



DPC: 25/30508790 DC

BSI Group Headquarters

389 Chiswick High Road London W4 4AL

Tel: +44 (0)20 8996 9000

Fax: +44 (0)20 8996 7400

www.bsigroup.com

Date: 02 April 2025

Origin: International

Latest date for receipt of comments: 26 May 2025

Project No. 2025/00213

Responsible committee: ICT/1 Information systems co-ordination

Interested committees:

Title: Draft BS ISO/IEC 25706 Information technology - Security protocol and data model (SPDM) collection.

Please notify admin.start@bsigroup.com if you are aware of any keywords that might assist in clarifying or identifying the standard or of the content of this standard:

- i) has any issues related to 3rd party IPR, patent or copyright
- ii) affects other national standard(s)
- iii) requires additional national guidance or information

WARNING: THIS IS A DRAFT AND MUST NOT BE REGARDED OR USED AS A BRITISH STANDARD. THIS DRAFT IS NOT CURRENT BEYOND 26 May 2025

This draft is issued to allow comments from interested parties; all comments will be given consideration prior to publication. No acknowledgement will normally be sent. **See overleaf for information on the submission of comments.**

No copying is allowed, in any form, without prior written permission from BSI except as permitted under the Copyright, Designs and Patent Act 1988 or for circulation within a nominating organization for briefing purposes. Electronic circulation is limited to dissemination by e-mail within such an organization by committee members.

Further copies of this draft may be purchased from BSI Shop <http://shop.bsigroup.com> or from BSI Customer Services, Tel: +44(0) 20 8996 9001 or email cservices@bsigroup.com. British, International and foreign standards are also available from BSI Customer Services.

Information on the co-operating organizations represented on the committees referenced above may be obtained from <https://standardsdevelopment.bsigroup.com/>.

E-mail: admin.start@bsigroup.com

Introduction

This draft standard is based on national and international discussions. Your comments on this draft are invited and will assist in the preparation of the consequent standard.

For international standards, comments will be reviewed by the relevant UK national committee before sending the consensus UK vote and comments to the international committee, which will then decide appropriate action. If the international standard is approved, it is usual for the text to be published as a British Standard.

For national standards, comments will be reviewed by the relevant UK national committee and the resulting standards published as a British Standard.

UK Vote

Please indicate whether you consider the UK should submit a negative (with supporting technical reasons) or positive vote on this draft. Please indicate if you are aware of any reason why this draft standard should not be published as a British Standard.

Submission of Comments

- **Annotated drafts are not acceptable and will be rejected.**
- All comments should be submitted online at <https://standardsdevelopment.bsigroup.com/>. You will need to register in order to comment.

Template for comments and secretariat observations

Date: xx/xx/20xx	Document: ISO/DIS xxxx
------------------	------------------------

1	2	(3)	4	5	(6)	7
MB	Clause No./ Subclause No./Annex (e.g. 3.1)	Paragraph/ Figure/Table/Note	Type of comment	Comment (justification for change) by the MB	Proposed change by the MB	Secretariat observations on each comment submitted
	3.1	Definition 1	ed	Definition is ambiguous and needs clarifying.	Amend to read '...so that the mains connector to which no connection...'	
	6.4	Paragraph 2	te	The use of the UV photometer as an alternative cannot be supported as serious problems have been encountered in its use in the UK.	Delete reference to UV photometer.	



DRAFT International Standard

ISO/IEC DIS 25706

Information technology — Security protocol and data model (SPDM) collection

ICS: ISO ics

ISO/IEC JTC 1

Secretariat: **ANSI**

Voting begins on:
2025-04-02

Voting terminates on:
2025-06-25

This document is circulated as received from the committee secretariat.

THIS DOCUMENT IS A DRAFT CIRCULATED FOR COMMENTS AND APPROVAL. IT IS THEREFORE SUBJECT TO CHANGE AND MAY NOT BE REFERRED TO AS AN INTERNATIONAL STANDARD UNTIL PUBLISHED AS SUCH.

IN ADDITION TO THEIR EVALUATION AS BEING ACCEPTABLE FOR INDUSTRIAL, TECHNOLOGICAL, COMMERCIAL AND USER PURPOSES, DRAFT INTERNATIONAL STANDARDS MAY ON OCCASION HAVE TO BE CONSIDERED IN THE LIGHT OF THEIR POTENTIAL TO BECOME STANDARDS TO WHICH REFERENCE MAY BE MADE IN NATIONAL REGULATIONS.

