies to the Responder or Requester.

147 **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.</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, such as the <a href="#">GET_CAPABILITIES</a> , <a href="#">GET_VERSION</a> , or <a href="#">NEGOTIATE_ALGORITHMS</a> request messages.</p>
T1	Requester	RTT+ST1	μs	<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 <a href="#">ST1</a> .</p>
CT	Requester and Responder	$2^{CTExponent}$	μs	<p><a href="#">CTExponent</a> is reported in <a href="#">GET_CAPABILITIES</a> and <a href="#">CAPABILITIES</a> messages.</p> <p>This timing parameter shall be the maximum amount of time the endpoint has to provide any response requiring cryptographic processing, such as the <a href="#">GET_MEASUREMENTS</a> or <a href="#">CHALLENGE</a> request messages.</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 CT .</p>
RDT	Responder	2 <sup>RDTExponent</sup>	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 RDTExponent as part of the ERROR response. See the ResponseNotReady extended error data table for the RDTExponent value.</p> <p>For details, see <code>ErrorCode=ResponseNotReady</code> in the ResponseNotReady extended error data table. An SPDM responder measures the RDT value from the end of the transmission of the ERROR message of <code>ErrorCode=ResponseNotReady</code> , to the reception of the next RESPOND_IF_READY 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.</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 RDT .</p>

Timing parameter	Ownership	Value	Units	Description
$WT_{Max}$	Requester	$(RDT * RDTM) - RTT$	$\mu s$	<p>Maximum wait time the Requester has to issue <a href="#">RESPOND_IF_READY</a> request unless the Requester issued a successful <a href="#">RESPOND_IF_READY</a> request message earlier.</p> <p>After this time the Responder is allowed to drop the response. The Requester shall take into account the transmission time of the <a href="#">ERROR</a> from the Responder to Requester when calculating <math>WT_{Max}</math>.</p> <p>The <math>RDTM</math> value appears in the <a href="#">ResponseNotReady extended error data</a>.</p> <p>The Responder should ensure that <math>WT_{Max}</math> does not result in less than <math>WT</math> in determination of <math>RDTM</math>.</p> <p>For details, see <a href="#">ErrorCode=ResponseNotReady</a> in the <a href="#">ResponseNotReady extended error data</a> table.</p>
$HeartbeatPeriod$	Requester and Responder	Variable	s	<p>See <a href="#">HEARTBEAT request and HEARTBEAT_ACK response</a> for detail.</p>

## 148 10 SPDM messages

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

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

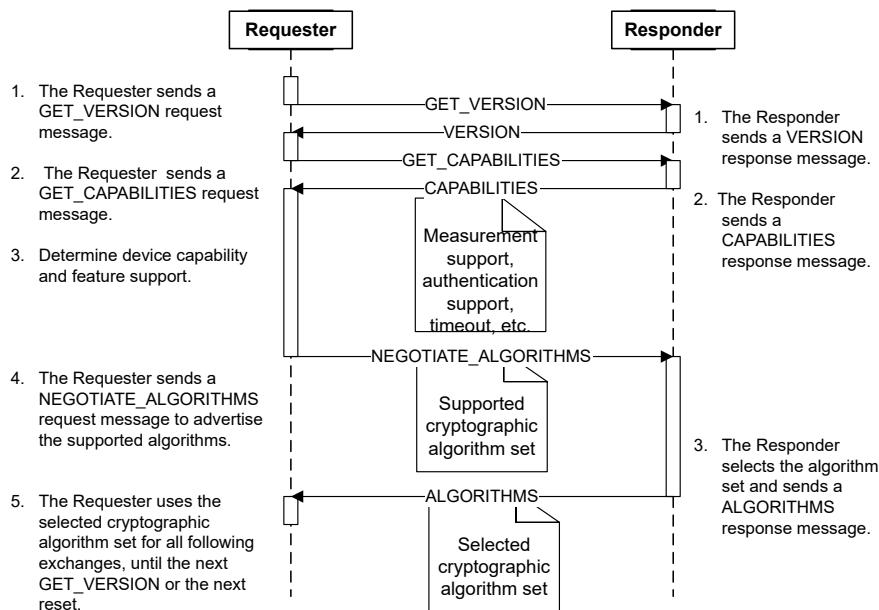
### 150 10.1 Capability discovery and negotiation

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

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

#### 153 Capability discovery and negotiation flow

154



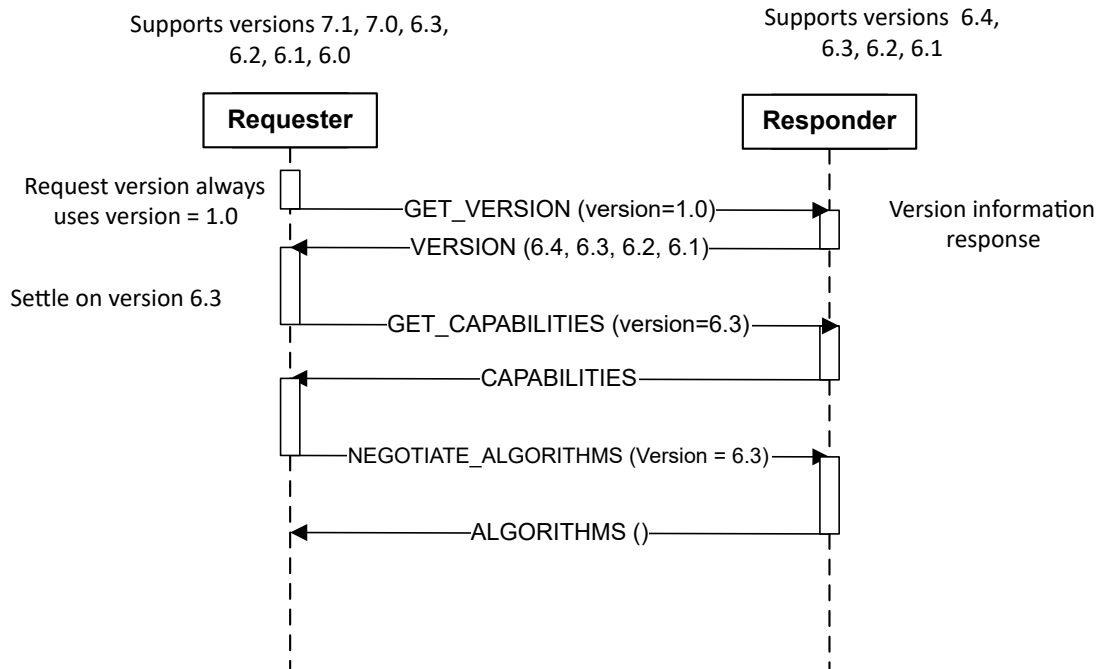
### 155 10.2 GET\_VERSION request and VERSION response messages

156 This request message shall retrieve the SPDM version of an endpoint. The [GET\\_VERSION request message format](#)

table shows the `GET_VERSION` request message format and the [Successful VERSION response message format](#) table shows the `VERSION` response message format.

- 157 In all future SPDM versions, the `GET_VERSION` and `VERSION` response messages will be backward compatible with all earlier versions.
- 158 The Requester shall begin the discovery process by sending a `GET_VERSION` request message with major version `0x1`. All Responders shall always support the `GET_VERSION` request message with major version `0x1` and provide a `VERSION` response containing all supported versions, as the [GET\\_VERSION request message format](#) table describes.
- 159 The Requester shall consult the `VERSION` response to select a common supported version, which is typically 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 an `ERROR` message except for `ErrorCode`s specified in this clause.
- 160 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.
- 161 After receiving a `GET_VERSION` request, the Responder shall cancel all previous requests from the same Requester. All active sessions between the Requester and the Responder are terminated, i.e., information (such as session keys, session IDs) for those sessions should not be used anymore. Additionally, this message shall clear or reset the previously [Negotiated State](#), if any, in both the Requester and its corresponding Responder.
- 162 All Responders that have completed a firmware update shall either respond with `ErrorCode=RequestResynch` to any request until a `GET_VERSION` request is received or silently discard the request.
- 163 **Discovering the common major version**

164



165 GET\_VERSION request message format

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

166 Successful VERSION response message format

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

Offset	Field	Size (bytes)	Value
5	VersionNumberEntryCount	1	Number of version entries present in this table (=n).
6	VersionNumberEntry1:<n>	2*n	16-bit version entry. See the <a href="#">VersionNumberEntry definition</a> table.

### 167 VersionNumberEntry definition

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

## 168 10.3 GET\_CAPABILITIES request and CAPABILITIES response messages

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

170 The [GET\\_CAPABILITIES request message format](#) table shows the GET\_CAPABILITIES request message format.

171 The [Successful CAPABILITIES response message format](#) table shows the CAPABILITIES response message format.

172 The [Requester flag fields definitions](#) table shows the flag fields definitions for the Requester.

173 Likewise, the [Responder flag fields definitions](#) table shows the flag fields definitions for the Responder.

### 174 GET\_CAPABILITIES request message format

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	<a href="#">RequestResponseCode</a>	1	0xE1=GET_CAPABILITIES

Offset	Field	Size (bytes)	Value
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 the <a href="#">Timing specification for SPDM messages</a> table.</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 the <a href="#">Requester flag fields definitions</a> table.

175 **Successful CAPABILITIES response message format**

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	<a href="#">RequestResponseCode</a>	1	0x61=CAPABILITIES
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 <math>CT</math>.</p> <p>See the <a href="#">Timing specification for SPDM messages</a> table.</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 the <a href="#">Responder flag fields definitions</a> table.

## 176 Requester flag fields definitions

177 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.

Byte	Bit	Field	Value
0	0	Reserved	Reserved.
0	1	<code>CERT_CAP</code>	If set, Requester shall support <code>DIGESTS</code> and <code>CERTIFICATE</code> response messages.
0	2	<code>CHAL_CAP</code>	If set, Requester shall support <code>CHALLENGE_AUTH</code> response message.
0	4:3	<code>MEAS_CAP</code>	The corresponding bits of the <a href="#">Responder flag fields definitions</a> indicate <code>MEASUREMENT</code> response capabilities. These bits shall be set to <code>00b</code> .
0	5	<code>MEAS_FRESH_CAP</code>	The corresponding bit of the <a href="#">Responder flag fields definitions</a> indicate <code>MEASUREMENT</code> response capabilities. This bit shall be set to <code>0b</code> .
0	6	<code>ENCRYPT_CAP</code>	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	<code>MAC_CAP</code>	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	<code>MUT_AUTH_CAP</code>	If set, Requester shall support mutual authentication.
1	1	<code>KEY_EX_CAP</code>	If set, Requester shall support <code>KEY_EXCHANGE</code> messages. If set, one or more of <code>ENCRYPT_CAP</code> and <code>MAC_CAP</code> shall be set.



Byte	Bit	Field	Value
1	3:2	PSK_CAP	<p>Pre-shared key capabilities of the Requester.</p> <p>00b . Requester shall not support pre-shared key capabilities.</p> <p>01b . Requester shall support pre-shared key</p> <p>10b and 11b . Reserved.</p> <p>If supported, one or more of ENCRYPT_CAP and MAC_CAP shall be set.</p>
1	4	ENCAP_CAP	<p>If set, Requester shall support GET_ENCAPSULATED_REQUEST , ENCAPSULATED_REQUEST , DELIVER_ENCAPSULATED_RESPONSE , and ENCAPSULATED_RESPONSE_ACK messages. If mutual authentication is supported, this field shall be set.</p>
1	5	HBEAT_CAP	<p>If set, Requester shall support HEARTBEAT messages.</p>
1	6	KEY_UPD_CAP	<p>If set, Requester shall support KEY_UPDATE messages.</p>
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 message authentication and/or encryption ( MAC_CAP and ENCRYPT_CAP ) are used in the handshake phase of a secure session.</p>

Byte	Bit	Field	Value
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 CERT_CAP of the Requester shall be 0.
2	7:1	Reserved	Reserved.
3	7:0	Reserved	Reserved.

### 178 Responder flag fields definitions

179 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.

Byte	Bit	Field	Value
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.

Byte	Bit	Field	Value
0	4:3	MEAS_CAP	<p>MEASUREMENT response capabilities of the Responder.</p> <p>00b . The Responder shall not support MEASUREMENTS response capabilities.</p> <p>01b . The Responder shall support MEASUREMENTS response but cannot perform signature generation.</p> <p>10b . The Responder shall support MEASUREMENTS response and can generate signatures.</p> <p>11b . Reserved.</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> <p>1 . The Responder shall support recomputing all MEASUREMENTS without requiring a reset or restart, and shall always return fresh MEASUREMENTS as part of MEASUREMENTS response message.</p>
0	6	ENCRYPT_CAP	<p>If set, Responder shall support message encryption in a secure session. If set, one or more of PSK_CAP or KEY_EX_CAP fields shall be specified accordingly to indicate support.</p>
0	7	MAC_CAP	<p>If set, Responder shall support message authentication in a secure session. If set, one or more of PSK_CAP or KEY_EX_CAP fields shall be specified 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.</p>
1	0	MUT_AUTH_CAP	<p>If set, Responder shall support mutual authentication.</p>
1	1	KEY_EX_CAP	<p>If set, Responder shall support KEY_EXCHANGE messages. If set, one or more of ENCRYPT_CAP and MAC_CAP shall be set.</p>

Byte	Bit	Field	Value
1	3:2	PSK_CAP	<p>Pre-Shared Key capabilities of the Responder.</p> <p>00b . Responder does not support Pre-Shared Key capabilities.</p> <p>01b . Responder shall support Pre-Shared Key but does not provide ResponderContext for session key derivation.</p> <p>10b . Responder shall support Pre-Shared Key and provides ResponderContext for session key derivation.</p> <p>11b . Reserved.</p> <p>If supported, one or more of ENCRYPT_CAP and 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. If mutual authentication is supported, this field shall be set.</p>
1	5	HBEAT_CAP	<p>If set, Responder shall support HEARTBEAT messages.</p>
1	6	KEY_UPD_CAP	<p>If set, Responder shall support KEY_UPDATE messages.</p>
1	7	HANDSHAKE_IN_THE_CLEAR_CAP	<p>If set, the Responder can only send and receive messages without encryption and message authentication during the Session Handshake Phase. If set, KEY_EX_CAP shall also be set. Setting this bit leads to changes in the contents of certain SPDM messages, 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 message authentication and/or encryption ( MAC_CAP and ENCRYPT_CAP ) are used in the handshake phase of a secure session.</p>

Byte	Bit	Field	Value
2	0	PUB_KEY_ID_CAP	If set, the public key of the Responder was provisioned to the Requester. The transport layer is responsible for identifying the Requester. In this case, CERT_CAP of the Responder shall be 0 .
2	7:1	Reserved	Reserved.
3	7:0	Reserved	Reserved.

180 **10.4 NEGOTIATE\_ALGORITHMS request and ALGORITHMS response messages**

181 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.

182 A Requester shall not issue any other SPDM requests, with the exception of GET\_VERSION until it receives a successful ALGORITHMS response message.

183 The NEGOTIATE\_ALGORITHMS request message format table shows the NEGOTIATE\_ALGORITHMS request message format.

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

185 **NEGOTIATE\_ALGORITHMS request message format**

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	RequestResponseCode	1	0xE3=NEGOTIATE_ALGORITHMS
2	Param1	1	Number of algorithms structure tables in this request using ReqAlgStruct
3	Param2	1	Reserved
4	Length	2	Length of the entire request message, in bytes. Length shall be less than or equal to 128 bytes.

Offset	Field	Size (bytes)	Value
6	MeasurementSpecification	1	<p>Bit mask. The measurement specification is used in the MEASUREMENTS response. Requester can set all available algorithms defined in the measurement specification format. The Requester can set zero bits if MEASUREMENTS are not supported.</p> <p>Bit 0: This bit shall indicate support for the DMTF-defined measurement specification. See <a href="#">DMTF specification for the Measurement field of a measurement block</a> clauses for details.</p>
7	Reserved	1	Reserved

Offset	Field	Size (bytes)	Value
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 S be the size of the signature in bytes. If the size of a signature component is less than specified size, then 0x00 octets are padded to the left of the most significant byte.</p> <p>Byte 0 Bit 0.  <a href="#">TPM_ALG_RSASSA_2048</a> where S=256.</p> <p>Byte 0 Bit 1.  <a href="#">TPM_ALG_RSAPSS_2048</a> where S=256.</p> <p>Byte 0 Bit 2.  <a href="#">TPM_ALG_RSASSA_3072</a> where S=384.</p> <p>Byte 0 Bit 3.  <a href="#">TPM_ALG_RSAPSS_3072</a> where S=384.</p> <p>Byte 0 Bit 4.  <a href="#">TPM_ALG_ECDSA_ECC_NIST_P256</a> where S=64 (32-byte r followed by 32-byte s).</p> <p>Byte 0 Bit 5.  <a href="#">TPM_ALG_RSASSA_4096</a> where S=512.</p> <p>Byte 0 Bit 6.  <a href="#">TPM_ALG_RSAPSS_4096</a> where S=512.</p> <p>Byte 0 Bit 7.  <a href="#">TPM_ALG_ECDSA_ECC_NIST_P384</a> where S=96 (48-byte r followed by 48-byte s).</p> <p>Byte 1 Bit 0.  <a href="#">TPM_ALG_ECDSA_ECC_NIST_P521</a> where S=132 (66-byte r followed by</p>

Offset	Field	Size (bytes)	Value
			66-byte s). All other values reserved.
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.  Byte 0 Bit 0. <a href="#">TPM_ALG_SHA_256</a>  Byte 0 Bit 1. <a href="#">TPM_ALG_SHA_384</a>  Byte 0 Bit 2. <a href="#">TPM_ALG_SHA_512</a>  Byte 0 Bit 3. <a href="#">TPM_ALG_SHA3_256</a>  Byte 0 Bit 4. <a href="#">TPM_ALG_SHA3_384</a>  Byte 0 Bit 5. <a href="#">TPM_ALG_SHA3_512</a>  All other values reserved.
16	Reserved	12	Reserved
28	ExtAsymCount	1	Number of Requester-supported extended asymmetric key signature algorithms (=A) for the purpose of signature verification. $A + E + \text{ExtAlgCount2} + \text{ExtAlgCount3} + \text{ExtAlgCount4} + \text{ExtAlgCount5}$ 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). $A + E + \text{ExtAlgCount2} + \text{ExtAlgCount3} + \text{ExtAlgCount4} + \text{ExtAlgCount5}$ 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



Offset	Field	Size (bytes)	Value
32	ExtAsym	4*A	List of Requester-supported extended asymmetric key signature algorithms for the purpose of signature verification. The <a href="#">Extended algorithm field format</a> table describes the format of this field.
32 + 4*A	ExtHash	4*E	List of the extended hashing algorithms supported by Requester. The <a href="#">Extended algorithm field format</a> table describes the format of this field.
32 + 4*A + 4*E	ReqAlgStruct	AlgStructSize	See the <a href="#">AlgStructure</a> request field.

186 `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.

187 **Algorithm request structure**

Offset	Field	Size (bytes)	Value
0	AlgType	1	Type of algorithm. [1:0] = Reserved 2 = DHE 3 = AEADCipherSuite 4 = ReqBaseAsymAlg 5 = KeySchedule All other values reserved.

Offset	Field	Size (bytes)	Value
1	AlgCount	1	<p>Requester supported fixed algorithms.</p> <p>Bit [7:4]. Number of Bytes required to describe Requester supported SPDM-enumerated fixed algorithms (= FixedAlgCount). FixedAlgCount + 2 shall be a multiple of 4</p> <p>Bit [3:0] Number of Requester supported extended algorithms (= ExtAlgCount).</p>
2	AlgSupported	FixedAlgCount	Bit mask listing Requester-supported SPDM-enumerated algorithms.
2 + FixedAlgCount	AlgExternal	4*ExtAlgCount	List of Requester-supported extended algorithms. The <a href="#">Extended algorithm field format</a> table describes the format of this field.