RECIPIENTS OF THIS DRAFT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

Reference number
ISO/IEC DIS 25706:2025(en)

Full document details: 25/30508790 DC

© ISO/IEC 2025  **Documents**
Document Sharing Hub



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2025

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

CONTENTS

Foreword	5
Acknowledgments	5
Introduction	7
<i>Security Protocol and Data Model (SPDM) Specification (DSP0274)</i>	7
<i>SPDM over MCTP Binding Specification (DSP0275)</i>	7
<i>Secured Messages using SPDM over MCTP Binding Specification (DSP0276)</i>	7
<i>Secured Messages using SPDM Specification (DSP0277)</i>	7
Advice	7
1 Scope	8
<i>Security Protocol and Data Model (SPDM) Specification (DSP0274)</i>	8
<i>SPDM over MCTP Binding Specification (DSP0275)</i>	8
<i>Secured Messages using SPDM over MCTP Binding Specification (DSP0276)</i>	8
<i>Secured Messages using SPDM Specification (DSP0277)</i>	8
2 Normative references	9
3 Terms and definitions	11
4 Symbols and abbreviated terms	15
5 Conventions	16
5.1 Document conventions	16
5.2 Reserved and unassigned values	16
5.3 Byte ordering	16
5.4 Sizes and lengths	17
5.5 SPDM data type conventions	18
5.6 Version encoding	18
5.7 Notations	19
5.8 Text or string encoding	20
5.9 Deprecated material	21
5.10 Figures	21
6 Security Protocol and Data Model (SPDM) Specification (DSP0274)	22
6.1 SPDM message exchanges	22
6.2 SPDM messaging protocol	27
6.3 Timing requirements	38
6.4 SPDM messages	45
6.5 Session	199
6.6 Key schedule	201
6.7 Application data	208
6.8 General opaque data format	208
6.9 Signature generation	209
6.10 Signature verification	213
6.11 General ordering rules	215
6.12 DMTF event types	215
7 SPDM over MCTP Binding Specification (DSP0275)	220
8 Secured Messages using SPDM over MCTP Binding Specification (DSP0276)	222
8.1 Secured messages over MCTP	222
8.2 Transport requirements or allowances	223
8.3 Timing requirements	224
9 Secured Messages using SPDM Specification (DSP0277)	225

9.1 Secured Message	225
9.2 Compatibility	231
9.3 Version support	231
9.4 Transport requirements or allowances	233
9.5 Secured Messages opaque data format	234
9.6 SPDM general opaque data format	238
ANNEX A (informative) TLS 1.3	239
ANNEX B (informative) Device certificate example	240
ANNEX C (informative) OID reference	242
ANNEX D (informative) Variable name reference	243
ANNEX E (informative) Sequence number layout	245
Bibliography	246

Foreword

The [Security Protocols and Data Models \(SPDM\)](#) Working Group of [DMTF](#) prepared the *Security Protocol and Data Model (SPDM) Specification* (DSP0274).

The [Platform Management Communications Infrastructure \(PMCI\)](#) Working Group of [DMTF](#) prepared the *Security Protocol and Data Model (SPDM) over MCTP Binding Specification* (DSP0275), the *Secured Messages using SPDM over MCTP Binding Specification* (DSP0276), and the *Secured Messages using SPDM Specification* (DSP0277).

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>.

[Table 1 — Component documents](#) lists the specifications that this collection contains.

Table 1 — Component documents

Document number	Document title	Version
DSP0274	Security Protocol and Data Model (SPDM) Specification	1.3.1
DSP0275	Security Protocol and Data Model (SPDM) over MCTP Binding Specification	1.0.2
DSP0276	Secured Messages using SPDM over MCTP Binding Specification	1.2.0
DSP0277	Secured Messages using SPDM Specification	1.2.0

Acknowledgments

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

- Richelle Ahlvers — Broadcom Inc.
- Jeff Andersen — Google
- Lee Ballard — Dell Technologies
- Steven Bellock — NVIDIA Corporation
- Joshua Cahill — Broadcom Inc.
- Heng Cai — Alibaba Group
- Patrick Caporale — Lenovo
- Yu-Yuan Chen — Intel Corporation
- Andrew Draper — Intel Corporation
- Nigel Edwards — Hewlett Packard Enterprise
- Daniil Egranov — Arm Limited
- Philip Hawkes — Qualcomm Inc.
- Brett Henning — Broadcom Inc.
- Jeff Hilland — Hewlett Packard Enterprise
- Yi Hou — Microchip
- Guernsey Hunt — IBM

- Yuval Itkin — NVIDIA Corporation
- Theo Koulouris — Hewlett Packard Enterprise
- Raghupathy Krishnamurthy — NVIDIA Corporation
- Benjamin Lei — Lenovo
- Eliel Louzoun — Intel Corporation
- Luis Luciani — Hewlett Packard Enterprise
- Masoud Manoo — Lenovo
- Donald Matthews — Advanced Micro Devices, Inc.
- Mahesh Natu — Intel Corporation
- Chandra Nelogal — Dell Technologies
- Edward Newman — Hewlett Packard Enterprise
- Alexander Novitskiy — Intel Corporation
- Jim Panian — Qualcomm Inc.
- Scott Phuong — Cisco Systems Inc., Axiado Corporation
- Jeffrey Plank — Microchip
- Viswanath Ponnuru — Dell Technologies
- Lohith Rangappa — Marvell Technology, Inc.
- Xiaoyu Ruan — Intel Corporation
- Nitin Sarangdhar — Intel Corporation
- Vidya Satyamsetti — Google
- Hemal Shah — Broadcom Inc.
- Yoni Shternhell — Western Digital Technologies, Inc.
- Bob Stevens — Dell Technologies
- Srikanth Varadarajan — Intel Corporation
- Peng Xiao — Alibaba Group
- Qing Yang — Alibaba Group
- Jiewen Yao — Intel Corporation

Introduction

Security Protocol and Data Model (SPDM) Specification (DSP0274)

DSP0274 defines [messages](#), data objects, and sequences for performing message exchanges over a variety of transport and physical media. The description of message exchanges includes [authentication](#) and provisioning of hardware identities, measurement for firmware identities, session key exchange protocols to enable confidentiality with integrity-protected data communication, and other related capabilities. SPDM enables efficient access to low-level security capabilities and operations. In addition, other mechanisms, including non-DMTF-defined mechanisms, can use the SPDM.

SPDM over MCTP Binding Specification (DSP0275)

DSP0275 defines how SPDM is transported over MCTP communications. SPDM is supported as a message type over MCTP, and the SPDM over MCTP binding defines the format of SPDM messages transported over MCTP.

Secured Messages using SPDM over MCTP Binding Specification (DSP0276)

DSP0276 binds Secured Messages using the SPDM specification (DSP0277) to MCTP transport.

Secured Messages using SPDM Specification (DSP0277)

DSP0277 defines the methodology that various PMCI transports can use to communicate various application data securely by utilizing SPDM. Specifically, DSP0277 defines the transport requirements for SPDM records, which form the basis of encryption and message authentication. Furthermore, DSP0277 contains guidance and certain decisions that it defers to the binding specification, which binds Secured Messages to a specific transport. Thus, the binding specification is expected to finalize those decisions or guidance by way of normalization or recommendation. DSP0277 was written with PMCI transports in mind, but nothing precludes specifying bindings to other transports.

Advice

The authors of these specifications recommend that readers visit tutorial and educational materials under [Security Protocols and Data Models \(SPDM\)](#) and [Platform Management Communications Infrastructure \(PMCI\)](#) on the DMTF website prior to or during the reading of these specifications to aid in understanding them fully.

1 Scope

Security Protocol and Data Model (SPDM) Specification (DSP0274)

DSP0274 describes how to use messages, data objects, and sequences to exchange messages between two devices over a variety of transports and physical media. DSP0274 contains the message exchanges, sequence diagrams, message formats, and other relevant semantics for such message exchanges, including authentication of hardware identities and firmware measurements.

Other specifications define the mapping of these messages to different transports and physical media. DSP0274 provides information to enable security policy enforcement but does not specify individual policy decisions.

SPDM over MCTP Binding Specification (DSP0275)

DSP0275 defines the format of Security Protocol and Data Model (SPDM) over MCTP messages. DSP0275 describes both SPDM over MCTP binding and the common format for SPDM over MCTP messages.

Secured Messages using SPDM over MCTP Binding Specification (DSP0276)

DSP0276 binds Secured Messages using SPDM to MCTP transport and further defines the transport specific details as outlined in *Secured Messages using SPDM Specification (DSP0277)*. DSP0276 1.2.0 (this version) binds to version 1.2.0 and all 1.2 errata versions of DSP0277.