188 The following tables describe the associated fixed fields for the individual types.

189 **DHE structure**

Offset	Field	Size (bytes)	Value
0	AlgType	1	0x2=DHE
1	AlgCount	1	<p>Bit [7:4] = 2.</p> <p>Bit [3:0] = Number of Requester-supported extended DHE groups (= ExtAlgCount2).</p>

Offset	Field	Size (bytes)	Value
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> <p>Byte 0 Bit 0. <a href="#">ffdhe2048</a> (D = 256)</p> <p>Byte 0 Bit 1. <a href="#">ffdhe3072</a> (D = 384)</p> <p>Byte 0 Bit 2. <a href="#">ffdhe4096</a> (D = 512)</p> <p>Byte 0 Bit 3. <a href="#">secp256r1</a> (D = 64, C = 32)</p> <p>Byte 0 Bit 4. <a href="#">secp384r1</a> (D = 96, C = 48)</p> <p>Byte 0 Bit 5. <a href="#">secp521r1</a> (D = 132, C = 66)</p> <p>All other values reserved.</p>
4	AlgExternal	4*ExtAlgCount2	<p>List of Requester-supported extended DHE groups. The <a href="#">Extended algorithm field format</a> table describes the format of this field.</p>

190 **AEAD structure**

Offset	Field	Size (bytes)	Value
0	AlgType	1	0x3=AEAD
1	AlgCount	1	<p>Bit [7:4] = 2.</p> <p>Bit [3:0] = Number of Requester supported extended AEAD algorithms (= ExtAlgCount3).</p>

Offset	Field	Size (bytes)	Value
2	AlgSupported	2	<p>Bit mask listing Requester-supported SPDM-enumerated AEAD algorithms.</p> <p>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.</p> <p>Byte 0 Bit 1. <a href="#">AES-256-GCM</a>. 256-bit key; 96-bit IV; tag size is specified by transport layer.</p> <p>Byte 0 Bit 2. <a href="#">CHACHA20_POLY1305</a>. 256-bit key; 96-bit IV; 128-bit tag.</p> <p>All other values reserved.</p>
4	AlgExternal	4*ExtAlgCount3	<p>List of Requester-supported extended AEAD algorithms. The <a href="#">Extended algorithm field format</a> table describes the format of this field.</p>

191 ReqBaseAsymAlg structure

Offset	Field	Size (bytes)	Value
0	AlgType	1	0x4=ReqBaseAsymAlg
1	AlgCount	1	<p>Bit [7:4] = 2.</p> <p>Bit [3:0] = Number of Requester supported extended asymmetric key signature algorithms for the purpose of signature generation.(= ExtAlgCount4).</p>

Offset	Field	Size (bytes)	Value
2	AlgSupported	2	<p>Bit mask listing Requester-supported SPDM-enumerated asymmetric key signature algorithms for the purposes of signature generation.</p> <p>Byte 0 Bit 0. <a href="#">TPM_ALG_RSASSA_2048</a></p> <p>Byte 0 Bit 1. <a href="#">TPM_ALG_RSAPSS_2048</a></p> <p>Byte 0 Bit 2. <a href="#">TPM_ALG_RSASSA_3072</a></p> <p>Byte 0 Bit 3. <a href="#">TPM_ALG_RSAPSS_3072</a></p> <p>Byte 0 Bit 4. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P256</a></p> <p>Byte 0 Bit 5. <a href="#">TPM_ALG_RSASSA_4096</a></p> <p>Byte 0 Bit 6. <a href="#">TPM_ALG_RSAPSS_4096</a></p> <p>Byte 0 Bit 7. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P384</a></p> <p>Byte 1 Bit 0. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P521</a></p> <p>All other values reserved.</p>
4	AlgExternal	4*ExtAlgCount4	<p>List of Requester-supported extended asymmetric key signature algorithms for the purpose of signature generation. The <a href="#">Extended algorithm field format</a> table describes the format of this field.</p>

192 **KeySchedule structure**

Offset	Field	Size (bytes)	Value
0	AlgType	1	0x5=KeySchedule

Offset	Field	Size (bytes)	Value
1	AlgCount	1	Bit [7:4] = 2.  Bit [3:0] = Number of Requester supported extended key schedule algorithms (= ExtAlgCount5).
2	AlgSupported	2	Bit mask listing Requester-supported SPDM-enumerated Key Schedule algorithms.  Byte 0 Bit 0. SPDM Key Schedule.  All other values reserved.
4	AlgExternal	4*ExtAlgCount5	List of Requester-supported extended key schedule algorithms. The <a href="#">Extended algorithm field format</a> table describes the format of this field.

193 **Successful ALGORITHMS response message format**

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	RequestResponseCode	1	0x63=ALGORITHMS
2	Param1	1	Number of algorithms structure tables in this response using RespAlgStruct
3	Param2	1	Reserved
4	Length	2	Length of the response message, in bytes.

Offset	Field	Size (bytes)	Value
6	MeasurementSpecificationSel	1	Bit mask. The Responder shall select one of the measurement specifications supported by the Requester and Responder. Thus, no more than one bit shall be set. The MeasurementSpecification field in NEGOTIATE_ALGORITHMS defines the format of this field.
7	Reserved	1	Reserved

Offset	Field	Size (bytes)	Value
8	MeasurementHashAlgo	4	<p>Bit mask indicating the SPDM-enumerated hashing algorithm selected for measurements.</p> <p>Bit 0. Raw Bit Stream Only</p> <p>Bit 1. <a href="#">TPM_ALG_SHA_256</a></p> <p>Bit 2. <a href="#">TPM_ALG_SHA_384</a></p> <p>Bit 3. <a href="#">TPM_ALG_SHA_512</a></p> <p>Bit 4. <a href="#">TPM_ALG_SHA3_256</a></p> <p>Bit 5. <a href="#">TPM_ALG_SHA3_384</a></p> <p>Bit 6. <a href="#">TPM_ALG_SHA3_512</a></p> <p>If the Responder supports <code>GET_MEASUREMENTS</code>, exactly one bit in this bit field shall be set. Otherwise, the Responder shall set this field to <code>0</code>.</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>



Offset	Field	Size (bytes)	Value
12	BaseAsymSel	4	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.
16	BaseHashSel	4	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.
20	Reserved	12	Reserved
32	ExtAsymSelCount	1	Number of extended asymmetric key signature algorithms selected for the purpose of signature generation. Shall be either 0 or 1 (=A'). If the capabilities do not support this algorithm, this value is not used and shall be set to zero.
33	ExtHashSelCount	1	The number of extended hashing algorithms selected. Shall be either 0 or 1 (=E'). If the capabilities do not support this algorithm, this value is not used and shall be set to zero.
34	Reserved	2	Reserved.

Offset	Field	Size (bytes)	Value
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="#">Response AlgStructure field format</a>

194 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.

195 **Response AlgStructure field format**

Offset	Field	Size (bytes)	Value
0	AlgType	1	Type of algorithm. [1:0] = Reserved 2 = DHE 3 = AEADCipherSuite 4 = ReqBaseAsymAlg 5 = KeySchedule All other values reserved.
1	AlgCount	1	Bit mask listing Responder supported fixed algorithm requested by the Requester. Bit [7:4]. Number of Bytes required to describe Requester supported SPDM-enumerated fixed algorithms (= FixedAlgCount). FixedAlgCount + 2 shall be a multiple of 4 Bit [3:0] Number of Requester-supported, Responder-selected, extended algorithms (= ExtAlgCount'). This value shall be either 0 or 1.
2	AlgSupported	FixedAlgCount	Bit mask for indicating a Requester-supported, Responder-selected, SPDM-enumerated algorithm. Responder shall set at most one bit to 1.

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

196 The tables for each of the individual type with the associated fixed fields are described below.

197 **DHE structure**

Offset	Field	Size (bytes)	Value
0	AlgType	1	0x2=DHE
1	AlgCount	1	Bit [7:4] = 2.  Bit [3:0] = Number of Requester-supported, Responder-selected, extended DHE groups (= ExtAlgCount2'). This value shall be either 0 or 1.

Offset	Field	Size (bytes)	Value
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> <p>Byte 0 Bit 0. <a href="#">ffdhe2048</a> (D = 256)</p> <p>Byte 0 Bit 1. <a href="#">ffdhe3072</a> (D = 384)</p> <p>Byte 0 Bit 2. <a href="#">ffdhe4096</a> (D = 512)</p> <p>Byte 0 Bit 3. <a href="#">secp256r1</a> (D = 64, C = 32)</p> <p>Byte 0 Bit 4. <a href="#">secp384r1</a> (D = 96, C = 48)</p> <p>Byte 0 Bit 5. <a href="#">secp521r1</a> (D = 132, C = 66)</p> <p>All other values reserved.</p>
4	AlgExternal	4*ExtAlgCount <sup>2</sup>	<p>If present: a Requester-supported, Responder-selected, extended DHE algorithm. The <a href="#">Extended algorithm field format</a> table describes the format of this field.</p>

198 **AEAD structure**

Offset	Field	Size (bytes)	Value
0	AlgType	1	0x3=AEAD

Offset	Field	Size (bytes)	Value
1	AlgCount	1	<p>Bit [7:4] = 2.</p> <p>Bit [3:0] = Number of Requester-supported, Responder-selected, extended AEAD algorithms (= ExtAlgCount3'). This value shall be either 0 or 1.</p>
2	AlgSupported	2	<p>Bit mask for indicating a Requester-supported, Responder-selected, SPDM-enumerated AEAD algorithm.</p> <p>Byte 0 Bit 0. <a href="#">AES-128-GCM</a></p> <p>Byte 0 Bit 1. <a href="#">AES-256-GCM</a></p> <p>Byte 0 Bit 2. <a href="#">CHACHA20_POLY1305</a></p> <p>All other values reserved.</p>
4	AlgExternal	4*ExtAlgCount3'	<p>If present: a Requester-supported, Responder-selected, extended AEAD algorithm. The <a href="#">Extended algorithm field format</a> table describes the format of this field.</p>

199 ReqBaseAsymAlg structure

Offset	Field	Size (bytes)	Value
0	AlgType	1	0x4=ReqBaseAsymAlg
1	AlgCount	1	<p>Bit [7:4] = 2.</p> <p>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.</p>

Offset	Field	Size (bytes)	Value
2	AlgSupported	2	<p>Bit mask for indicating a Requester-supported, Responder-selected, SPDM-enumerated asymmetric key signature algorithm for the purposes of signature verification.</p> <p>Byte 0 Bit 0. <a href="#">TPM_ALG_RSASSA_2048</a></p> <p>Byte 0 Bit 1. <a href="#">TPM_ALG_RSAPSS_2048</a></p> <p>Byte 0 Bit 2. <a href="#">TPM_ALG_RSASSA_3072</a></p> <p>Byte 0 Bit 3. <a href="#">TPM_ALG_RSAPSS_3072</a></p> <p>Byte 0 Bit 4. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P256</a></p> <p>Byte 0 Bit 5. <a href="#">TPM_ALG_RSASSA_4096</a></p> <p>Byte 0 Bit 6. <a href="#">TPM_ALG_RSAPSS_4096</a></p> <p>Byte 0 Bit 7. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P384</a></p> <p>Byte 1 Bit 0. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P521</a></p> <p>All other values reserved.</p>
4	AlgExternal	4*ExtAlgCount4'	<p>If present: a Requester-supported, Responder-selected, extended asymmetric key signature algorithm for the purpose of signature verification. The <a href="#">Extended algorithm field format</a> table describes the format of this field.</p>

200 **KeySchedule structure**

Offset	Field	Size (bytes)	Value
0	AlgType	1	0x5=KeySchedule

Offset	Field	Size (bytes)	Value
1	AlgCount	1	<p>Bit [7:4] = 2.</p> <p>Bit [3:0] Number of Requester-supported, Responder-selected, extended key schedule algorithms (= ExtAlgCount5'). This value shall be either 0 or 1.</p>
2	AlgSupported	2	<p>Bit mask for indicating a Requester-supported, Responder-selected, SPDM-enumerated Key Schedule algorithm.</p> <p>Byte 0 Bit 0. SPDM Key Schedule.</p> <p>All other values reserved.</p>
4	AlgExternal	4*ExtAlgCount5'	<p>If present: a Requester-supported, Responder-selected, extended key schedule algorithm. The <a href="#">Extended algorithm field format</a> table describes the format of this field.</p>

201 **Extended Algorithm field format**

202 Describes algorithms that are external to this specification.

Offset	Field	Description
0	Registry ID	Shall represent the registry or standards body. The <b>ID</b> column in the <a href="#">Registry or standards body ID</a> table describes the value of this field.
1	Reserved	Reserved
[2:3]	Algorithm ID	Shall indicate the desired algorithm. The registry or standards body owns the value of this field. For details, see the <a href="#">Registry or standards body ID</a> table.

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

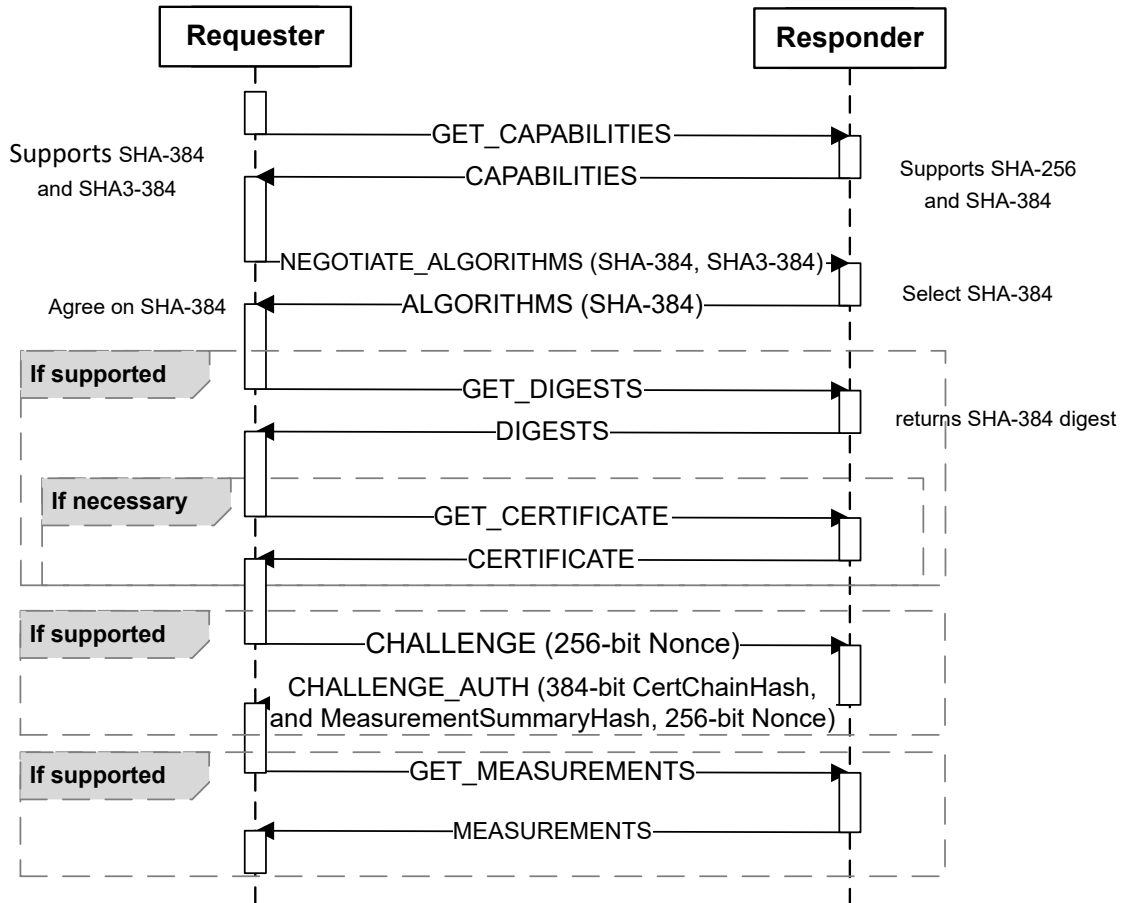
204 [Hashing algorithm selection: Example 1](#) illustrates how two endpoints negotiate a base hashing algorithm.



205 In [Hashing algorithm selection: Example 1](#), endpoint A issues `NEGOTIATE_ALGORITHMS` request message and endpoint B selects an algorithm of which both endpoints are capable.

206 **Hashing algorithm selection: Example 1**

207



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

209 **10.4.1 Behavior after VERSION, CAPABILITIES and ALGORITHMS**

210 With the successful completion of the `ALGORITHMS` message, all of the parameters for further SPDM message exchanges between the same pair of Requester and Responder have been determined. Thus, all SPDM message exchanges after the `VERSION`, `CAPABILITIES` AND `ALGORITHMS` 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. The Responder selects the `TPM_ALG_RSASSA_2048` asymmetric algorithm in `BaseAsymSel` and the `TPM_ALG_SHA_256` hash algorithm in `BaseHashSel` . If the corresponding Requester 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.

- 211 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, the corresponding Responder shall either silently discard the request or return an `ERROR` message with an appropriate error code.

## 212 10.5 Responder identity authentication

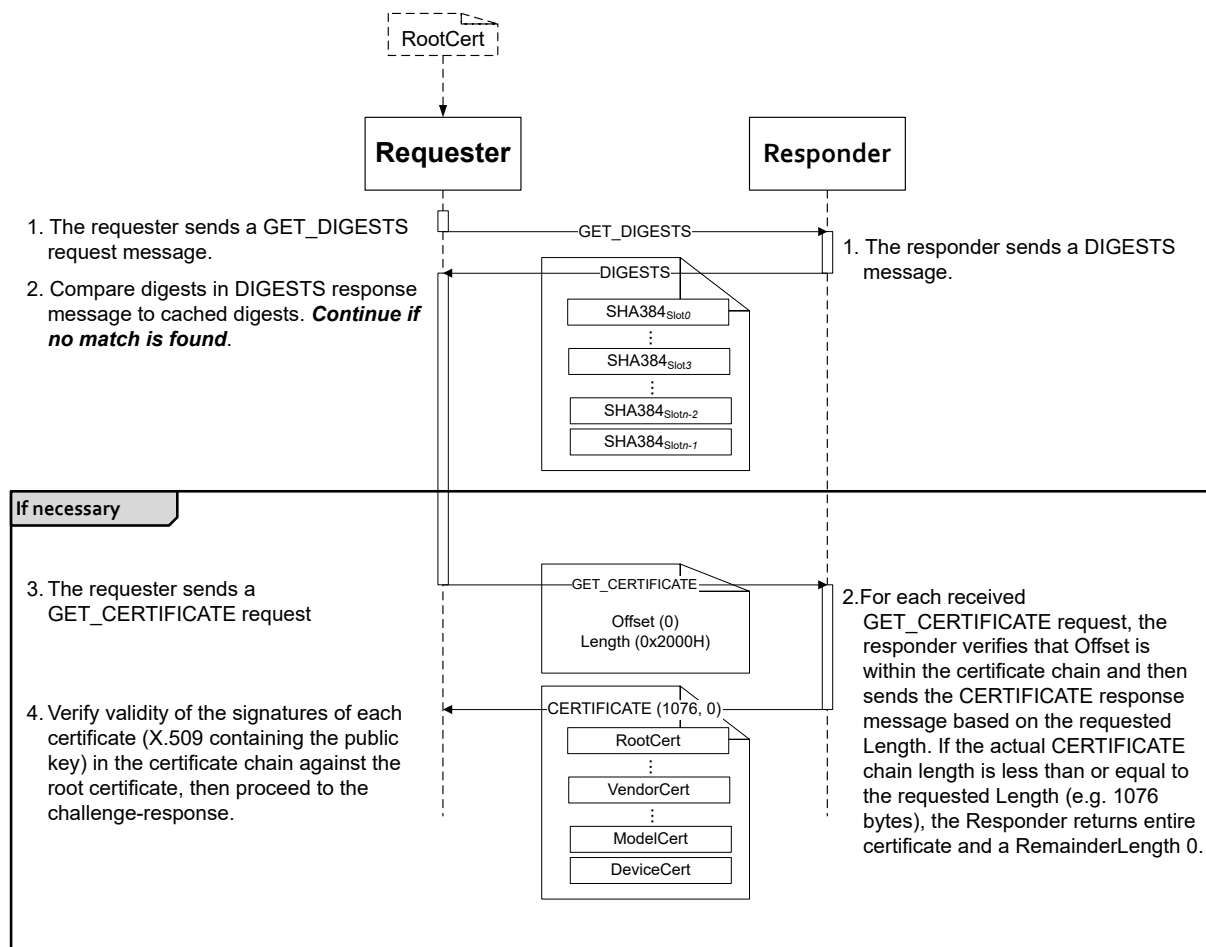
---

- 213 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 defined in this clause 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.

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

- 215 **Responder authentication: Example certificate retrieval flow**

216



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

218 For the runtime challenge-response flow, the signature field in the CHALLENGE\_AUTH response message payload shall contain the signature generated by using the device private key over the hash of the message transcript. See the [Request ordering and message transcript computation rules for M1/M2 table](#).

219 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.

220 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.

## 221 10.6 Requester identity authentication

222 If the Requester supports mutual authentication, the requirements placed on the Responder in [Responder identity authentication](#) shall also apply to the Requester.

223 If the Responder supports mutual authentication, the requirements placed on the Requester in [Responder identity authentication](#) shall also apply to the Responder. These two statements essentially describe a role reversal.