Secured Messages using SPDM Specification (DSP0277)

DSP0277 defines a generic record format used to encrypt and authenticate any application data within SPDM's secure session. Also, relating to encryption, message authentication, and secure sessions, DSP0277 further defines those areas in SPDM that the specification states are the responsibilities of the transport layer. DSP0277 requires SPDM version 1.1 or later.

2 Normative references

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

- DMTF DSP0004, *Common Information Model (CIM) Metamodel*, https://www.dmtf.org/sites/default/files/standards/documents/DSP0004_3.0.pdf
- DMTF DSP0223, *Generic Operations*, https://www.dmtf.org/sites/default/files/standards/documents/DSP0223_1.0.pdf
- DMTF DSP0236, *MCTP Base Specification 1.3*, https://www.dmtf.org/sites/default/files/standards/documents/DSP0236_1.3.pdf
- DMTF DSP0239, *MCTP IDs and Codes 1.7*, https://www.dmtf.org/sites/default/files/standards/documents/DSP0239_1.7.pdf
- DMTF DSP0240, *Platform Level Data Model (PLDM) Base Specification*, https://www.dmtf.org/sites/default/files/standards/documents/DSP0240_1.0.pdf
- DMTF DSP0274, *Security Protocol and Data Model (SPDM) Specification*, <https://www.dmtf.org/dsp/DSP0274>
- DMTF DSP0275, *Security Protocol and Data Model (SPDM) over MCTP Binding Specification*, <https://www.dmtf.org/dsp/DSP0275>
- DMTF DSP0276, *Secured Messages using SPDM over MCTP Binding Specification*, <https://www.dmtf.org/dsp/DSP0276>
- DMTF DSP0277, *Secured Messages using SPDM Specification*, <https://www.dmtf.org/dsp/DSP0277>
- DMTF DSP1001, *Management Profile Usage Guide*, https://www.dmtf.org/sites/default/files/standards/documents/DSP1001_1.2.pdf
- GB/T 32905-2016, *Information security technology—SM3 cryptographic hash algorithm*, August 2016
- GB/T 32907-2016, *Information security technology—SM4 block cipher algorithm*, August 2016
- GB/T 32918.1-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 1: General*, August 2016
- GB/T 32918.2-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 2: Digital signature algorithm*, August 2016
- GB/T 32918.3-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 3: Key exchange protocol*, August 2016
- GB/T 32918.4-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 4: Public key encryption algorithm*, August 2016
- GB/T 32918.5-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 5: Parameter definition*, August 2016
- IETF RFC 2986, *PKCS #10: Certification Request Syntax Specification*, November 2000, <https://tools.ietf.org/html/rfc2986>
- IETF RFC 4716, *The Secure Shell (SSH) Public Key File Format*, November 2006, <https://tools.ietf.org/html/rfc4716>
- IETF RFC 5116, *An Interface and Algorithms for Authenticated Encryption*, January 2008, <https://tools.ietf.org/html/rfc5116>