### 224 10.6.1 Certificates and certificate chains

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

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

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

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

#### 229 Certificate chain format

Offset	Field	Size	Description
0	<code>Length</code>	2	Total length of the certificate chain, in bytes, including all fields in this table. This field is little endian.
2	<code>Reserved</code>	2	Reserved.
4	<code>RootHash</code>	H	Digest of the Root Certificate. Note that Root Certificate is ASN.1 DER-encoded for this digest. This field shall be big endian.

Offset	Field	Size	Description
4 + H	Certificates	Length - (4 + H)	One or more ASN.1 DER-encoded X.509 v3 certificates where the first certificate is signed by the Root Certificate or is the Root Certificate itself and each subsequent certificate is signed by the preceding certificate. The last certificate is the <i>leaf certificate</i> . This field shall be big endian.

**230 10.7 GET\_DIGESTS request and DIGESTS response messages**

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

232 The [GET\\_DIGESTS request message format](#) table shows the GET\_DIGESTS request message format.

233 The [Successful DIGESTS response message](#) table shows the DIGESTS response message format.

234 The digests in the [Successful DIGESTS response message](#) table shall be big endian, and the digest shall be computed over the certificate chain as shown in [Certificate chain format](#).

**235 GET\_DIGESTS request message format**

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	RequestResponseCode	1	0x81=GET_DIGESTS
2	Param1	1	Reserved
3	Param2	1	Reserved

**236 Successful DIGESTS response message format**

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	RequestResponseCode	1	0x01=DIGESTS
2	Param1	1	Reserved

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

## 237 10.8 GET\_CERTIFICATE request and CERTIFICATE response messages

238 This request message shall retrieve the certificate chains.

239 The [GET\\_CERTIFICATE request message format](#) table shows the GET\_CERTIFICATE request message format.

240 The [Successful CERTIFICATE response message](#) table shows the CERTIFICATE response message format.

241 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.

### 242 GET\_CERTIFICATE request message format

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	RequestResponseCode	1	0x82=GET_CERTIFICATE

Offset	Field	Size (bytes)	Value
2	Param1	1	Slot number of the target certificate chain to read from. The value in this field shall be between 0 and 7 inclusive.
3	Param2	1	Reserved
4	Offset	2	Offset in bytes from the start of the certificate chain to where the read request message begins. The Responder should send its certificate chain starting from this offset. For the first GET_CERTIFICATE request for a given slot, the Requester shall set this field to 0. For subsequent requests, Offset is set to the next portion of the certificate in that slot.
6	Length	2	<p>Length of certificate chain data, in bytes, to be returned in the corresponding response.</p> <p>Length is an unsigned 16-bit integer.</p> <p>This value is the smaller of the following values:</p> <p>Capacity of the internal buffer of the Requester for receiving the certificate chain of the Responder.</p> <p>The RemainderLength of the preceding GET_CERTIFICATE response.</p> <p>If offset=0 and length=0xFFFF, the Requester is requesting the entire chain.</p>

243 **Successful CERTIFICATE response message format**

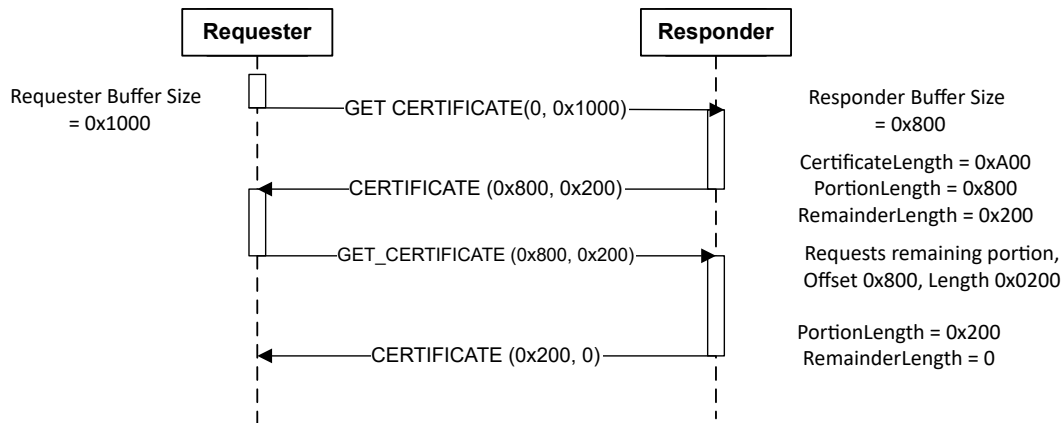
Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	RequestResponseCode	1	0x02=CERTIFICATE
2	Param1	1	Slot number of the certificate chain returned.
3	Param2	1	Reserved.
4	PortionLength	2	Number of bytes of this portion of certificate chain. This should be less than or equal to Length received as part of the request. For example, the Responder might set this field to a value less than Length received as part of the request due to limitations on the internal buffer of the Responder.
6	RemainderLength	2	Number of bytes of the certificate chain that have not been sent yet after the current response. For the last response, this field shall be 0 as an indication to the Requester that the entire certificate chain has been sent.
8	CertChain	PortionLength	Requested contents of target certificate chain, as described in <a href="#">Certificates and certificate chains</a> .

244 The [Responder unable to return full length data flow](#) shows the high-level request-response message flow for Responder response when it cannot return the entire data requested by the Requester in the first response.

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



246



**247 10.8.1 Mutual authentication requirements for GET\_CERTIFICATE and CERTIFICATE messages**

248 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 describes a role reversal.

**249 10.8.2 Leaf certificate**

250 The SPDM endpoints for authentication shall be provisioned with DER-encoded X.509 v3 format certificates. The leaf certificate shall be signed by a trusted CA and provisioned to the device. For endpoint devices to verify the certificate, the following [required fields](#) shall be present. In addition, to provide device information, use the `Subject Alternative Name` certificate extension `otherName` field. See the [Definition of otherName using the DMTF OID](#).

**251 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.

Field	Description
Validity	Certificates may include this attribute. See <a href="#">RFC5280</a> for further details.
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.

## 252 Optional fields

Field	Description
Basic Constraints	If present, the CA value shall be FALSE .
Subject Alternative Name otherName	In some cases, it might be desirable to provide device specific information as part of the device certificate. DMTF chose the otherName field with a specific format to represent the device information. The use of the otherName field also provides flexibility for other alliances to provide device specific information as part of the device certificate. See the <a href="#">Definition of otherName using the DMTF OID</a> .

## 253 Definition of otherName using the DMTF OID

```

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

-- All printable characters except ":" --
DMTF-device-string
    ::= UTF8String (ALL EXCEPT ":")

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

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

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

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

```

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

255 **10.9 CHALLENGE request and CHALLENGE\_AUTH response messages**

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

257 The [CHALLENGE request message format](#) table shows the CHALLENGE request message format.

258 The [Successful CHALLENGE\\_AUTH response message](#) table shows the CHALLENGE\_AUTH response message format.

259 **CHALLENGE request message format**

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	RequestResponseCode	1	0x83=CHALLENGE
2	Param1	1	Slot number of the certificate chain of the Responder that shall be used for authentication. It shall be 0xFF if the public key of the Responder was provisioned to the Requester in a trusted environment.
3	Param2	1	Requested measurement summary hash Type: 0x0 . No measurement summary hash. 0x1 . TCB measurement hash. 0xFF . All measurements hash. All other values reserved. When Responder does not support any measurements, Requester shall set this value to 0x0 .
4	Nonce	32	The Requester should choose a random value.

260 **Successful CHALLENGE\_AUTH response message format**

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	RequestResponseCode	1	0x03=CHALLENGE_AUTH
2	Param1	1	Response Attribute Field. Please see <a href="#">CHALLENGE_AUTH Response Attribute Table</a> for details.
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 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 is big endian.
4 + H	Nonce	32	Responder-selected random value.

Offset	Field	Size (bytes)	Value
36 + H	MeasurementSummaryHash	H	<p>When the Responder does not support measurements ( <code>MEAS_CAP=00b</code> in <code>CAPABILITIES</code> response) or requested <code>Param2 =0</code>, this field shall be absent.</p> <p>When the requested <code>Param2 =1</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 <a href="#">index</a>.</p> <p>When the requested <code>Param2 =1</code> and there are no measurable components in the TCB required to generate this response, this field shall be <code>0</code>.</p> <p>When requested <code>Param2=0xFF</code>, this field is computed as the <code>hash(Concatenation(MeasurementBlock[0], MeasurementBlock[1], ..., MeasurementBlock[n]))</code> of all supported measurement blocks available in the measurement index range <code>0x01 - 0xFE</code>, concatenated in ascending index order. Indices with no associated measurements shall not be included in the hash calculation.</p>
36 + 2H	OpaqueLength	2	Size of the <code>OpaqueData</code> field. The value shall not be greater than 1024 bytes.
38 + 2H	OpaqueData	OpaqueLength	Free-form field, if present. The Responder may include Responder-specific information and/or information defined by its transport.

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

## 261 CHALLENGE\_AUTH response attribute

Bit Offset	Field Name	Description
[3:0]	SlotID	This field shall contain the slot number 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	BasicMutAuthReq	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> for more details.  If mutual authentication is not supported, this bit shall be zero; otherwise, it should be considered an error.

## 262 10.9.1 CHALLENGE\_AUTH signature generation

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

- 264 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.

265 where:

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

267 ◦ If a response contains `ErrorCode=ResponseNotReady` :

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

269 ◦ If a response contains an `ErrorCode` other than `ResponseNotReady` :

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

271 2. The Responder shall generate:

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

272 where:

273 ◦ `Sign`

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

275 The [Successful ALGORITHMS response message format](#) table describes the `BaseAsymSel` , `ExtAsymSel` and `RespAlgStruct` (when `AlgType == ReqBaseAsymAlg` ) fields.

276 ◦ `SK`

277 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 `SK` is the associated private key.

278 ◦ `Hash`

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

280 The [Successful ALGORITHMS response message format](#) table describes the `BaseHashSel` and `ExtHashSel` fields.

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

## 282 10.9.2 CHALLENGE\_AUTH signature verification

283 Modifications to the previous request messages or the corresponding response messages by an active person-in-the-middle adversary or media error result in  $M2 \neq M1$  and lead to verification failure.

284 To complete the CHALLENGE\_AUTH signature verification process, the Requester shall complete this step:

285 1. The Requester shall perform:

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

286 where:

287 ◦ Verify

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

289 The Successful ALGORITHMS response message format table describes the BaseAsymSel, ExtAsymSel and RespAlgStruct (when AlgType == ReqBaseAsymAlg) fields.

290 ◦ PK

291 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, then PK is the provisioned public key.

292 ◦ Hash

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

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

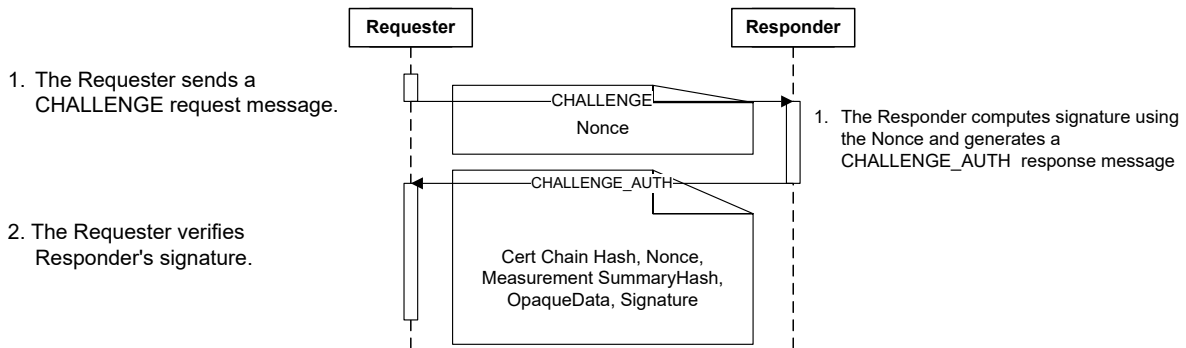
295 If the verification algorithm first hashes the message before generating the signature, the verification algorithm's hashing step shall be skipped.

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

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



298



**299 10.9.2.1 Request ordering and message transcript computation rules for M1 and M2**

300 This clause applies to Responder-only authentication.

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

302 The possible request orderings after reset leading up to and including `CHALLENGE` are:

- `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` , `CHALLENGE` (A1, B2, C1)
- `GET_DIGESTS` , `GET_CERTIFICATE` , `CHALLENGE` (A2, B1, C1)
- `GET_DIGESTS` , `CHALLENGE` (A2, B3, C1)
- `CHALLENGE` (A2, B2, C1)

303 Immediately after reset, M1 and M2 shall be null.

304 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.

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

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

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

Requests	Implementation requirements	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=Concatenate(GET_VERSION, VERSION, GET_CAPABILITIES, CAPABILITIES, NEGOTIATE_ALGORITHMS, ALGORITHMS)
GET_VERSION , GET_CAPABILITIES , NEGOTIATE_ALGORITHMS Skipped	Requester skipped issuing these requests after a new reset if the Responder has previously indicated CACHE_CAP=1 . In this case, the Requester and Responder shall proceed with the previously determined Negotiated State .	A2=null
GET_DIGESTS , GET_CERTIFICATE issued	Requester issued these requests in this order after NEGOTIATE_ALGORITHMS request completion or immediately after reset, 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 new reset since it could use previously cached certificate information.	B2=null
GET_DIGESTS issued, GET_CERTIFICATE skipped	Requester skipped GET_CERTIFICATE request after a new reset since it could use the previously cached CERTIFICATE response.	B3=(GET_DIGESTS, DIGESTS)
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=(CHALLENGE, CHALLENGE_AUTH\Signature) . See the <a href="#">CHALLENGE request message format</a> table.
CHALLENGE completion	Completion of CHALLENGE resets M1 and M2.	M1/M2=null
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 and M2 are reset to null .	M1/M2=null

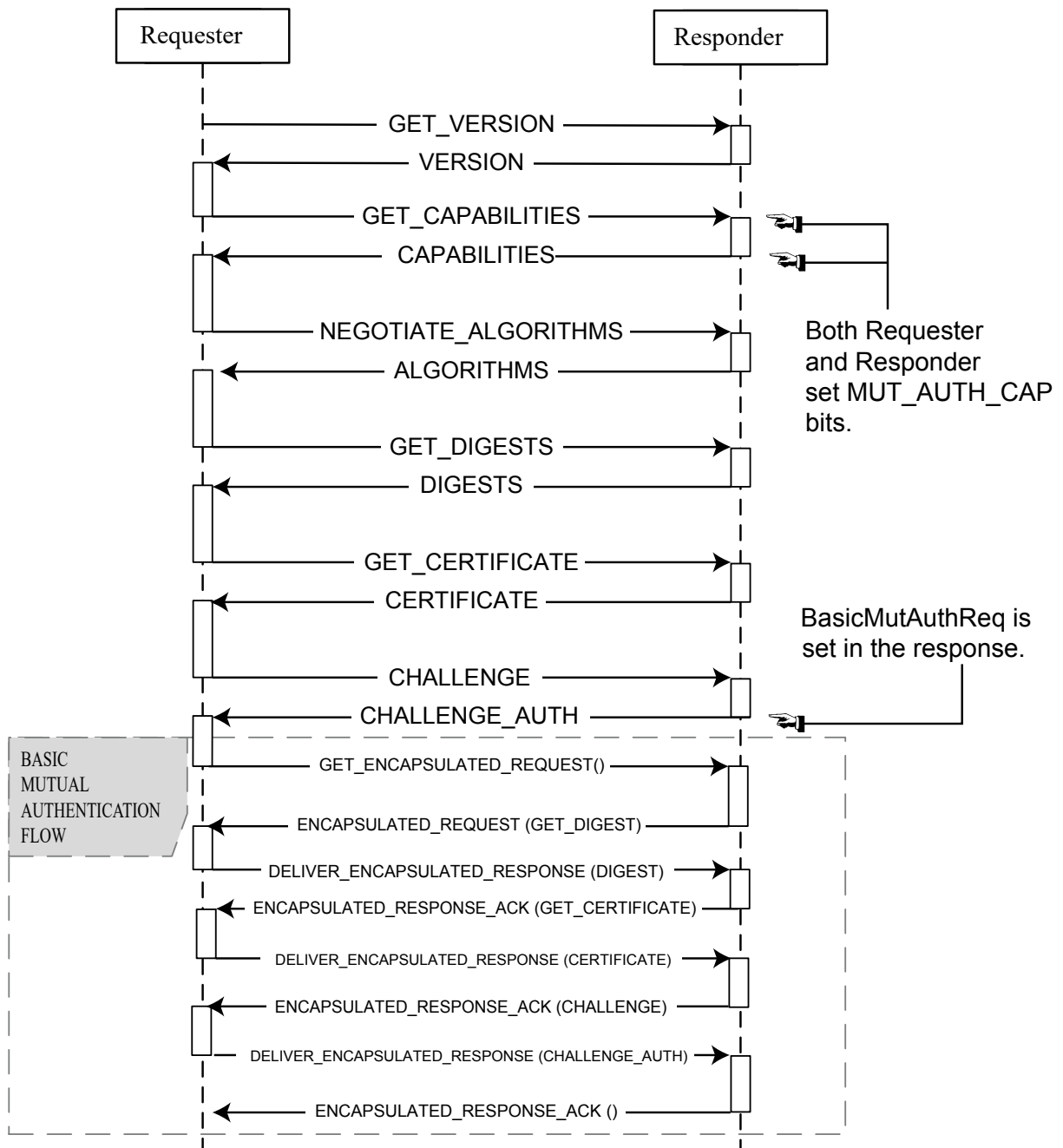
### 307 10.9.3 Basic mutual authentication

308 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.

- 309 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.
- 310 This flow shall only allow `GET_DIGESTS` , `GET_CERTIFICATE` , `CHALLENGE` and their corresponding responses to be encapsulated.
- 311 The [Mutual authentication basic flow](#) illustrates, as an example, the basic mutual authentication flow.
- 312 **Mutual authentication basic flow**

313



**314 10.9.3.1 Mutual authentication message transcript**

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

316 The [Basic mutual authentication message transcript](#) table defines how the message transcript is constructed for M1

and M2, which are used in signature calculation and verification in the `CHALLENGE_AUTH` response message when the Responder authenticates the Requester.

317 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*)

318 When the basic mutual authentication flow starts (i.e., when `GET_ENCAPSULATED_REQUEST` is issued) M1 and M2 shall be set to NULL.

319 **Basic mutual authentication message transcript**

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

320 For `GET_CERTIFICATE` and `CERTIFICATE` , these messages may 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.

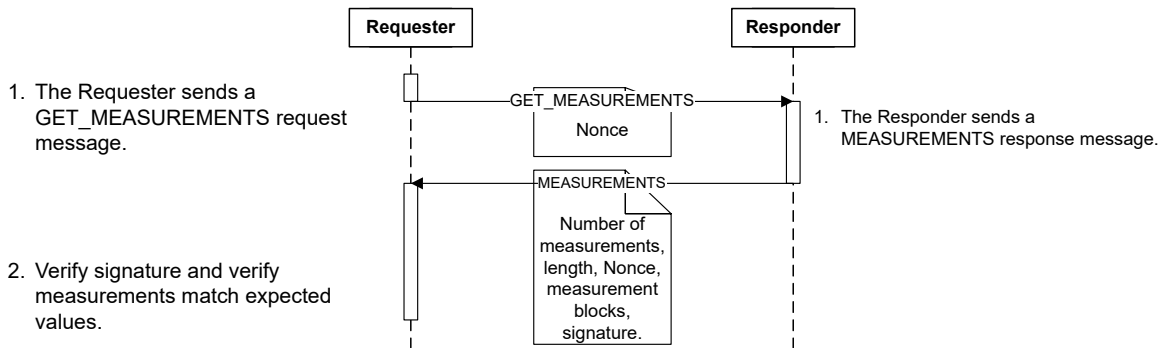
321 **10.10 Firmware and other measurements**

322 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.

323 The [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.

324 **Measurement retrieval flow**

325



### 326 10.11 GET\_MEASUREMENTS request and MEASUREMENTS response messages

327 This request message shall retrieve measurements in the form of measurements blocks. A Requester should not send this message until it has received at least one successful CHALLENGE\_AUTH response message from the Responder, or should send this message in a secure session. The successful CHALLENGE\_AUTH response may have been received before the last reset.

328 The GET\_MEASUREMENTS request message format table shows the GET\_MEASUREMENTS request message format.

329 The GET\_MEASUREMENTS request attributes table shows the GET\_MEASUREMENTS request message attributes.

330 The Successful MEASUREMENTS response message format table shows the MEASUREMENTS response message format. The measurement blocks in MeasurementRecord shall be placed contiguously from index 1 and shall be sorted in ascending order by index.

#### 331 GET\_MEASUREMENTS request message format

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	RequestResponseCode	1	0xE0=GET_MEASUREMENTS
2	Param1	1	Request attributes. See the GET_MEASUREMENTS request attributes table.