- IETF RFC 5234, *Augmented BNF for Syntax Specifications: ABNF*, January 2008, <https://tools.ietf.org/html/rfc5234>
- IETF RFC 5280, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, May 2008, <https://tools.ietf.org/html/rfc5280>
- IETF RFC 7250, *Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*, June 2014, <https://tools.ietf.org/html/rfc7250>
- IETF RFC 7919, *Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)*, August 2016, <https://tools.ietf.org/html/rfc7919>
- IETF RFC 8017, *PKCS #1: RSA Cryptography Specifications Version 2.2*, November 2016, <https://tools.ietf.org/html/rfc8017>
- IETF RFC 8032, *Edwards-Curve Digital Signature Algorithm (EdDSA)*, January 2017, <https://tools.ietf.org/html/rfc8032>
- IETF RFC 8439, *ChaCha20 and Poly1305 for IETF Protocols*, June 2018, <https://tools.ietf.org/html/rfc8439>
- IETF RFC 8446, *The Transport Layer Security (TLS) Protocol Version 1.3*, August 2018, <https://tools.ietf.org/html/rfc8446>
- IETF RFC 8998, *ShangMi (SM) Cipher Suites for TLS 1.3*, March 2021, <https://tools.ietf.org/html/rfc8998>
- IETF RFC 9147, *The Datagram Transport Layer Security (DTLS) Protocol Version 1.3*, April 2022, <https://datatracker.ietf.org/doc/html/rfc9147>
- *ISO/IEC Directives, Part 2, Principles and rules for the structure and drafting of ISO and IEC documents*, <https://www.iso.org/sites/directives/current/part2/index.xhtml>
- *ASCII — ISO/IEC 646:1991, 09/1991*, <https://www.iso.org/standard/4777.html>
- NIST Special Publication 800-38D, *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, November 2007, <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf>
- *TCG Algorithm Registry, Family "2.0", Level 00 Revision 01.32*, June 25, 2020, <https://trustedcomputinggroup.org/resource/tcg-algorithm-registry/>
- *USB Authentication Specification Rev 1.0 with ECN and Errata through January 7, 2019*, <https://www.usb.org/document-library/usb-authentication-specification-rev-10-ecn-and-errata-through-january-7-2019>
- **ASN.1 — ISO-822-1-4, DER — ISO-8825-1**
 - ITU-T X.680, X.681, X.682, X.683, X.690, 08/2015, <https://www.itu.int/rec/T-REC-X.680-X.693-201508-S/en>
- **ECDSA**
 - Section 6, *The Elliptic Curve Digital Signature Algorithm (ECDSA) in FIPS PUB 186-5 Digital Signature Standard (DSS)* <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf>
 - IETF RFC 6979, *Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)*, August 2013, <https://tools.ietf.org/html/rfc6979>
 - NIST SP 800-186 *Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters* <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186.pdf>
- **SHA2-256, SHA2-384, and SHA2-512**
 - FIPS PUB 180-4 *Secure Hash Standard (SHS)* <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>
- **SHA3-256, SHA3-384, and SHA3-512**
 - FIPS PUB 202 *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions* <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>

3 Terms and definitions

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

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

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

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

The terms that [DSP0004](#), [DSP0223](#), [DSP0236](#), [DSP0239](#), and [DSP1001](#) define also apply to the specifications in this collection.

The specifications in this collection use the following terms:

Term	Definition
alias certificate	Certificate that is dynamically generated by the component or component firmware.
application data	Data that is specific to the application and whose definition and format is outside the scope of these specifications. Application data usually exists at the application layer, which is, in general, the layer above SPDM and the transport layer. Examples of data that could be application data include: messages carried as DMTF MCTP payloads; Internet traffic; PCIe transaction layer packets (TLPs); camera images and video (MIPI CSI-2 packets); video display stream (MIPI DSI-2 packets); and touchscreen data (MIPI I3C Touch).
authentication initiator	Endpoint that initiates the authentication process by challenging another endpoint.
authentication	Process of determining whether an entity is who or what it claims to be.
byte	Eight-bit quantity. Also known as an <i>octet</i> .
certificate authority (CA)	Trusted entity that issues certificates.
certificate chain	Typically a series of two or more certificates. Each certificate is signed by the preceding certificate in the chain.
certificate	Digital form of identification that provides information about an entity and certifies ownership of a particular asymmetric key-pair.
component	Physical device, contained in a single package. A "component" may also refer to a functional block implemented in hardware, firmware, and/or software.
device certificate	Certificate that contains information that identifies the component. Can be a leaf certificate or an intermediate certificate .
device	Physical entity such as a network controller or a fan.