Offset	Field	Size (bytes)	Value
3	Param2	1	<p>Measurement operation.</p> <p>A value of 0x0 shall query the Responder for the total number of measurement blocks available.</p> <p>A value of 0xFF shall request all measurement blocks.</p> <p>A value between 0x1 and 0xFE, inclusively, shall request the measurement block at the index corresponding to that value.</p>
4	Nonce	32	<p>The Requester should choose a random value. This field is only present if a signature is required on the response. See the <a href="#">GET_MEASUREMENTS request attributes</a> table.</p>
36	SlotIDParam	1	<p>Bit[7:4] = Reserved.</p> <p>Bit[3:0] = SlotID. Slot number of the certificate chain of the Responder 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. This field is only present if a signature is required on the response. See the <a href="#">GET_MEASUREMENTS request attributes</a> table.</p>

332 GET\_MEASUREMENTS request attributes

Bits	Value	Description
0	1	If the Responder can generate a signature ( <code>MEAS_CAP</code> is <code>10b</code> in the <code>CAPABILITIES</code> response), the value of this bit shall indicate to the Responder that a signature is required. The Responder shall generate a signature in the corresponding response. The <code>Nonce</code> field shall be present in the request.
0	0	For Responders that can generate signatures, the value of this bit shall indicate that the Requester does not require a signature. The Responder shall not generate a signature in the response. The <code>Nonce</code> field shall be absent in the request.  For Responders that cannot generate a signature ( <code>MEAS_CAP</code> is <code>01b</code> in the <code>CAPABILITIES</code> response) the Requester shall always use this value.
[7:1]	Reserved	Reserved

333 **Successful MEASUREMENTS response message format**

Offset	Field	Size (bytes)	Value
0	<code>SPDMVersion</code>	1	V1.1 = <code>0x11</code>
1	<code>RequestResponseCode</code>	1	<code>0x60=MEASUREMENTS</code>
2	<code>Param1</code>	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	<code>Param2</code>	1	Bit[7:4] = Reserved.  Bit[3:0] = SlotID. 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> .



Offset	Field	Size (bytes)	Value
4	NumberOfBlocks	1	Number of measurement blocks (N) in <code>MeasurementRecord</code> . If <code>Param2</code> in the requested measurement operation is $\emptyset$ , this field shall be $\emptyset$ .
5	MeasurementRecordLength	3	Size of the <code>MeasurementRecord</code> field in bytes. If <code>Param2</code> in the requested measurement operation is $\emptyset$ , this field shall be $\emptyset$ .
8	MeasurementRecord	L = MeasurementRecordLength	Concatenation of all measurement blocks that correspond to the requested Measurement operation. <a href="#">Measurement block</a> defines the measurement block structure.
8 + L	Nonce	32	The Responder should choose a random value.
40 + L	OpaqueLength	2	Size of the <code>OpaqueData</code> field in bytes. The value shall not be greater than 1024 bytes.
42 + L	OpaqueData	OpaqueLength	Free-form field, if present. The Responder may include Responder-specific information and/or information defined by its transport.

Offset	Field	Size (bytes)	Value
42 + L + OpaqueLength	Signature	S	Signature of the GET_MEASUREMENTS request and MEASUREMENTS response messages, excluding the Signature field and signed using the device private key. The Responder shall use the asymmetric signing algorithm it selected during the last ALGORITHMS response message to the Requester, and S is the size of that asymmetric signing algorithm output. This field is conditional.

**334 10.11.1 Measurement block**

335 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 .

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

**337 Measurement block format**

Offset	Field	Size (bytes)	Value
0	Index	1	Index. Shall represent the index of the measurement. In the range of [1, N].

Offset	Field	Size (bytes)	Value
1	MeasurementSpecification	1	<p>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 the <a href="#">Successful ALGORITHMS response message format table</a>. Only one bit shall be set in the measurement block.</p> <p>Bit 0=DMTF, as specified in the <a href="#">Measurement field format when MeasurementSpecification field is Bit 0 = DMTF</a> table.</p> <p>All other bits are reserved.</p>
2	MeasurementSize	2	Size of <code>Measurement</code> , in bytes.
4	Measurement	MeasurementSize	The <code>MeasurementSpecification</code> defines the format of this field.

**338 10.11.1.1 DMTF specification for the Measurement field of a measurement block**

339 The present clause is the specification for the format of the `Measurement` field in a measurement block when the `MeasurementSpecification` field selects Bit 0=DMTF. This format is specified in [Measurement field format when MeasurementSpecification field is Bit 0 = DMTF](#).

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

**341 Measurement field format when MeasurementSpecification field is bit 0 = DMTF**

Offset	Field	Size (bytes)	Value
0	DMTFSpecMeasurementValueType	1	<p>Composed of:</p> <p>Bit [7] indicates the representation in <code>DMTFSpecMeasurementValue</code>.</p> <p>Bits [6:0] indicate what is being measured by <code>DMTFSpecMeasurementValue</code>.</p> <p>These values are set independently and are interpreted as follows:</p> <p>[7]=0b . Digest.</p> <p>[7]=1b . Raw bit stream.</p> <p>[6:0]=00h . Immutable ROM.</p> <p>[6:0]=01h . Mutable firmware.</p> <p>[6:0]=02h . Hardware configuration, such as straps, debug modes.</p> <p>[6:0]=03h . Firmware configuration, such as configurable firmware policy.</p> <p>[6:0]=04h . Measurement manifest.</p> <p>All other values reserved.</p>
1	DMTFSpecMeasurementValueSize	2	<p>Size of <code>DMTFSpecMeasurementValue</code>, in bytes.</p> <p>When <code>DMTFSpecMeasurementValueType[7]=0b</code>, the <code>DMTFSpecMeasurementValueSize</code> shall be derived from the measurement hash algorithm that the <code>ALGORITHM</code> response message returns.</p>
3	DMTFSpecMeasurementValue	<code>DMTFSpecMeasurementValueSize</code>	<p><code>DMTFSpecMeasurementValueSize</code> bytes of cryptographic hash or raw bit stream, as indicated in <code>DMTFSpecMeasurementValueType[7]</code>.</p>

342 **10.11.2 MEASUREMENTS signature generation**

343 While a Requester may opt to require a signature in each individual `MEASUREMENTS` response, it is advisable that the

cost of the signature generation process is minimized by amortizing it over multiple `MEASUREMENTS` responses where applicable. In this scheme, the Requester issues a number of `GET_MEASUREMENTS` requests without requiring signatures followed by a final `GET_MEASUREMENTS` request requiring a signature over the entire set of `GET_MEASUREMENTS` requests and corresponding `MEASUREMENTS` responses exchanged. The steps to complete this scheme are as follows:

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

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

- 345 where:

- 346 ◦ `Concatenate()`

347 Standard concatenation function.

- 348 ◦ `GET_MEASUREMENTS_REQUEST1`

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

- 350 ◦ `MEASUREMENTS_RESPONSE1`

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

- 352 ◦ `GET_MEASUREMENTS_REQUESTn-1`

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

- 354 ◦ `MEASUREMENTS_RESPONSEn-1`

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

- 356 ◦ `GET_MEASUREMENTS_REQUESTn`

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

- 358  $n$  is a number greater than or equal to 1.

- 359 When  $n$  equals 1, the Requester has not made any `GET_MEASUREMENTS` requests without signature prior to issuing a `GET_MEASUREMENTS` request with signature.
- 360 ◦ `MEASUREMENTS_RESPONSE $n$`
- 361 Entire `MEASUREMENTS` response message without the signature bytes that the Responder sent in response to `GET_MEASUREMENTS_REQUEST $n$` .
- 362 Any communication between Requester and Responder other than a `GET_MEASUREMENTS` request or response resets L1/L2 computation to null.

- 363 2. The Responder shall generate:

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

- 364 where:

- 365 ◦ `Sign`
- 366 Asymmetric signing algorithm that the Responder selected through the last `ALGORITHMS` response message that the Responder sent.
- 367 The [Successful ALGORITHMS response message format](#) table describes the `BaseAsymSel` and `ExtAsymSel` fields.
- 368 ◦ `SK`
- 369 Private key of the Responder associated with the leaf certificate stored in `SlotID`. If the public key of the Responder was provisioned to the Requester, then `SK` is the associated private key.
- 370 ◦ `Hash`
- 371 Hashing algorithm that the Responder selected through the last `ALGORITHMS` response message that the Responder sent.
- 372 The [Successful ALGORITHMS response message format](#) table describes the `BaseHashSel` and `ExtHashSel` fields.
- 373 If the signing algorithm first hashes the message before generating the signature, the signing algorithm's hashing step shall be skipped.

### 374 10.11.3 MEASUREMENTS signature verification

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

376 1. The Requester shall perform:

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

377 where:

378 ◦ `PK`

379 Public key associated with the slot 0 certificate of the Responder. `PK` is extracted from the `CERTIFICATES` response. If the public key of the Responder was provisioned to the Requester, then `PK` is the provisioned public key.

380 ◦ `Verify`

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

382 The [Successful ALGORITHMS response message format](#) table describes the `BaseAsymSel` and `ExtAsymSel` fields.

383 ◦ `Hash`

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

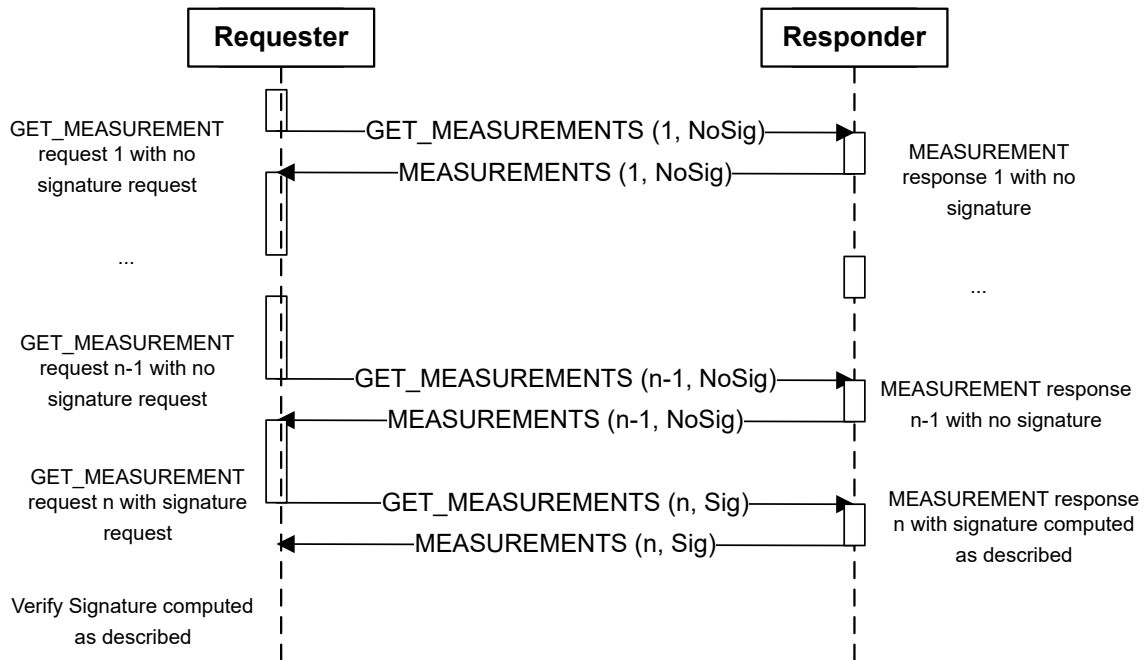
385 The [Successful ALGORITHMS response message format](#) table describes the `BaseHashSel` and `ExtHashSel` fields.

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

387 The [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.

### 388 Measurement signature computation example

389



390 **10.12 ERROR response message**

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

392 The [ERROR response message format](#) table shows the `ERROR` response format.

393 The [Error code and error data](#) table shows the detailed error code, error data, and extended error data.

394 The [ResponseNotReady extended error data](#) table shows the `ResponseNotReady` extended error data.

395 The [Registry or standards body ID](#) table shows the registry or standards body ID.

396 The [ExtendedErrorData format for vendor or other standards-defined ERROR response message](#) table shows the `ExtendedErrorData` format definition for vendor or other standards-defined `ERROR` response message.

397 **ERROR response message format**

Offset	Field	Size (bytes)	Value
0	<code>SPDMVersion</code>	1	<code>V1.1 = 0x11</code>
1	<code>RequestResponseCode</code>	1	<code>0x7F=ERROR</code>



Offset	Field	Size (bytes)	Value
2	Param1	1	Error Code. See <a href="#">Error code and error data</a> .
3	Param2	1	Error Data. See <a href="#">Error code and error data</a> .
4	ExtendedErrorData	0-32	Optional extended data. See <a href="#">Error code and error data</a> .

398 **Error code and error data**

Error code	Value	Description	Error data	ExtendedErrorData
Reserved	0x00	Reserved	Reserved	Reserved
InvalidRequest	0x01	One or more request fields are invalid	0x00	No extended error data is provided.
Reserved	0x02	Reserved	Reserved	Reserved
Busy	0x03	The Responder received the request message and the Responder decided to ignore the request message, but the Responder may be able to process the request message if the request message is sent again in the future.	0x00	No extended error data is provided.
UnexpectedRequest	0x04	The Responder received an unexpected request message. For example, CHALLENGE before NEGOTIATE_ALGORITHMS .	0x00	No extended error data is provided.
Unspecified	0x05	Unspecified error occurred.	0x00	No extended error data is provided.
DecryptError	0x06	The receiver of the record cannot decrypt the record or verify data during the session handshake.	Reserved	No extended error data is provided.

Error code	Value	Description	Error data	ExtendedErrorData
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
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.
Reserved	0x0b - 0x40	Reserved	Reserved	Reserved
MajorVersionMismatch	0x41	Requested SPDM Major Version is not supported.	0x00	No extended error data provided.
ResponseNotReady	0x42	See the <a href="#">RESPOND_IF_READY request message format</a> .	0x00	See the <a href="#">ResponseNotReady extended error data table</a> .
RequestResynch	0x43	Responder is requesting Requester to reissue GET_VERSION to resynchronize. An example is following a firmware update.	0x00	No extended error data provided.
Reserved	0x44 - 0xFE	Reserved	Reserved.	Reserved
Vendor/Other Standards Defined	0xFF	Vendor or Other Standards defined	Shall indicate the registry or standard body using one of the values in the ID column in the <a href="#">Registry or standards body ID table</a> .	See the <a href="#">ExtendedErrorData format for vendor or other standards-defined ERROR response message table</a> for format definition.

399 **ResponseNotReady extended error data**

Offset	Field	Size (bytes)	Value
0	RDTExponent	1	<p>Exponent expressed in logarithmic (base 2 scale) to calculate <code>RDTime</code> 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>RDTime</code> to avoid continuous pinging and issue the <code>RESPOND_IF_READY</code> request message after <code>RDTime</code> time.</p> <p>For timing requirement details, see the <a href="#">Timing specification for SPDM messages</a> table.</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.

Offset	Field	Size (bytes)	Value
3	RDTM	1	<p>Multiplier used to compute <math>WT_{Max}</math> in <math>\mu s</math> to indicate the response may be dropped after this delay.</p> <p>The multiplier shall always be greater than 1.</p> <p>The Responder may also stop processing the initial request if the same Requester issues a different request.</p> <p>For timing requirement details, see the <a href="#">Timing specification for SPDM messages</a> table.</p>

400 **Registry or standards body ID**

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

ID	Vendor ID length (bytes)	Registry or standards body name	Description
0x0	0	DMTF	DMTF does not have a Vendor ID registry. At present, DMTF does not have any algorithms defined for use in extended algorithms fields.
0x1	2	TCG	VendorID is identified by using <a href="#">TCG Vendor ID Registry</a> . For extended algorithms, see <a href="#">TCG Algorithm Registry</a> .
0x2	2	USB	VendorID is identified by using the vendor ID assigned by USB.
0x3	2	PCI-SIG	VendorID is identified using <a href="#">PCI-SIG Vendor ID</a> .

ID	Vendor ID length (bytes)	Registry or standards body name	Description
0x4	4	IANA	The <a href="#">Private Enterprise Number (PEN)</a> 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 <a href="#">Manufacturer ID</a> 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.

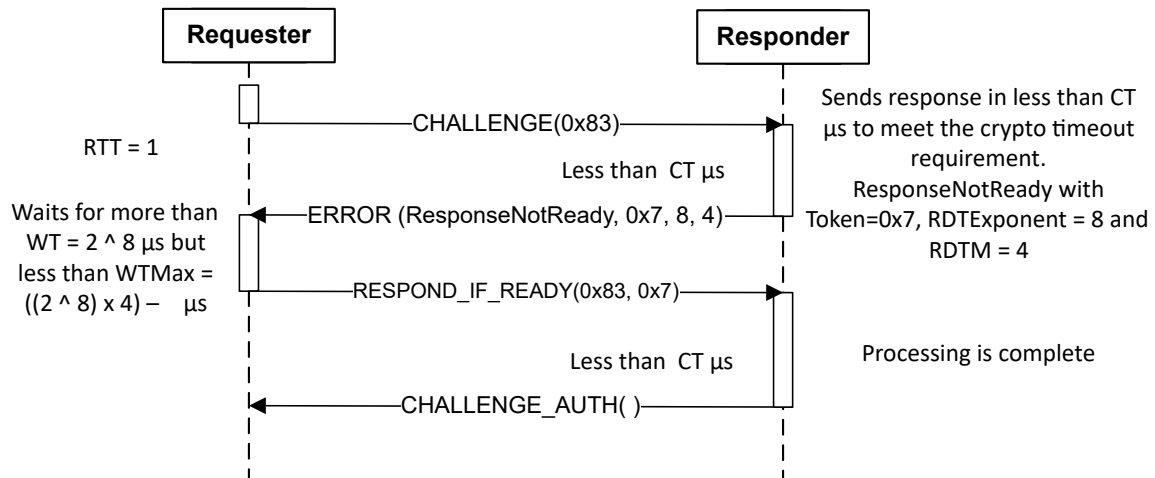
402 **ExtendedErrorData format for vendor or other standards-defined ERROR response message**

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

### 403 10.13 RESPOND\_IF\_READY request message format

404 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.

405



406 The [RESPOND\\_IF\\_READY request message format](#) table shows the `RESPOND_IF_READY` request message format.

407 **RESPOND\_IF\_READY request message format**

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	RequestResponseCode	1	0xFF=RESPOND_IF_READY
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.

408 **10.14 VENDOR\_DEFINED\_REQUEST request message**

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

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

411 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 hashes and/or message transcript are changed to add the vendor defined commands.

412 The [VENDOR\\_DEFINED\\_REQUEST request message format](#) table shows the `VENDOR_DEFINED_REQUEST` request message format.

413 **VENDOR\_DEFINED\_REQUEST request message format**

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	RequestResponseCode	1	0xFE=VENDOR_DEFINED_REQUEST
2	Param1	1	Reserved
3	Param2	1	Reserved
4	StandardID	2	Shall indicate the registry or standards body by using one of the values in the ID column in the <a href="#">Registry or standards body ID</a> table.
6	Len	1	Length of the Vendor ID field. If the VendorDefinedRequest is standard defined, Len shall be 0. If the VendorDefinedRequest is vendor-defined, Len shall equal Vendor ID Len, as the <a href="#">Registry or standards body ID</a> table describes.
7	VendorID	Len	Vendor ID, as assigned by the registry or standards body. Shall be in little endian format.
7 + Len	ReqLength	2	Length of the VendorDefinedReqPayload.
7 + Len + 2	VendorDefinedReqPayload	ReqLength	The standard or vendor shall use this field to send the request payload.

414 Other DMTF specifications may define VENDOR\_DEFINED\_REQUEST with StandardID set to 0. See [VendorDefinedReqPayload](#) and [VendorDefinedRespPayload](#) defined by DMTF specifications for more information.



## 415 10.15 VENDOR\_DEFINED\_RESPONSE response message

416 A Responder can use this response message in response to `VENDOR_DEFINED_REQUEST`. The [VENDOR\\_DEFINED\\_RESPONSE response message format](#) table defines the format.

417 The [VENDOR\\_DEFINED\\_RESPONSE response message format](#) table shows the response message format.

### 418 VENDOR\_DEFINED\_RESPONSE response message format

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	RequestResponseCode	1	0x7E=VENDOR_DEFINED_RESPONSE
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 ID column in the <a href="#">Registry or standards body ID</a> table.
6	Len	1	Length of the Vendor ID field. If the VendorDefinedRequest is standards-defined, length shall be 0. If the VendorDefinedRequest is vendor-defined, length shall equal Vendor ID Len, as the <a href="#">Registry or standards body ID</a> table describes.
7	VendorID	Len	Shall indicate the Vendor ID, as assigned by the registry or standards body. Shall be in little endian format.
7 + Len	RespLength	2	Length of the VendorDefinedRespPayload
7 + Len + 2	VendorDefinedRespPayload	RespLength	Standard or vendor shall use this value to send the response payload.