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.
encapsulated request	A request embedded into an <code>ENCAPSULATED_REQUEST</code> or <code>ENCAPSULATED_RESPONSE_ACK</code> response message to allow the Responder to issue a request to a Requester. See GET_ENCAPSULATED_REQUEST request and ENCAPSULATED_REQUEST response messages .
generic certificate	A certificate, for use in certificate slots 1 or greater, that has minimal SPDM requirements to allow for numerous use cases that the vendor, standards body, or user defines.
endpoint	Logical entity that communicates with other endpoints over one or more transport protocols.
event notifier	An SPDM endpoint that is capable of sending asynchronous notifications using SPDM event mechanisms. See Event mechanism .
event recipient	An SPDM endpoint that is capable of receiving asynchronous notifications using SPDM event mechanisms. See Event mechanism .
hardware identity	A value that represents a unique instance of an endpoint. See Identity authentication .
intermediate certificate	Certificate that is neither a root certificate nor a leaf certificate.
invasive debug mode	A device mode that enables debug access that might expose or allow modification of firmware, hardware, or settings that can access (read or write) security keys, states, and contexts of the device. A device should not be trusted when it is operating in this mode.
large SPDM message	An SPDM message that is greater than the <code>DataTransferSize</code> of the receiving SPDM endpoint or greater than the transmit buffer size of the sending SPDM endpoint.
large SPDM request message	A large SPDM message that is an SPDM request.
large SPDM response message	A large SPDM message that is an SPDM response.
leaf certificate	Last certificate in a certificate chain. A leaf certificate is synonymous with an end entity certificate as RFC 5280 describes.
measurement	Representation of hardware/firmware/software or configuration data on an endpoint.
message	See SPDM message .
message body	Portion of an SPDM message that carries additional data.
message transcript	The concatenation of a sequence of messages in the order in which they are sent and received by an endpoint. The final message included in the message transcript may be truncated to allow inclusion of a signature in that message which is computed over the message transcript. If an endpoint is communicating with multiple peer endpoints concurrently, the message transcripts for the peers are accumulated separately and independently.
monotonically increasing	These specifications uses the term <i>monotonically increasing</i> to describe an integer field where the value of each instance of the field in a series increases from a lower starting point to a higher ending point without repeating values. For instance, a <i>monotonically increasing</i> field may contain the values 1, 3, 4, 7, and 9.
most significant byte (MSB)	Highest-order byte in a number consisting of multiple bytes.

Term	Definition
Negotiated State	Set of parameters that represents the state of the communication between a corresponding pair of Requester and Responder at the successful completion of the <code>NEGOTIATE_ALGORITHMS</code> messages. These parameters may include values provided in <code>VERSION</code>, <code>CAPABILITIES</code>, and <code>ALGORITHMS</code> messages. Additionally, they may include parameters associated with the transport layer. They may include other values deemed necessary by the Requester or Responder to continue or preserve communication with each other.
nibble	Computer term for a four-bit aggregation, or half of a byte.
non-invasive debug mode	A device mode that enables debug access that does not expose or allow modification of security-critical firmware, hardware, or settings.
nonce	Number that is unpredictable to entities other than its generator. The probability of the same number occurring more than once is negligible. A nonce may be generated by combining a random number of at least 64 bits, optionally concatenated with a monotonically increasing counter of size suitable for the application.
opaque data	Opaque data fields transfer data that is outside the scope of these specifications. The semantics and usage of this data are implementation specific and are also outside the scope of these specifications.
payload	Information-bearing fields of a message. These fields are separate from the transport fields and elements, such as address fields, framing bits, and checksums, that transport the message from one point to another.
physical transport binding	Specifications that define how a base messaging protocol is implemented on a particular physical transport type and medium, such as SMBus/I ² C or PCI Express™ Vendor Defined Messaging.
Platform Management Communications Infrastructure (PMCI)	Working group of DMTF that defines standardized communication protocols, low-level data models, and transport definitions that support communications with and between management controllers and management devices that form a platform management subsystem within a managed computer system.
record	A unit or chunk of data that is either encrypted and/or authenticated.
Requester	Original transmitter, or source, of an SPDM request message. It is also the ultimate receiver, or destination, of an SPDM response message. A Requester is the sender of the <code>GET_VERSION</code> request and remains the requester for the remainder of that connection.
Reset	This term is used to denote a Reset or restart of a device that runs the Requester or Responder code, which typically leads to the loss of all volatile state on the device.
Responder	Ultimate receiver, or destination, of an SPDM request message. It is also the original transmitter, or source of an SPDM response message.
root certificate	First certificate in a certificate chain, which acts as the trust anchor and is typically self-signed.
secure session	Provides either encryption or message authentication or both for communicating data over a transport; however, the following exception applies: DSP0277 defines a secure session as consisting of both encryption and message authentication or message authentication without encryption.