### 419 **10.15.1 VendorDefinedReqPayload and VendorDefinedRespPayload defined by DMTF specifications**

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

#### 421 **Format of VendorDefinedReqPayload and VendorDefinedRespPayload when StandardID is DMTF**

Byte offset	Field	Size (bytes)	Description
0	DSPNumber	2	Shall be the DMTF specification's DSP number in a 16-bit integer. For example, DSP0287 would use 0x011F.
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 the <a href="#">VersionNumberEntry definition</a> table.
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.

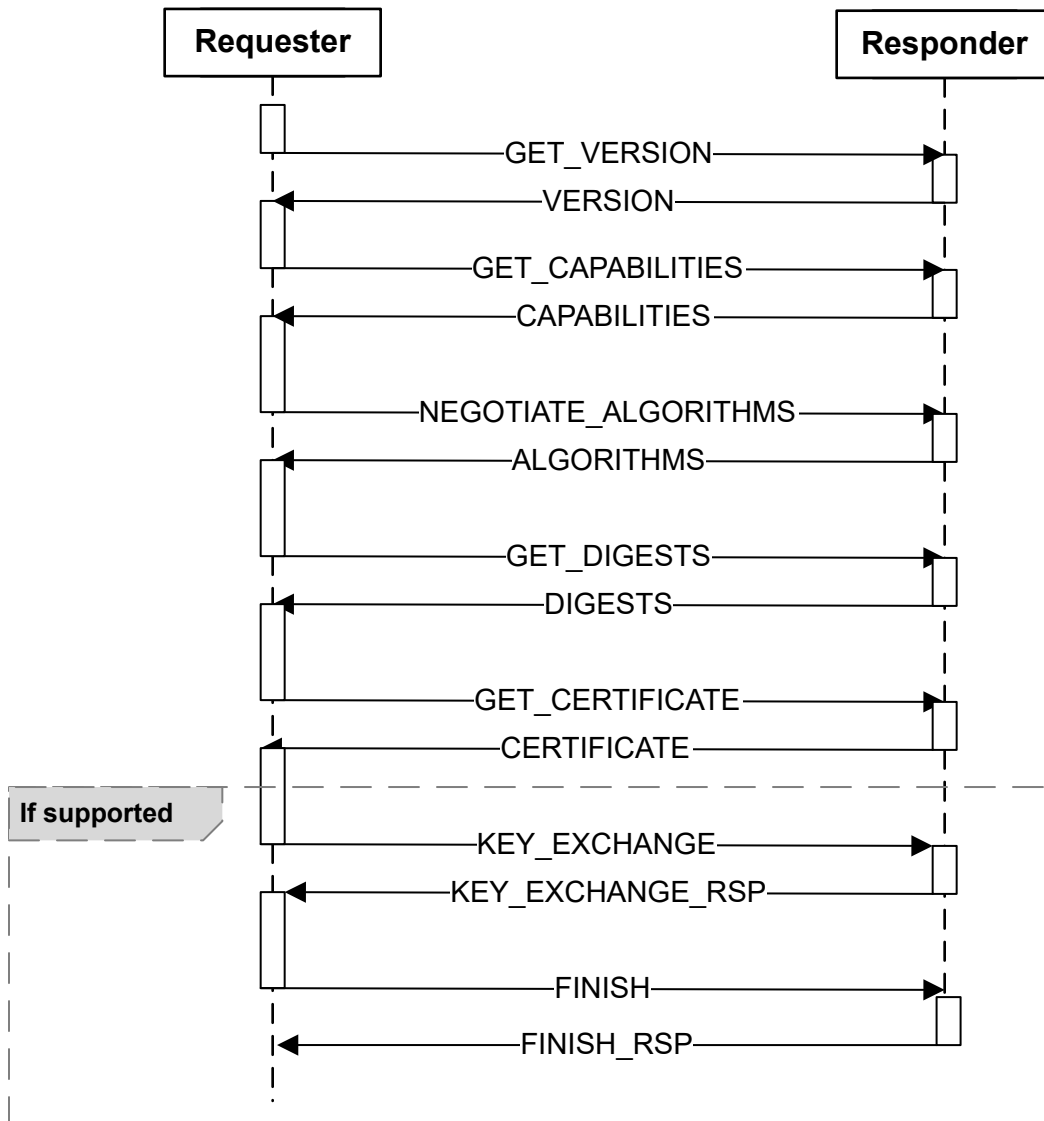
### 422 **10.16 KEY\_EXCHANGE request and KEY\_EXCHANGE\_RSP response messages**

423 This request message shall initiate a handshake between Requester and Responder intended to authenticate the Responder (or optionally both parties), negotiate cryptographic parameters (in addition to those negotiated in the last `NEGOTIATE_ALGORITHMS / ALGORITHMS` exchange), and establish shared keying material. The [KEY\\_EXCHANGE request message format](#) table shows the `KEY_EXCHANGE` request message format and the [Successful KEY\\_EXCHANGE\\_RSP response message format](#) table 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.

424 The Requester and Responder pair may 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.

#### 425 **Responder authentication key exchange example**

426

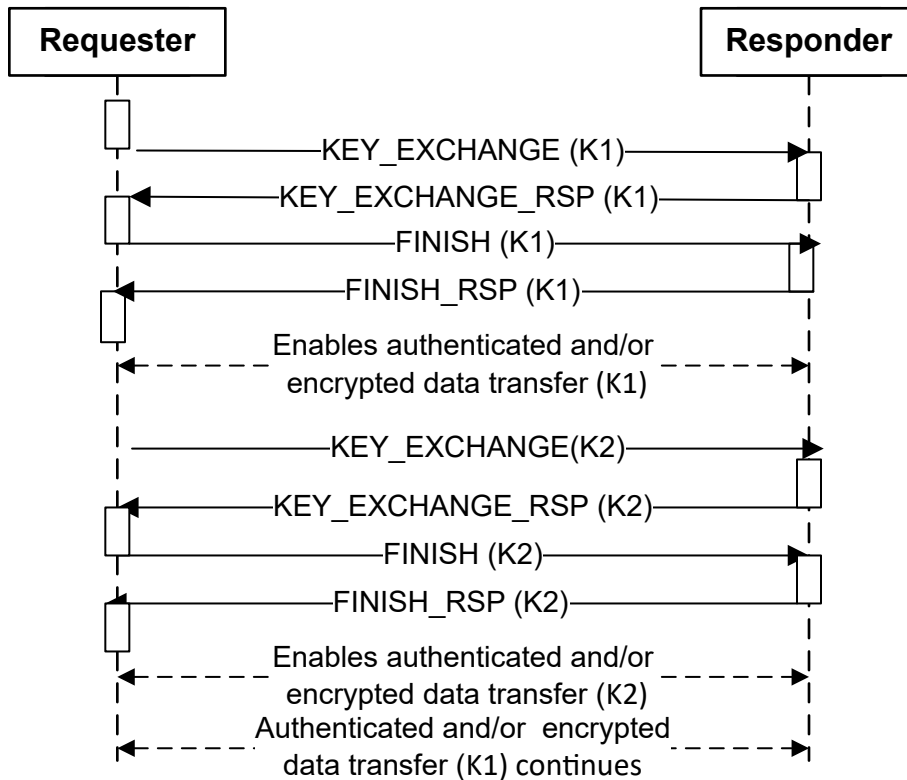


427 The [Responder authentication multiple key exchange example](#) provides an example of multiple sessions using two independent sets of root session keys that coexist at the same time. When `HANDSHAKE_IN_THE_CLEAR_CAP = 0` for both Requester and Responder, the specification does not require a specific 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.

#### 428 Responder authentication multiple key exchange example

429



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

431 Diffie-Hellman group parameters are determined by the DHE group in use, which is selected in the most recent `ALGORITHMS` response. The contents of the `ExchangeData` field are computed as specified in clause 4.2.8 of [RFC 8446](#). Specifically, if the DHE key exchange is based on finite-fields (FFDHE), the `ExchangeData` field in `KEY_EXCHANGE` and `KEY_EXCHANGE_RSP` shall contain the computed public value ( $Y = g^X \text{ mod } p$ ) for the specified group (see [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](#).

432 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.

433 **KEY\_EXCHANGE request message format**

Offset	Field	Size in bytes	Value
0	<code>SPDMVersion</code>	1	V1.1 = 0x11
1	<code>RequestResponseCode</code>	1	0xE4 = KEY_EXCHANGE
2	<code>Param1</code>	1	<p>Requested <code>MeasurementSummaryHash</code> type:</p> <p>0x0 . No measurement summary hash.</p> <p>0x1 . TCB measurement hash.</p> <p>0xFF . All measurements hash.</p> <p>All other values reserved.</p> <p>When Responder does not support any measurements, Requester shall set this value to 0x0 .</p>

Offset	Field	Size in bytes	Value
3	Param2	1	The slot number of the target certificate chain that the Responder will use for authentication. The value in this field shall be between 0 and 7 inclusive to identify a valid certificate slot. It shall be 0xFF 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	Reserved	2	Reserved
8	RandomData	32	Requester-provided random data.

Offset	Field	Size in bytes	Value
40	ExchangeData	D	DHE public information generated by the Requester. 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.
40 + D	OpaqueDataLength	2	Size of the OpaqueData field that follows in bytes. Shall be 0 if no OpaqueData is provided.
42 + D	OpaqueData	OpaqueDataLength	If present, OpaqueData sent by the Requester. Used to indicate any parameters that Requester wishes to pass to the Responder as part of key exchange.

434 Successful KEY\_EXCHANGE\_RSP response message format

Offset	Field	Size in bytes	Value
0	SPDMVersion	1	V1.1 = 0x11
1	RequestResponseCode	1	0x64 = KEY_EXCHANGE_RSP

Offset	Field	Size in bytes	Value
2	Param1	1	HeartbeatPeriod  The value of this field shall be zero if Heartbeat is not supported. Otherwise, the value shall be in units of seconds. 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 (mutual authentication) without using the encapsulated request flow.</p> <p>Bit 1 - If set, Responder is requesting mutual authentication with the encapsulated request flow.</p> <p>Bit 2 - If set, Responder is requesting 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>Only one of Bit 0, Bit 1 and Bit 2 shall be set.</p> <p>For details on the encapsulated request flow or the optimized encapsulated request flow, see the <a href="#">GET_ENCAPSULATED_REQUEST request and ENCAPSULATED_REQUEST response messages</a> clause.</p>



Offset	Field	Size in bytes	Value
7	SlotIDParam	1	<p>Bit[7:4] = Reserved.</p> <p>Bit[3:0] = SlotID. The slot number of the certificate chain of the Requester to 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. 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>

Offset	Field	Size in bytes	Value
40 + D	MeasurementSummaryHash	H	<p>When the Responder does not support measurements ( <code>MEAS_CAP=00b</code> in <code>CAPABILITIES</code> response) or requested <code>Param1 =0</code>, this field shall be absent.</p> <p>When the requested <code>Param1 =1</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 <a href="#">index</a>.</p> <p>When the requested <code>Param1 =1</code> and there are no measurable components in the TCB required to generate this response, this field shall be <code>0</code>.</p> <p>When requested <code>Param1=0xFF</code>, this field is computed as the <code>hash(Concatenation(MeasurementBlock[0], MeasurementBlock[1], ..., MeasurementBlock[n]))</code> of all supported measurement blocks available in the measurement index range <code>0x01 - 0xFE</code>, concatenated in ascending index order. Indices with no associated measurements shall not be included in the hash calculation.</p>
40 + D + H	OpaqueDataLength	2	Size of the <code>OpaqueData</code> field that follows in bytes. Shall be <code>0</code> if no <code>OpaqueData</code> is provided.
42 + D + H	OpaqueData	OpaqueDataLength	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.

Offset	Field	Size in bytes	Value
42 + D + H + OpaqueDataLength	Signature	S	Signature over the transcript hash. S is the size of the asymmetric signing algorithm output the Responder selected via the last <code>ALGORITHMS</code> response message using the private key of the leaf certificate of the Responder. The construction of the transcript hash is defined in <a href="#">Transcript Hash for KEY_EXCHANGE_RSP</a> signature.
42 + D + H + OpaqueDataLength + S	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 <code>finished_key</code> as the secret key and using the negotiated hash algorithm as the hash function. The transcript hash shall be the <a href="#">Transcript Hash for KEY_EXCHANGE_RSP</a> HMAC. The <code>finished_key</code> shall be derived from the Response Direction Handshake Secret and is described in the <a href="#">finished_key</a> derivation clause. HMAC is described in <a href="#">RFC 2104</a>.</p> <p>If both the Requester and Responder set <code>HANDSHAKE_IN_THE_CLEAR_CAP</code> to 1, then this field shall be absent.</p>

**435 10.16.1 Mutual authentication**

436 To perform authentication of the Requester in the `KEY_EXCHANGE` flow, either the encapsulated request flow or the optimized encapsulated request flow shall be used. For details and illustration of this flow, see [GET\\_ENCAPSULATED\\_REQUEST request and ENCAPSULATED\\_REQUEST response messages](#).

437 The only messages that shall be encapsulated in this case are `GET_DIGESTS` , `DIGESTS` , `GET_CERTIFICATE` , and `CERTIFICATE` .

**438 10.16.2 Specifying Requester certificate for mutual authentication**

439 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.

- 440 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 and `SlotID` is set to 0, the Responder shall use a `ENCAPSULATED_RESPONSE_ACK` request with `Param2 = 0x02` and an 1-byte long `Encapsulated Request` field containing the `SlotID` value. This shall be interpreted by the Requester as a valid request indicating the slot number to be used, and the `SlotID` field in `KEY_EXCHANGE_RSP` shall be ignored.
- 441 If Bit 0 of `MutAuthRequested` is set, then mutual authentication shall be performed without exchanging any messages between `KEY_EXCHANGE_RSP` and `FINISH` request. The certificate chain of the Requester is determined by the value of `SlotID`. This is useful for Responders which have obtained a certificate chains of the Requester in a previous interaction.

## 442 10.17 FINISH request and FINISH\_RSP response messages

- 443 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 identify of each party to the exchanged keys and protect the entire handshake against manipulation by an active attacker. The [FINISH request message format](#) table shows the `FINISH` request message format and the [Successful FINISH\\_RSP response message format](#) table shows the `FINISH_RSP` response message format.

### 444 FINISH request message format

Offset	Field	Size in bytes	Value
0	<code>SPDMVersion</code>	1	V1.1 = 0x11
1	<code>RequestResponseCode</code>	1	0xE5 = FINISH
2	<code>Param1</code>	1	Bit 0 – If set, the Signature field is included. This bit shall be set when mutual authentication occurs. All other bits reserved.

Offset	Field	Size in bytes	Value
3	Param2	1	Slot ID. Only valid if Param1 = 0x01 , otherwise reserved. Slot number of the Requester Certificate Chain being 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	S	Signature over the transcript hash. S is the size of the asymmetric signing algorithm output the Responder selected via the last ALGORITHMS response message using the private key of the leaf certificate of the Requester. S is zero and field not present if Param1 = 0x00 . The construction of the transcript hash is defined in <a href="#">Transcript Hash for FINISH signature, mutual authentication</a> .

Offset	Field	Size in bytes	Value
4+S	RequesterVerifyData	H	<p>This field shall be an HMAC of the transcript hash using the <code>finished_key</code> as the secret key and using the negotiated hash algorithm as the hash function. For mutual authentication, the transcript hash shall be the <a href="#">Transcript Hash for FINISH HMAC, mutual authentication</a>. Otherwise, it shall be the <a href="#">Transcript Hash for FINISH HMAC, Responder-only authentication</a>. The <code>finished_key</code> shall be derived from Request Direction Handshake Secret and is described in the <a href="#">finished_key derivation</a> clauses. HMAC is described in <a href="#">RFC 2104</a>.</p>

445 **Successful FINISH\_RSP response message format**

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

Offset	Field	Size in bytes	Value
4	ResponderVerifyData	H	<p>Conditional field.</p> <p>If the Session Handshake Phase is encrypted and/or message authenticated (i.e., 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 mutual authentication, the transcript shall be the <a href="#">Transcript Hash for <code>FINISH_RSP</code> HMAC, mutual authentication</a>. Otherwise, the transcript hash shall be the <a href="#">Transcript Hash for <code>FINISH_RSP</code> HMAC, Responder Only authentication</a>. 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="#">RFC 2104</a>.</p>

**446 10.17.1 Transcript hash calculation rules**

447 The transcript hash is calculated by hashing the concatenation of the prescribed full messages or message fields in order. For messages that are encrypted, the plaintext messages shall be used in calculating the transcript hash.

448 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.OpacityData` ).

**449 Transcript hash for KEY\_EXCHANGE\_RSP signature**

1. [GET\_VERSION].\* (if issued)
2. [VERSION].\* (if issued)
3. [GET\_CAPABILITIES].\* (if issued)
4. [CAPABILITIES].\* (if issued)
5. [NEGOTIATE\_ALGORITHMS].\* (if issued)
6. [ALGORITHMS].\* (if issued)
7. Hash of the specified certificate chain in DER format (i.e., KEY\_EXCHANGE Param2)
8. [KEY\_EXCHANGE].\*
9. [KEY\_EXCHANGE\_RSP].\* except the `Signature` and `ResponderVerifyData` fields.

**450 Transcript hash for KEY\_EXCHANGE\_RSP HMAC**

1. [GET\_VERSION].\* (if issued)
2. [VERSION].\* (if issued)
3. [GET\_CAPABILITIES].\* (if issued)
4. [CAPABILITIES].\* (if issued)
5. [NEGOTIATE\_ALGORITHMS].\* (if issued)
6. [ALGORITHMS].\* (if issued)
7. Hash of the specified certificate chain in DER format (i.e., KEY\_EXCHANGE Param2)
8. [KEY\_EXCHANGE].\*
9. [KEY\_EXCHANGE\_RSP].\* except the `ResponderVerifyData` field.

**451 Transcript hash for FINISH signature, mutual authentication**

1. [GET\_VERSION].\* (if issued)
2. [VERSION].\* (if issued)
3. [GET\_CAPABILITIES].\* (if issued)
4. [CAPABILITIES].\* (if issued)
5. [NEGOTIATE\_ALGORITHMS].\* (if issued)
6. [ALGORITHMS].\* (if issued)
7. Hash of the specified certificate chain in DER format (i.e., KEY\_EXCHANGE Param2)
8. [KEY\_EXCHANGE].\*
9. [KEY\_EXCHANGE\_RSP].\*
10. Hash of the specified certificate chain in DER format (i.e., FINISH Param2)
11. [FINISH].SPDM Header Fields

**452 Transcript hash for FINISH HMAC, Responder-only authentication**

1. [GET\_VERSION].\* (if issued)
2. [VERSION].\* (if issued)
3. [GET\_CAPABILITIES].\* (if issued)
4. [CAPABILITIES].\* (if issued)
5. [NEGOTIATE\_ALGORITHMS].\* (if issued)
6. [ALGORITHMS].\* (if issued)
7. Hash of the specified certificate chain in DER format (i.e., KEY\_EXCHANGE's request Param2)
8. [KEY\_EXCHANGE].\*



9. [KEY\_EXCHANGE\_RSP].\*
10. [FINISH].SPDM Header Fields

#### 453 Transcript hash for FINISH HMAC, mutual authentication

1. [GET\_VERSION].\* (if issued)
2. [VERSION].\* (if issued)
3. [GET\_CAPABILITIES].\* (if issued)
4. [CAPABILITIES].\* (if issued)
5. [NEGOTIATE\_ALGORITHMS].\* (if issued)
6. [ALGORITHMS].\* (if issued)
7. Hash of the specified certificate chain in DER format (i.e., KEY\_EXCHANGE's request Param2)
8. [KEY\_EXCHANGE].\*
9. [KEY\_EXCHANGE\_RSP].\*
10. Hash of the specified certificate chain in DER format (i.e., FINISH's Param2).
11. [FINISH].SPDM Header Fields
12. [FINISH].Signature

#### 454 Transcript hash for FINISH\_RSP HMAC, Responder-only authentication

1. [GET\_VERSION].\* (if issued)
2. [VERSION].\* (if issued)
3. [GET\_CAPABILITIES].\* (if issued)
4. [CAPABILITIES].\* (if issued)
5. [NEGOTIATE\_ALGORITHMS].\* (if issued)
6. [ALGORITHMS].\* (if issued)
7. Hash of the specified certificate chain in DER format (i.e., KEY\_EXCHANGE's request Param2)
8. [KEY\_EXCHANGE].\*
9. [KEY\_EXCHANGE\_RSP].\*
10. [FINISH].\*
11. [FINISH\_RSP].SPDM Header fields

#### 455 Transcript hash for FINISH\_RSP HMAC, mutual authentication

1. [GET\_VERSION].\* (if issued)
2. [VERSION].\* (if issued)
3. [GET\_CAPABILITIES].\* (if issued)
4. [CAPABILITIES].\* (if issued)
5. [NEGOTIATE\_ALGORITHMS].\* (if issued)
6. [ALGORITHMS].\* (if issued)
7. Hash of the specified certificate chain in DER format (i.e., KEY\_EXCHANGE's request Param2)
8. [KEY\_EXCHANGE].\*
9. [KEY\_EXCHANGE\_RSP].\*
10. Hash of the specified certificate chain in DER format (i.e., FINISH's Param2).
11. [FINISH].\*
12. [FINISH\_RSP].SPDM Header fields

#### 456 When multiple session keys are being established between the same Requester and Responder pair, Signature over

Transcript HASH during FINISH request is computed using only the corresponding KEY\_EXCHANGE, KEY\_EXCHANGE\_RSP and FINISH request parameters.

## 457 **10.18 PSK\_EXCHANGE request and PSK\_EXCHANGE\_RSP response messages**

---

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

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

460 A Requester may be paired with multiple Responders. Likewise, a Responder may be paired with multiple Requesters. A pair of Requester and Responder may be provisioned with one or more PSKs. If both endpoints can act as Requester or Responder, then the endpoints shall use different PSKs for each role.

461 An endpoint may act as a Requester to one device and simultaneously a Responder to another device. It is the responsibility of the transport layer to identify the peer and establish communication between the two endpoints, before the PSK-based session key exchange starts.

462 The PSK may be provisioned in a trusted environment, for example, during the secure manufacturing process. In an untrusted environment, the PSK may be agreed upon between the two endpoints using a secure protocol. The mechanism for PSK provisioning is out of scope of this specification. The size of the provisioned PSK is determined by the requirement of security strength of the application, but should be at least 128 bits and recommended to be 256 bits or larger, to resist dictionary attacks especially when the Requester and Responder cannot both contribute sufficient entropy during the exchange. If the negotiated capabilities and algorithms are provisioned to both endpoints alongside the PSK, then the Requester shall not issue `GET_CAPABILITIES` and `NEGOTIATE_ALGORITHMS`.

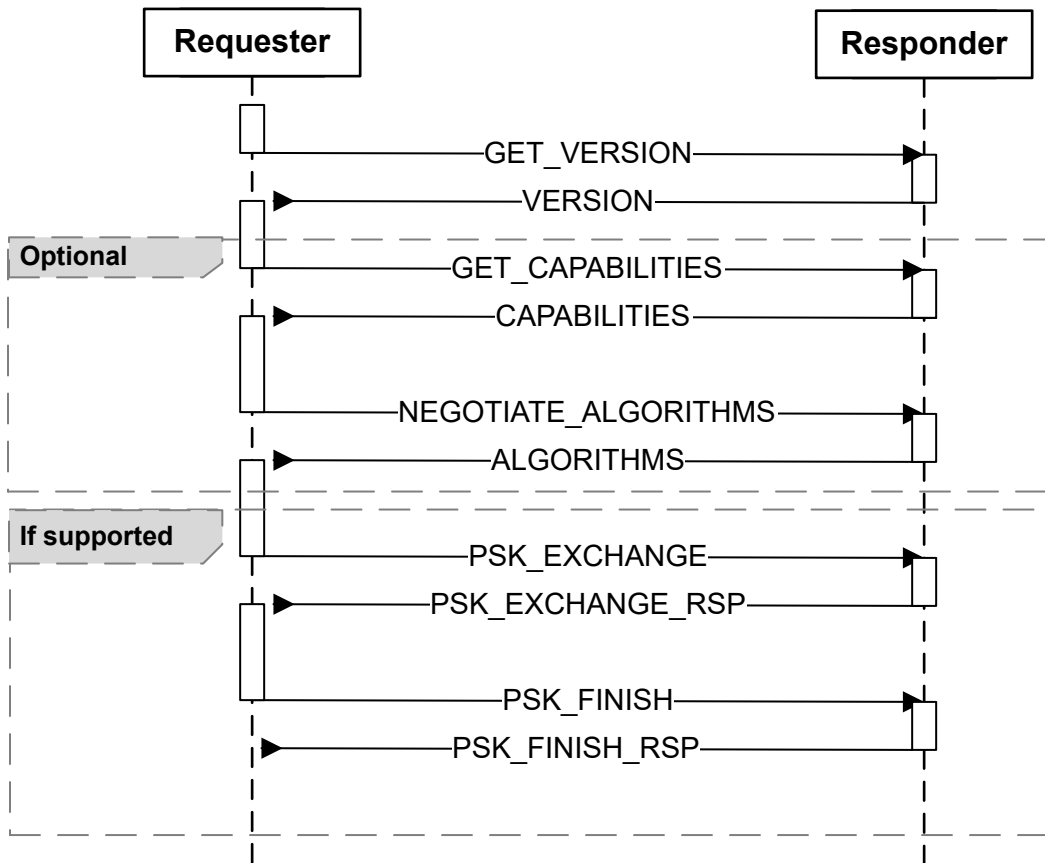
463 Two message pairs are defined for this option: `PSK_EXCHANGE / PSK_EXCHANGE_RSP` and `PSK_FINISH / PSK_FINISH_RSP`.

464 The `PSK_EXCHANGE` message carries three responsibilities:

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

465 **PSK\_EXCHANGE: Example**

466



467 **PSK\_EXCHANGE request message format**

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	V1.1 = 0x11
1	<a href="#">RequestResponseCode</a>	1	0xE6 = PSK_EXCHANGE

Offsets	Field	Size in bytes	Value
2	Param1	1	<p>Requested measurement summary hash Type:</p> <p>0x0 . No measurement summary hash.</p> <p>0x1 . TCB measurement hash.</p> <p>0xFF . All measurements hash.</p> <p>All other values reserved.</p> <p>When Responder does not support any measurements, Requester shall set this value to 0x0 .</p>
3	Param2	1	Reserved.
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. R shall be equal to or greater than H, where H is the size of the underlying HMAC used in the context of the Requester.
10	OpaqueDataLength	2	Length of OpaqueData in bytes.

Offsets	Field	Size in bytes	Value
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 the information belonging to 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.

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

- The Responder is provisioned with multiple PSKs and stores them in secure storage. The Requester uses PSKHint as an identifier to specify which PSK will be used in this session.
- The Responder does not store the value of the PSK, but can derive the PSK using PSKHint. For example, if the Responder has an immutable UDS (Unique Device Secret) in fuses, then during provisioning, a PSK may be derived from the UDS or its derivative and a non-secret salt known by the Requester. During session key establishment, the same salt 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.

469 The RequesterContext is the contribution of the Requester to session key derivation. It shall contain a nonce of at least 32 bytes to make sure that the derived session keys are ephemeral to mitigate against replay attacks. It is recommended that the Requester use random number as the nonce. If a random number generator is not available, the Requester may use a monotonic counter with protection against reset attacks. The RequesterContext may also contain other information from the Requester.

470 Upon receiving 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.

471 **PSK\_EXCHANGE\_RSP response message format**

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	V1.1 = 0x11
1	<a href="#">RequestResponseCode</a>	1	0x66 = PSK_EXCHANGE_RSP
2	Param1	1	HeartbeatPeriod The value of this field shall be zero if Heartbeat is not supported. Otherwise, the value shall be in units of seconds. 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 ResponderContext in bytes.
10	OpaqueDataLength	2	Length of OpaqueData in bytes.

Offsets	Field	Size in bytes	Value
12	MeasurementSummaryHash	H	<p>When the Responder does not support measurements ( <code>MEAS_CAP=00b</code> in <code>CAPABILITIES</code> response) or requested <code>Param1 =0</code>, this field shall be absent.</p> <p>When the requested <code>Param1 =1</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 <a href="#">index</a>.</p> <p>When the requested <code>Param1 =1</code> and there are no measurable components in the TCB required to generate this response, this field shall be <code>0</code>.</p> <p>When requested <code>Param1=0xFF</code>, this field is computed as the <code>hash(Concatenation(MeasurementBlock[0], MeasurementBlock[1], ..., MeasurementBlock[n]))</code> of all supported measurement blocks available in the measurement index range <code>0x01 - 0xFE</code>, concatenated in ascending index order. Indices with no associated measurements shall not be included in the hash calculation.</p>
12 + H	ResponderContext	Q	Context of the Responder. Optional. If present, shall include a nonce and/or information belonging to the Responder.
12 + H + Q	OpaqueData	OpaqueDataLength	Optional. If present, the OpaqueData 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.
12 + H + Q + OpaqueDataLength	ResponderVerifyData	H	Data to be verified by the Requester using the <code>finished_key</code> of the Responder.

472 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. Because the Responder may be a

constrained device that is not able to generate a nonce, `ResponderContext` is optional. However, the Responder is required to use `ResponderContext` if it can generate a nonce.

473 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 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.

474 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 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.

475 To calculate `ResponderVerifyData`, the Responder calculates a HMAC. The HMAC key is the `finished_key` of the Responder. The data is the hash of the concatenation of specific messages, listed in [ResponderVerifyData messages](#), needed to fully establish the new session between the Requester and the Responder. For messages that are encrypted, the plaintext messages shall be used in calculating the hash.

#### 476 **ResponderVerifyData messages**

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

477 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 aborts 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.

## 478 **10.19 PSK\_FINISH request and PSK\_FINISH\_RSP response messages**

479 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 the `PSK_FINISH` only if `ResponderContext` is present in `PSK_EXCHANGE_RSP`.

#### 480 **PSK\_FINISH request message format**



Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	V1.1 = 0x11
1	RequestResponseCode	1	0xE7 = PSK_FINISH
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.

481 To calculate RequesterVerifyData, the Requester calculates a 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 shall be used in calculating the hash.

1. [GET\_VERSION].\* (if issued)
2. [VERSION].\* (if issued)
3. [GET\_CAPABILITIES].\* (if issued)
4. [CAPABILITIES].\* (if issued)
5. [NEGOTIATE\_ALGORITHMS].\* (if issued)
6. [ALGORITHMS].\* (if issued)
7. [PSK\_EXCHANGE].\*
8. [PSK\_EXCHANGE\_RSP].\*
9. [PSK\_FINISH].\* except the RequesterVerifyData field

482 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.

483 **Successful PSK\_FINISH\_RSP response message format**

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

## 484 10.20 HEARTBEAT request and HEARTBEAT\_ACK response messages

485 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 if no other messages are received in this secure session in the HeartbeatPeriod. 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 of the scope of this specification. The Requester may retry HEARTBEAT requests.

486 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.

487 Each secure session shall track the heartbeat period independently of other sessions between the same Requester-Responder pair.

488 For further details of session termination, see [Session termination phase](#).

489 The [HEARTBEAT request message format](#) describes the message format.

### 490 HEARTBEAT request message format

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	V1.1 = 0x11
1	<a href="#">RequestResponseCode</a>	1	0xE8 = HEARTBEAT Request
2	Param1	1	Reserved.
3	Param2	1	Reserved.

491 The [HEARTBEAT\\_ACK response message format](#) describes the format for the Heartbeat Response.

### 492 HEARTBEAT\_ACK response message format

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	V1.1 = 0x11
1	<a href="#">RequestResponseCode</a>	1	0x68 = HEARTBEAT_ACK Response
2	Param1	1	Reserved.
3	Param2	1	Reserved.

493 **10.20.1 Heartbeat additional information**

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

495 **10.21 KEY\_UPDATE request and KEY\_UPDATE\_ACK response messages**

496 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 update session keys in the direction of the request only. Because the Responder can also send this request, it is possible that two simultaneous key updates, one for each direction, can occur. However, only one `KEY_UPDATE` request for a single direction shall occur. Until the session key update synchronization successfully completes, subsequent `KEY_UPDATE` request for the same direction shall be considered a retry of the original `KEY_UPDATE` request.

497 **KEY\_UPDATE request message format**

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	V1.1 = 0x11
1	<a href="#">RequestResponseCode</a>	1	0xE9 = KEY_UPDATE Request
2	Param1	1	Key Operation. See <a href="#">KEY_UPDATE Operations Table</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.

498 **KEY\_UPDATE\_ACK response message format**

Offsets	Field	Size in bytes	Value
0	SPDMVersion	1	V1.1 = 0x11
1	<a href="#">RequestResponseCode</a>	1	0x69 = KEY_UPDATE_ACK Response

Offsets	Field	Size in bytes	Value
2	Param1	1	Key Operation. This field shall reflect the Key Operation field of the request.
3	Param2	1	Tag. This field shall reflect the Tag number in the <code>KEY_UPDATE</code> request.

#### 499 **KEY\_UPDATE operations**

Value	Operation	Description
0	Reserved	Reserved
1	UpdateKey	Update the single-direction key.
2	UpdateAllKeys	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

### 500 **10.21.1 Session key update synchronization**

501 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.

502 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 of the scope of this specification.

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

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

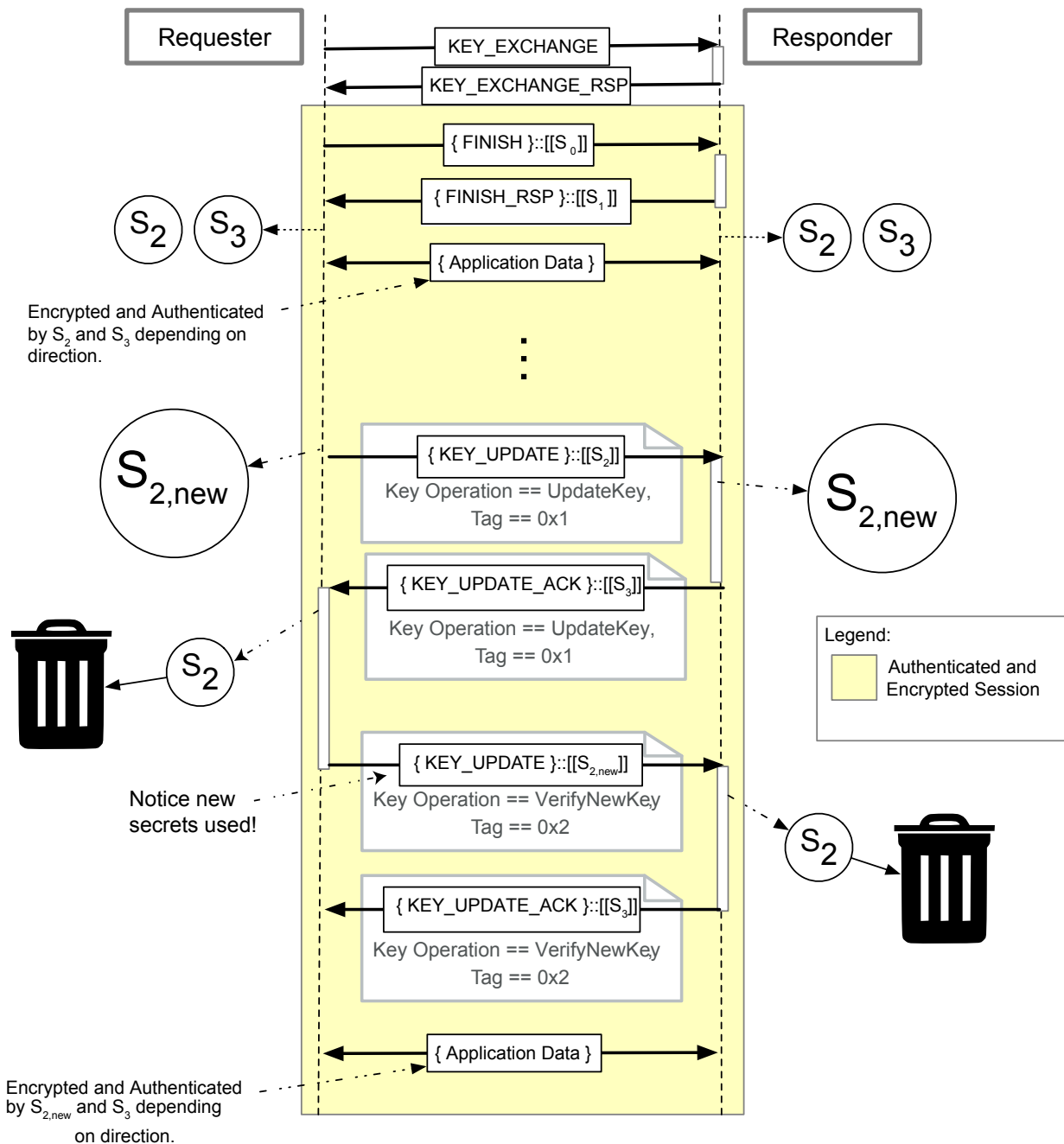
505 The sender and receiver shall use the new key on the `KEY_UPDATE` request with `VerifyNewKey` command and all subsequent commands until another key update is performed.

506 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.

507 If the sender has not received `KEY_UPDATE_ACK`, the sender may retry or end the session. The sender shall not proceed to the next step until successfully receiving the corresponding `KEY_UPDATE_ACK`.

- 508 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 may 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. Upon successful reception of the `KEY_UPDATE` with `VerifyNewKey` operation, the receiver can now discard the old session keys. After the sender successfully receives the corresponding `KEY_UPDATE_ACK` , the transport layer may start using the new keys.
- 509 In certain scenarios, the receiver may need additional time to process the `KEY_UPDATE` request such as processing data already in its buffer. Thus, the receiver may 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.
- 510 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.
- 511 The [KEY\\_UPDATE protocol example flow](#) figure 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.
- 512 **KEY\_UPDATE protocol example flow**

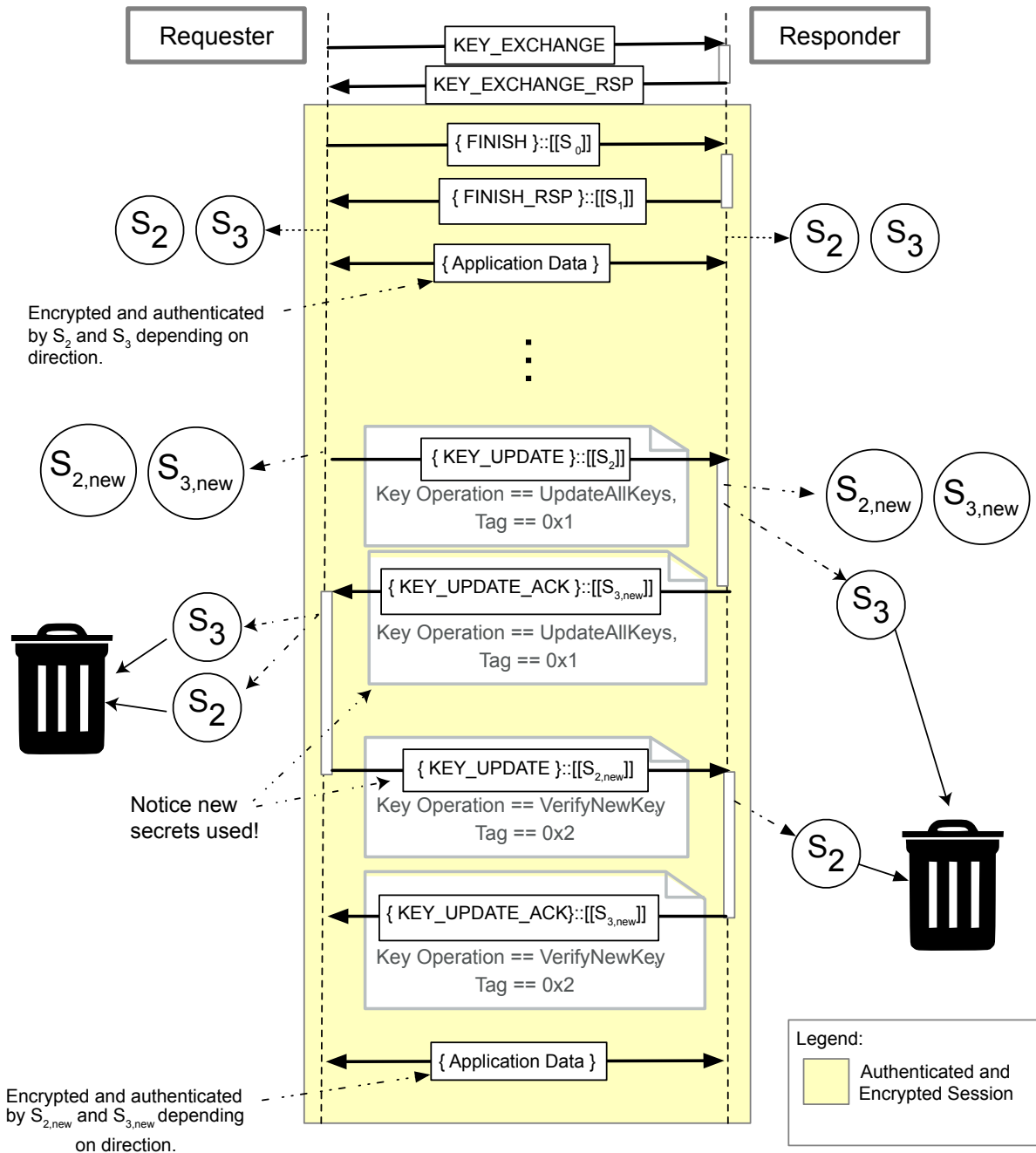
513



514 The [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.

515 **KEY\_UPDATE protocol change all keys example flow**

516



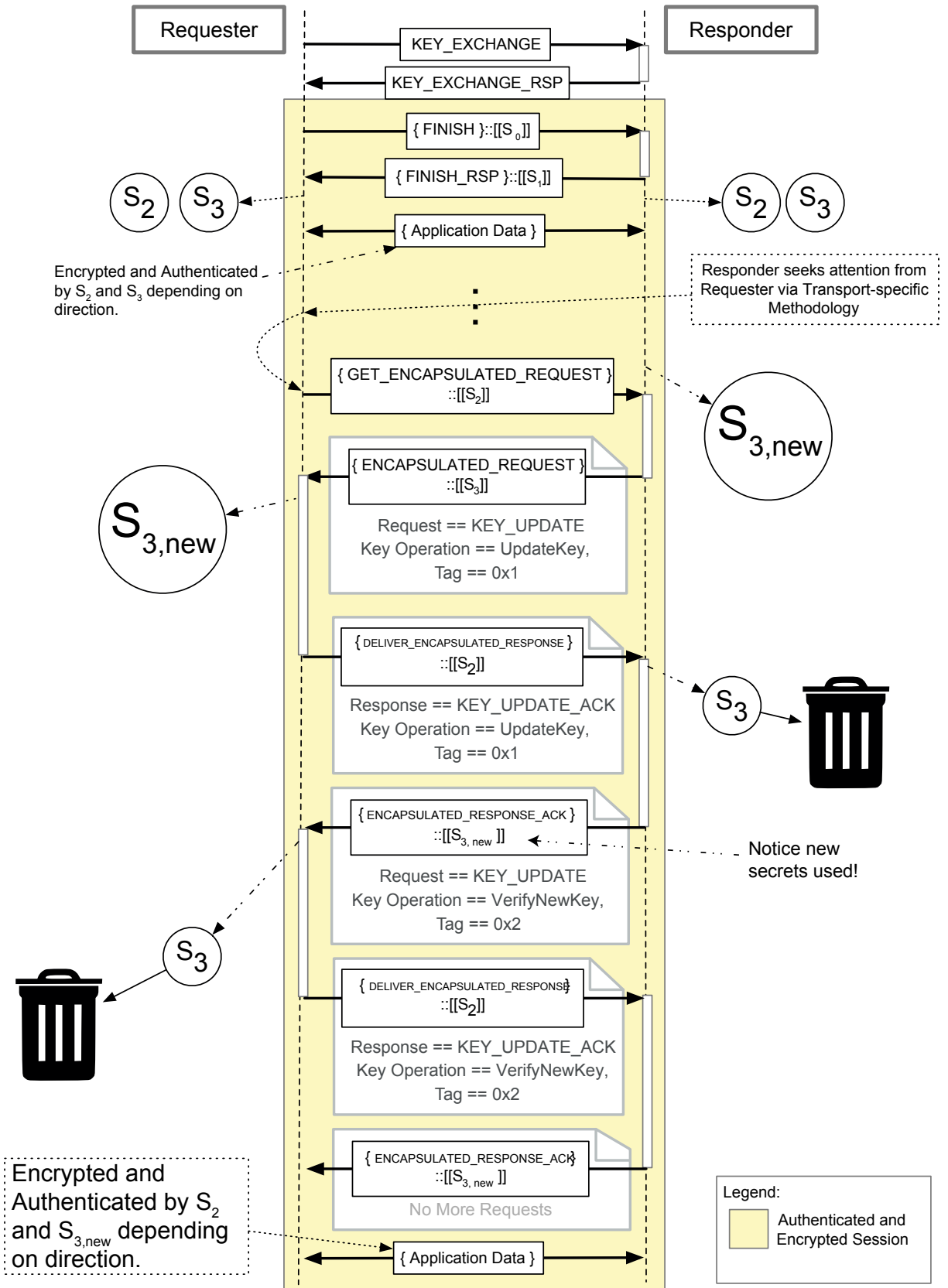
**517 10.21.2 KEY\_UPDATE transport allowances**

518 On some transports, bidirectional communication can occur asynchronously. On such transports, the transport may allow or disallow the `KEY_UPDATE` to be sent asynchronously without using the `GET_ENCAPSULATED_REQUEST` mechanism. The actual method to use should be defined by the transport and is outside the scope of this specification.

- 519 The [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 via a sideband 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.
- 520 **KEY\_UPDATE protocol example flow 2**



521



## 522 **10.22 GET\_ENCAPSULATED\_REQUEST request and ENCAPSULATED\_REQUEST response messages**

---

523 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. The [Mutual authentication key exchange](#) figure and [Optimized mutual authentication key exchange example](#) figure are examples that illustrate the use of this scheme.

524 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. The request message format is described in [GET\\_ENCAPSULATED\\_REQUEST request format table](#). The Responder shall use the same SPDM version the Requester used.

### 525 **10.22.1 Encapsulated request flow**

526 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 [Mutual authentication key exchange](#).

527 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`.

### 528 **10.22.2 Optimized encapsulated request flow**

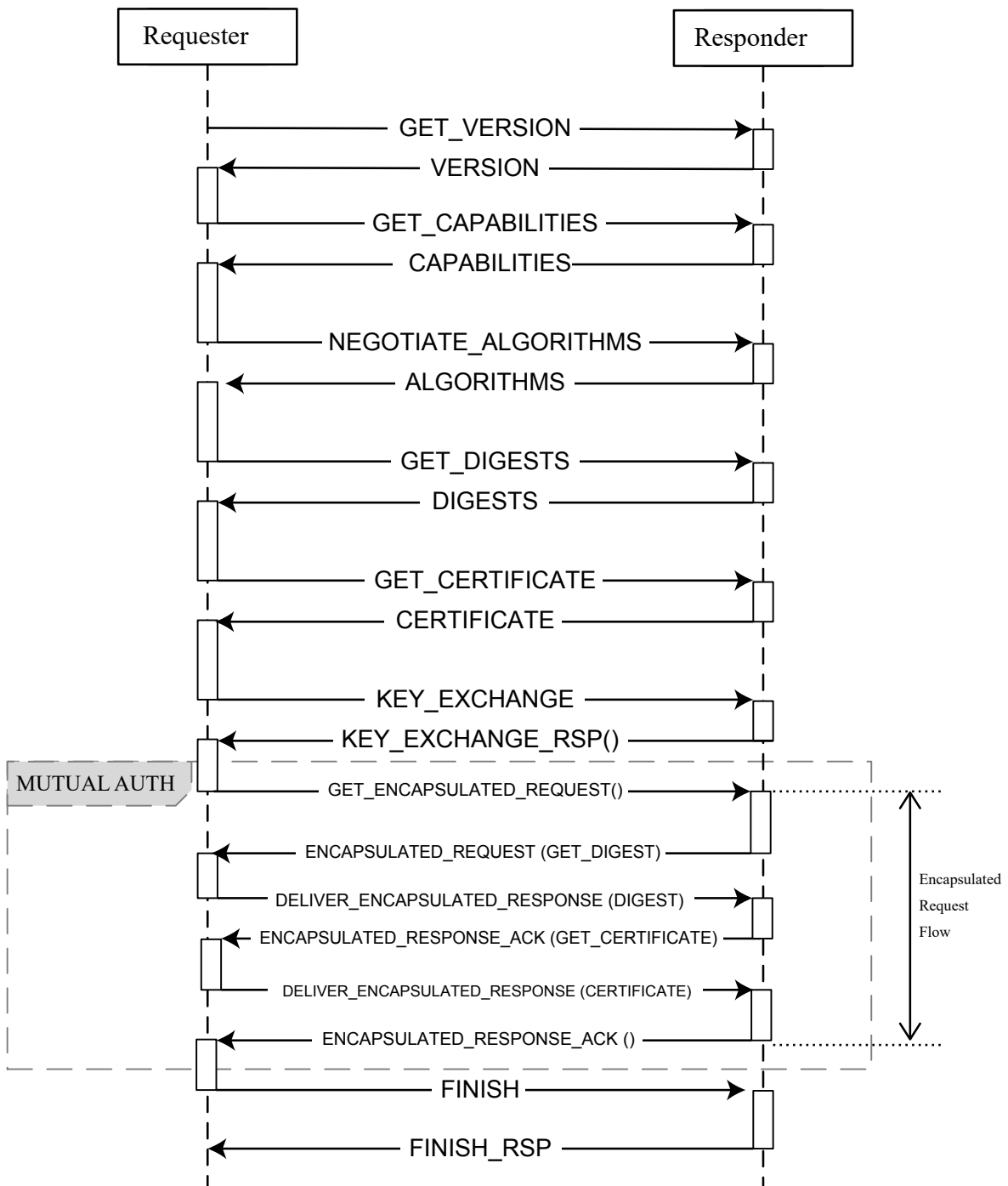
529 The optimized encapsulated request flow is similar to the encapsulated request flow but without the need of `GET_ENCAPSULATED_REQUEST`. This is because the encapsulated request accompanies one of the `Session-Secrets-Exchange` responses; thereby, removing the necessity on the Requester from issuing a `GET_ENCAPSULATED_REQUEST`. When the Responder includes an encapsulated requests with a `Session-Secrets-Exchange` response, the optimized encapsulated request flow shall start. This is also illustrated in [Optimized mutual authentication key exchange](#).

530 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`, `GET_VERSION` or `Session-Secrets-Exchange`, then the Responder should respond with `ErrorCode=RequestInFlight`.

---

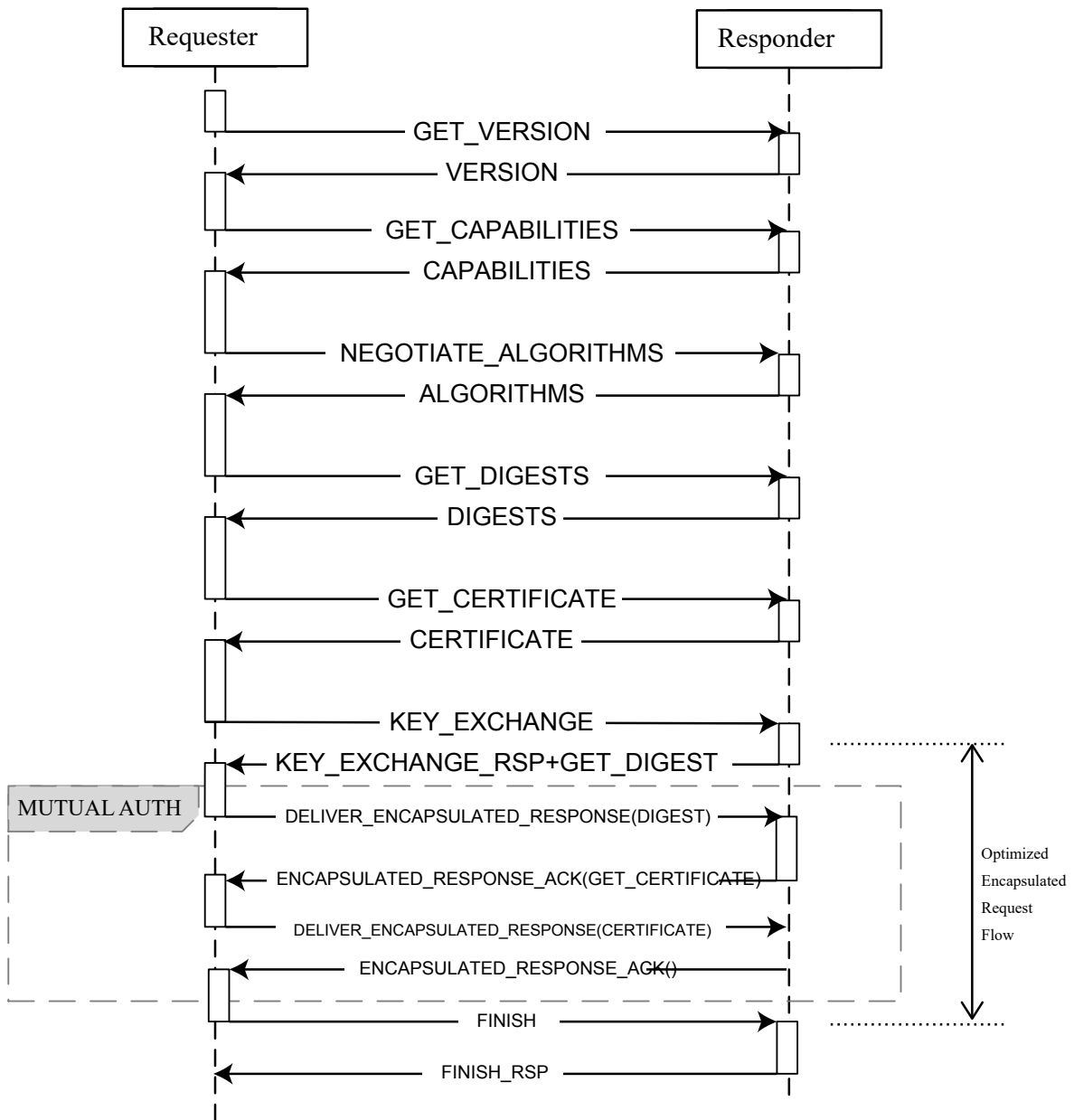
531 **Mutual authentication key exchange example**

532



533 **Optimized mutual authentication key exchange example**

534



535 GET\_ENCAPSULATED\_REQUEST request message format

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

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

536 The [ENCAPSULATED\\_REQUEST response message format](#) describes the format this response.

537 **ENCAPSULATED\_REQUEST response message format**

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

### 538 10.22.3 Triggering GET\_ENCAPSULATED\_REQUEST

539 Once a session has been established, the Responder may 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.

### 540 10.22.4 Additional constraints

541 The GET\_ENCAPSULATED\_REQUEST and ENCAPSULATED\_REQUEST messages shall only be allowed to encapsulate certain  
 requests in certain scenarios. For details on these constraints, see the [Session](#), [Basic mutual authentication](#), and  
[KEY\\_UPDATE request and KEY\\_UPDATE\\_ACK response messages](#) clauses.

## 542 10.23 DELIVER\_ENCAPSULATED\_RESPONSE request and ENCAPSULATED\_RESPONSE\_ACK response messages

543 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.

544 Furthermore, if there are additional requests from the Responder, the Responder shall provide the next request in the  
 ENCAPSULATED\_RESPONSE\_ACK response message.

545 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.

546 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.

547 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](#) for more details.

548 The request message format is described in DELIVER\_ENCAPSULATED\_RESPONSE Request Message Format Table.

### 549 DELIVER\_ENCAPSULATED\_RESPONSE request message format

Offsets	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11

Offsets	Field	Size (bytes)	Value
1	<a href="#">RequestResponseCode</a>	1	0xEB = DELIVER_ENCAPSULATED_RESPONSE Request
2	Param1	1	Request ID. The Requester shall use the same Request ID as provided by the Responder in the corresponding ENCAPSULATED_REQUEST or ENCAPSULATED_RESPONSE_ACK .
3	Param2	1	Reserved.
4	Encapsulated Response	Variable	SPDM Response Message. The value of this field shall represent a valid SPDM response message. The length of this field is dependent on the SPDM Response message. The field shall start with the 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.

550 The [ENCAPSULATED\\_RESPONSE\\_ACK response message format](#) describes the response message format.

551 **ENCAPSULATED\_RESPONSE\_ACK response message format**

Offsets	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.1 = 0x11
1	<a href="#">RequestResponseCode</a>	1	0x6B = ENCAPSULATED_RESPONSE_ACK

Offsets	Field	Size (bytes)	Value
2	Param1	1	<p>Request ID.</p> <p>If a request is encapsulated ( Param2 = 0x01 ) this field should contain a unique, non-zero number to help the Responder match response to request. Otherwise, this field shall be 0x00 .</p>
3	Param2	1	<p>Payload Type.</p> <p>If set to 0x00 no request message is encapsulated and the Encapsulated_Request field is absent.</p> <p>If set to 0x01 the Encapsulated_Request field follows.</p> <p>If set to 0x02 a 1-byte Encapsulated_Request field follows containing the slot number corresponding to the certificate chain the Requester shall authenticate against.</p> <p>All other values Reserved.</p>



Offsets	Field	Size (bytes)	Value
4	Encapsulated Request	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 Encapsulated Request 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 slot number corresponding to the certificate chain the Requester shall authenticate against. The field size shall be 1 Byte.</p> <p>If Param2 = 0x00 , this field shall be absent.</p>

**552 10.23.1 Additional information**

553 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 may cause additional mistakes to occur.

554 In general, the response timing for ENCAPSULATED\_RESP\_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, shall adhere to CT timing rules when it has a subsequent request. The Responder may return ErrorCode=ResponseNotReady .

555 The DELIVER\_ENCAPSULATED\_RESPONSE and ENCAPSULATED\_RESPONSE\_ACK messages shall only be allowed to

encapsulate certain requests in certain scenarios. For details on these constraints, see [Session](#), [Basic mutual authentication](#), and [KEY\\_UPDATE request and KEY\\_UPDATE\\_ACK response messages](#) clauses.

## 556 10.24 END\_SESSION request and END\_SESSION\_ACK response messages

557 This request shall terminate a session. Further communication between the Requester and Responder using the same session ID shall be prohibited. See [Session termination phase](#) clause for details.

558 The [END\\_SESSION request message format](#) table describes this format.

### 559 END\_SESSION request message format

Offset	Value	Field	Description
0	SPDMVersion	1	V1.1 = 0x11
1	<a href="#">RequestResponseCode</a>	1	0xEC = END_SESSION
2	Param1	1	See the <a href="#">End session request attributes</a> table.
3	Param2	1	Reserved.

### 560 End session request attributes

Offset	Value	Field	Description
0	0	Negotiated State Preservation Indicator	If the Responder supports Negotiated State caching ( <code>CACHE_CAP=1</code> ), the Responder shall preserve the Negotiated State.
0	1	Negotiated State Preservation Indicator	If the Responder supports Negotiated State caching, the Responder shall also clear the Negotiated State as part of session termination.
[7:1]	Reserved	Reserved	Reserved.

561 The [END\\_SESSION\\_ACK response message format](#) describes the response message.

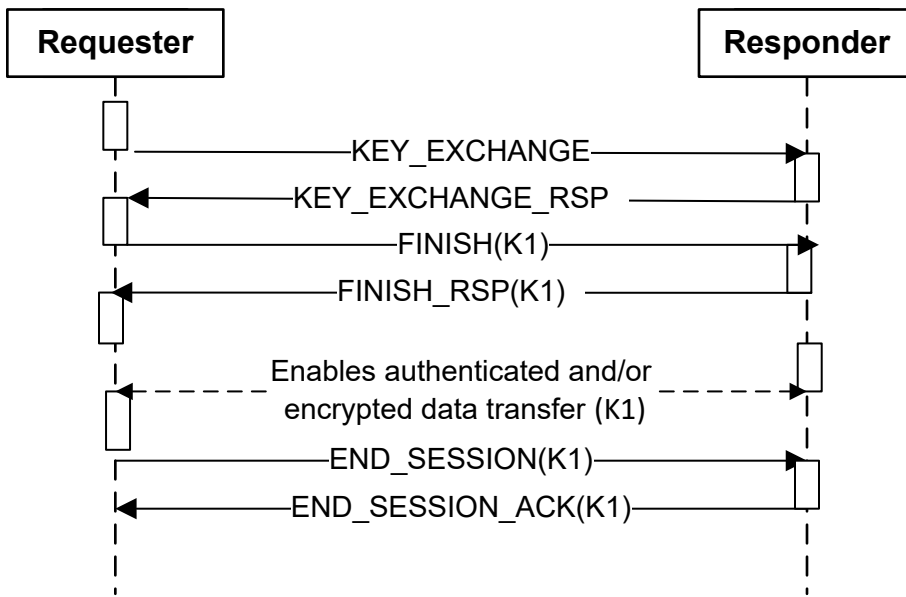
### 562 END\_SESSION\_ACK response message format

Offset	Value	Field	Description
0	SPDMVersion	1	V1.1 = 0x11

Offset	Value	Field	Description
1	RequestResponseCode	1	0x6C = END_SESSION_ACK
2	Param1	1	Reserved.
3	Param2	1	Reserved.

563 **END\_SESSION** protocol flow

564

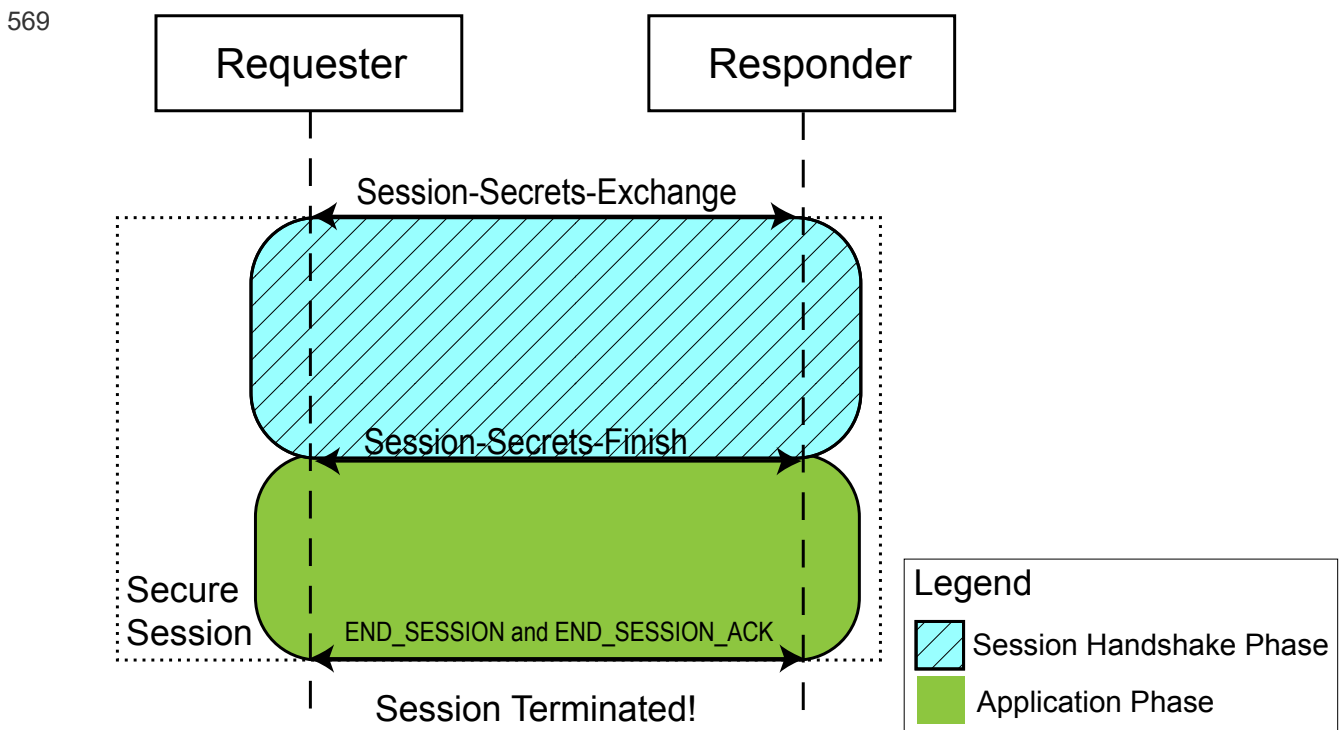


## 565 11 Session

566 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.

567 There are three phases in a session, as [Session phases](#) shows: the handshake, the application, and termination.

### 568 Session phases



### 570 11.1 Session handshake phase

571 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](#).

572 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.

573 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. 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.

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

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

## 576 11.2 Application phase

---

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

578 During this phase, a Requester can send SPDM messages such as `GET_MEASUREMENTS`. 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.

579 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.

## 580 11.3 Session termination phase

---

581 This phase signals the end of the Application phase and the enactment of internal clean-up procedures by the endpoints. Requesters and Responders may have various reasons for terminating a session, outside the scope of this specification.

582 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 may simply be an internal phase with no explicit SPDM messages sent or received.

583 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 may have other internal data associated with a session that they should also clean up.

## 584 11.4 Simultaneous active sessions

---

585 If a Responder supports key exchanges, the maximum number of simultaneous active sessions shall be a minimum of one. If the `KEY_EXCHANGE` or `PSK_EXCHANGE` request will exceed the maximum number of simultaneous active sessions of the Responder, the Responder shall respond with an `ErrorCode = SessionLimitExceeded`.

586 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. Since these two sessions are distinct and separate, the two endpoints should ensure they do not mix sessions. To ensure proper session handling, each endpoint should 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 may use information at the transport layer to further ensure proper handling of sessions.

## 587 **11.5 Records and session ID**

---

- 588 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.
- 589 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.

## 590 12 Key schedule

591 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 `HMAC` as defined by [RFC2104](#) and `HKDF-Expand` as described in [RFC5869](#). SPDM defines the following additional functions:

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

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

### 593 BinConcat details

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

594 If Context is NULL, then `BinConcat` is the concatenation of the first three components only.

595 The following table shows the value of the 8-byte version text for SPDM version 1.1. Hexadecimal equivalents are shown in parentheses for clarity.

### 596 Version details

SPDM version	byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7
SPDM 1.1	's' ( 0x73 )	'p' ( 0x70 )	'd' ( 0x64 )	'm' ( 0x6D )	'1' ( 0x31 )	':' ( 0x2E )	'1' ( 0x31 )	space ( 0x20 )

597 Note that the eighth byte of the version text is a space ( 0x20 ).

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

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

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

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

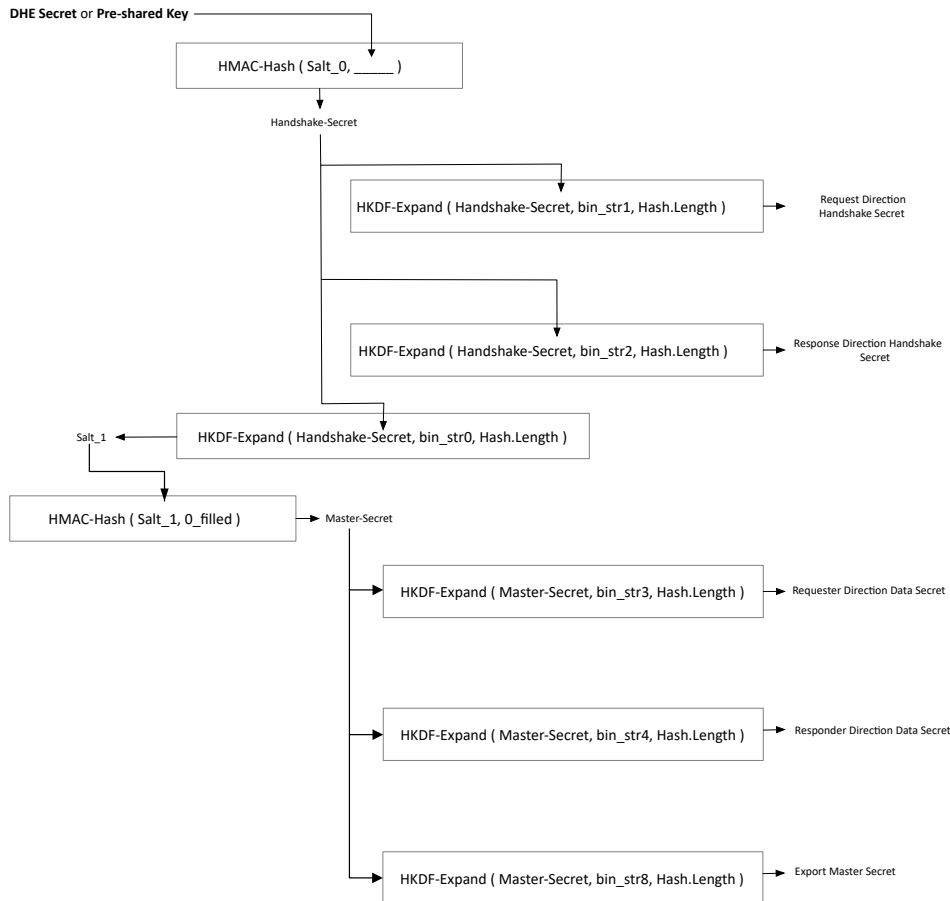
600 where IKM is the Input Keying Material and HMAC-Hash uses `HMAC` as defined in [RFC2104](#).

601 For HKDF-Expand and HMAC-Hash, 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.

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

603 **Key schedule**

604



605 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.

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

607 **Key schedule**

Variable	Definition
Salt_0	A zero filled array of Hash.Length length.



Variable	Definition
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 <code>KEY_EXCHANGE/KEY_EXCHANGE_RSP</code>
Pre-shared Key	PSK

608 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 labels defined above have all been chosen to fit within this limit.

## 609 12.1 Transcript hash in key derivation

610 There are two transcript hashes used in the key schedule, namely, **TH1** and **TH2**.

## 611 12.2 TH1 definition

612 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. `[GET_VERSION].*` (if issued)
2. `[VERSION].*` (if issued)
3. `[GET_CAPABILITIES].*` (if issued)
4. `[CAPABILITIES].*` (if issued)
5. `[NEGOTIATE_ALGORITHMS].*` (if issued)
6. `[ALGORITHMS].*` (if issued)
7. Hash of the specified certificate chain in DER format (i.e., `KEY_EXCHANGE Param2`)
8. `[KEY_EXCHANGE].*`
9. `[KEY_EXCHANGE_RSP].*` except the `ResponderVerifyData` field

613 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. `[GET_VERSION].*` (if issued)
2. `[VERSION].*` (if issued)
3. `[GET_CAPABILITIES].*` (if issued)

4. [CAPABILITIES].\* (if issued)
5. [NEGOTIATE\_ALGORITHMS].\* (if issued)
6. [ALGORITHMS].\* (if issued)
7. [PSK\_EXCHANGE].\*
8. [PSK\_EXCHANGE\_RSP].\* except the ResponderVerifyData field

## 614 12.3 TH2 definition

---

615 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. [GET\_VERSION].\* (if issued)
2. [VERSION].\* (if issued)
3. [GET\_CAPABILITIES].\* (if issued)
4. [CAPABILITIES].\* (if issued)
5. [NEGOTIATE\_ALGORITHMS].\* (if issued)
6. [ALGORITHMS].\* (if issued)
7. Hash of the specified certificate chain in DER format (i.e., KEY\_EXCHANGE Param2)
8. [KEY\_EXCHANGE].\*
9. [KEY\_EXCHANGE\_RSP].\*
10. Hash of the specified certificate chain in DER format (i.e., FINISH's Param2). (Valid only in mutual authentication)
11. [FINISH].\*
12. [FINISH\_RSP].\*

616 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. [GET\_VERSION].\* (if issued)
2. [VERSION].\* (if issued)
3. [GET\_CAPABILITIES].\* (if issued)
4. [CAPABILITIES].\* (if issued)
5. [NEGOTIATE\_ALGORITHMS].\* (if issued)
6. [ALGORITHMS].\* (if issued)
7. [PSK\_EXCHANGE].\*
8. [PSK\_EXCHANGE\_RSP].\*
9. [PSK\_FINISH].\* (if issued)
10. [PSK\_FINISH\_RSP].\* (if issued)

## 617 **12.4 Key schedule major secrets**

---

618 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 ( $S_3$ )

619 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.

### 620 **12.4.1 Request-direction handshake secret**

621 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`.

### 622 **12.4.2 Response-direction handshake secret**

623 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`.

### 624 **12.4.3 Requester-direction data secret**

625 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.

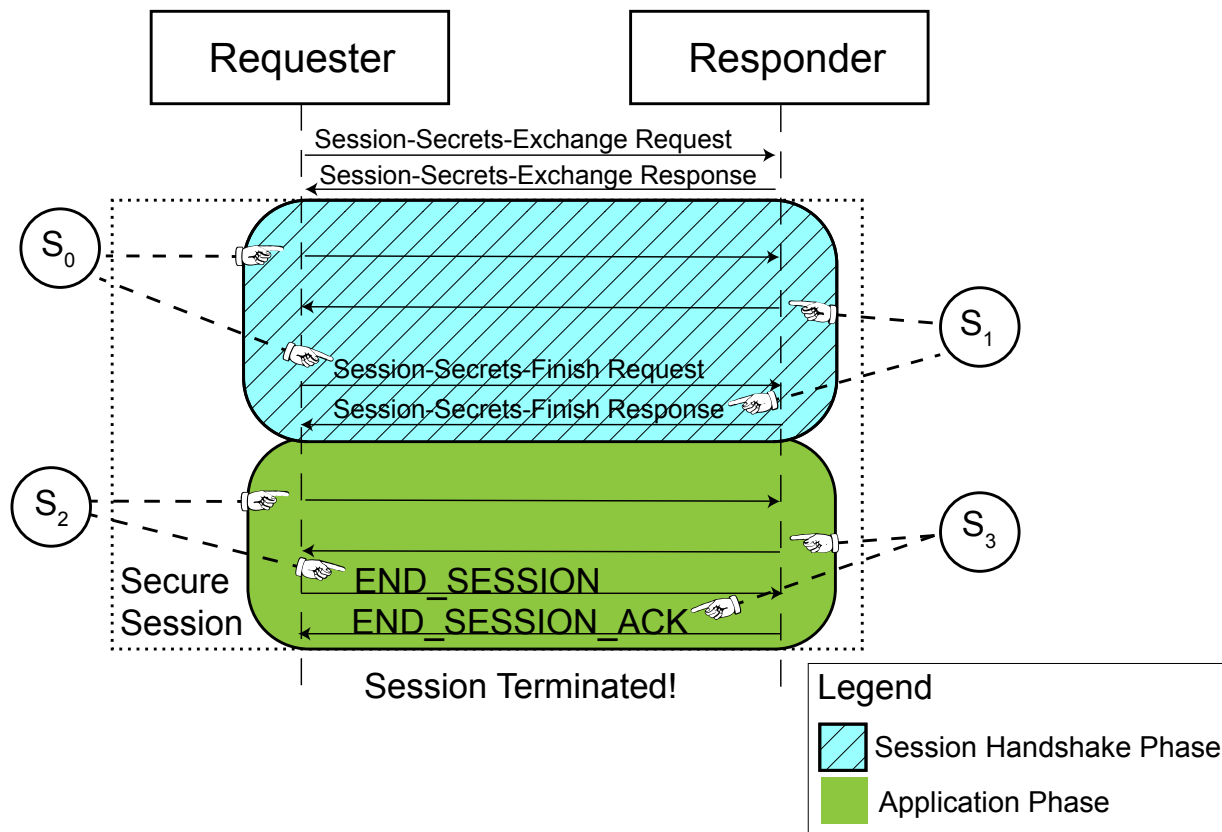
### 626 **12.4.4 Responder-direction data secret**

627 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.

628 The [Secrets Usage Figure](#) illustrates where each of the major secrets are used as described previously.

629 **Secrets usage**

630



## 631 12.5 Encryption key and IV derivation

632 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);
```

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

## 634 12.6 finished\_key derivation

635 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);
```

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

## 637 12.7 Deriving additional keys from the Export Master Secret

---

638 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);
```

## 639 12.8 Major secrets update

---

640 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.

641 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);
```

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

## 643 13 Application data

644 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.

645 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);
```

646 where

Value	Description
AEAD_Encrypt	Function that fully encrypts the <code>plaintext</code> , computes the MAC across both the <code>associated_data</code> and <code>plaintext</code> , and produces the <code>ciphertext</code> , which includes the MAC.
AEAD_Decrypt	Function that verifies the MAC and if validation is successful, fully decrypts the <code>ciphertext</code> and produces the original <code>plaintext</code> .
encryption_key	Derived encryption key for the respective direction. For details, see the <a href="#">Key schedule</a> clause.
nonce	Nonce computation. For details, see the <a href="#">Nonce derivation</a> clause.
associated_data	Associated data.
plaintext	Data to encrypt.
ciphertext	Data to decrypt.

### 647 13.1 Nonce derivation

648 Certain AEAD ciphers have specific requirements on nonce construction, as their security properties may be compromised by the accidental reuse of a nonce value. Implementations should follow the requirements, such as those provided in [RFC5116](#) for nonce derivation.

## 649 **14 ANNEX A (informative) TLS 1.3**

---

650 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.

651 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 [DTLS 1.3](#), which at the time of this specification is still in draft form.

## 652 15 ANNEX B (normative) Leaf certificate example

653 The [Leaf certificate example](#) shows an example leaf certificate:

### 654 Leaf certificate example

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



```
36:02:20:1a:ea:3a:59:ca:1e:bc:6d:6e:61:79:af:a2:05:7c:  
7d:da:41:a9:45:6d:cb:04:49:43:e6:0b:a8:8d:cd:da:e
```

## 655 **16 ANNEX C (informative) Change log**

---

### 656 **16.1 Version 1.0.0 (2019-10-16)**

---

- Initial Release

### 657 **16.2 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.
- 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.

### 658 **16.3 Version 1.1.1 (2021-05-12)**

---

- Fix improper reference in `DMTFSpecMeasurementValue` field in "Measurement field format when MeasurementSpecification field is Bit 0 = DMTF" table.
  - Certificate digests in `DIGEST` calculation clarified.
  - Format of certificate in `CertChain` parameter of `CERTIFICATE` message clarified.
-

- Validity period of X.509v3 certificate clarified in [Required Fields](#)
- Clarify which algorithms in `NEGOTIATE_ALGORITHMS` or `ALGORITHMS` are for signature generation or verification.
- 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 that plaintext messages are used when calculating the transcript hash.
- `ERROR` responses are no longer required in most error scenarios.
- In `Sign()` and `Verify()` operations, referenced the correct fields in `ALGORITHMS`.
- Clarify which key to use in `Signature` fields of `KEY_EXCHANGE_RSP` and `FINISH`.
- Clarify messages to hash for `ResponderVerifyData` in `PSK_EXCHANGE_RSP`.

## 659 16.4 Version 1.1.2 (2022-03-09)

---

- Fix typo and inconsistency in description of `PSK_FINISH`.
- Clarified measurement specification related fields in `NEGOTIATE_ALGORITHMS` and `ALGORITHMS`.
- Changed Measurement Summary Hash concatenation function inputs.
- Clarified minimum timing for [HEARTBEAT request and HEARTBEAT\\_ACK response messages](#) to be sufficiently greater than `T1`. Removed command specific guidance on retry timing.
- Clarify that Responder [Timing measurements](#) are measured under the assumption that the Responder can access the bus.
- 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 `Offset` and `Length` fields in `GET_CERTIFICATE` message.
- Clarified how retried messages affect transcript hash in [Timing requirements](#).
- Clarified that extended algorithms are external to this specification.
- Added definition of [opaque data](#).
- Fixed typo in the `ExchangeData` field of table "Successful `KEY_EXCHANGE_RSP` response message format".

## 660 16.5 Version 1.1.3 (2023-10-08)

---

- Clarified capabilities and algorithms provisioning.
- Added the VESA standards body to [Registry or standards body ID](#).
- Clarified that messages are only hashed once before being signed and verified.

- Added clause that sizes and lengths are in units of bytes.
- Added section " `VendorDefinedReqPayload` and `VendorDefinedRespPayload` defined by DMTF specifications".
- Clarified that SPDM messages sent outside of a session do not contribute to in-session transcripts.
- Clarified that start of the Heartbeat timer can include `PSK_EXCHANGE_RSP` .
- Clarified that measurement block indices are to be in ascending order.
- Clarified in [Registry or standards body ID](#) that the registry specifies the value used for the `VendorID` field.
- Clarified that `ERROR` is only allowed in response to `GET_VERSION` in cases explicitly defined in this specification.
- Clarified the value of the `S1otIDParam` field in `KEY_EXCHANGE_RSP` based on the value of `MutAuthRequested` .
- Added normative information in the [Requester flag fields definitions](#) table and the [Responder flag fields definitions](#) table.
- Clarified sessions can be established one at a time when `HANDSHAKE_IN_THE_CLEAR_CAP` is set.
- Removed text that `ENCRYPT_CAP` and `MAC_CAP` apply to all phases of a secure session.
- 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.
- Stated that all figures are informative unless otherwise specified explicitly.

661

## 16.6 Version 1.1.4 (2024-08-19)

---

- Clarified that `HEARTBEAT` shall be sent if no other messages within the session were sent/received within `HeartbeatPeriod`
- Correct `ReqLength` to `RespLength` in [VENDOR\\_DEFINED\\_RESPONSE response message format](#).
- Clarified that `HeartbeatPeriod` for each secure session is tracked independently

## 662 **17 Bibliography**

---

663 DMTF DSP4014, [DMTF Process for Working Bodies 2.6](#).