Term	Definition
Security Protocols and Data Models (SPDM)	Working group of DMTF that defines standards to enable security for platforms, whether for the control plane, data plane, or other infrastructure.
sequentially decreasing	These specifications use the term <i>sequentially decreasing</i> to describe an integer field where the value of each instance of the field in a series decrements from a higher starting point to a lower ending point without skipping or repeating values. For instance, a <i>sequentially decreasing</i> field may contain the values 255, 254, 253, 252, and 251.
sequentially increasing	These specifications use the term <i>sequentially increasing</i> to describe an integer field where the value of each instance of the field in a series increments from a lower starting point to a higher ending point without skipping or repeating values. For instance, a <i>sequentially increasing</i> field may contain the values 1, 2, 3, 4, and 5.
session keys	Any secrets, derived cryptographic keys, or any cryptographic information bound to a session.
Session-Secrets-Exchange	Any SPDM request and their corresponding response that initiates a session and provides initial cryptographic exchange. Examples of such requests are <code>KEY_EXCHANGE</code> and <code>PSK_EXCHANGE</code> .
Session-Secrets-Finish	This term denotes any SPDM request and its corresponding response that finalizes a session setup and provides the final exchange of cryptographic or other information before application data can be securely transmitted. Examples of such requests are <code>FINISH</code> and <code>PSK_FINISH</code> .
SPDM message payload	Portion of the message body of an SPDM message. This portion of the message is separate from those fields and elements that identify the SPDM version, the SPDM request and response codes, and the two parameters.
SPDM message	Unit of communication in SPDM communications. See Generic SPDM message format .
SPDM request message	Message that is sent to an endpoint to request a specific SPDM operation. A corresponding SPDM response message acknowledges receipt of an SPDM request message.
SPDM response message	Message that is sent in response to a specific SPDM request message. This message includes a <code>Response Code</code> field that indicates whether the request completed normally.
trusted computing base (TCB)	Set of all hardware, firmware, and/or software components that are critical to its security, in the sense that bugs or vulnerabilities occurring inside the TCB might jeopardize the security properties of the entire system. By contrast, parts of a computer system outside the TCB shall not be able to misbehave in a way that would leak any more privileges than are granted to them in accordance with the security policy. Reference: https://en.wikipedia.org/wiki/Trusted_computing_base
trusted environment	An environment where the operator is assured of no unauthorized interference in operations.

4 Symbols and abbreviated terms

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

The following additional abbreviations are used in this document.

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

5 Conventions

The following conventions apply to all SPDM specifications.

5.1 Document conventions

- Document titles appear in *italics*.
- The first occurrence of each important term appears in *italics* with a link to its definition.
- [ABNF rules](#) appear in a `monospaced` font.

5.2 Reserved and unassigned values

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

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

5.3 Byte ordering

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

5.3.1 Hash byte order

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

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

5.3.2 Encoded ASN.1 byte order

For fields or values containing DER, CER, or BER encoded data, SPDM preserves the byte order as described in [X.690](#) specification. SPDM views a DER, CER, or BER encoded data as simply a string of octets where the

first byte is the leftmost byte of Figure 1 or Figure 2 in the X.690 specification, the second byte is the second leftmost byte, the third byte is the third leftmost byte, and this pattern continues until the last byte. The first byte is also called either the Identifier octet or the Leading identifier octet. The X.690 specification defines Figure 1, Figure 2, and identifier octets. When populating a DER, CER, or BER encoded data in SPDM fields, the first byte is placed in the lowest address, the second byte is placed in the second lowest offset, the third byte is placed in the third lowest offset in the field or value and this pattern continues until the last byte.

5.3.3 Octet string byte order

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

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

5.3.4 Signature byte order

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

5.3.4.1 ECDSA signatures byte order

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

5.3.4.2 SM2 signatures byte order

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

5.4 Sizes and lengths

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

5.5 SPDM data type conventions

5.5.1 SPDM data types

Table 2 — SPDM data types lists the abbreviations and descriptions for common data types that SPDM message fields and data structure definitions use. These definitions follow DSP0240.

Table 2 — SPDM data types

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

5.5.2 Integers

Unless noted otherwise, integers shall be unsigned.

5.6 Version encoding

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

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

EXAMPLE:

Version 3.7 → 0x37

Version 1.0 → 0x10

Version 1.2 → 0x12

An endpoint that supports Version 1.2 can interoperate with an older endpoint that supports Version 1.0 or other previous minor versions. Whether an endpoint supports inter-operation with previous minor versions of the SPDM specification is an implementation-specific decision.

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

These specifications consider two minor versions to be interoperable when it is possible for an implementation that is conformant to a higher minor version number to also communicate with an implementation that is

conformant to a lower minor version number with minimal differences in operation. In such a case, the following rules apply:

- Both endpoints shall use the same lower version number in the `SPDMVersion` field for all messages.
- Functionality shall be limited to what the lower minor version of the SPDM specification defines.
- Computations and other operations between different minor versions of the Secured Messages using SPDM specification should remain the same, unless security issues of lower minor versions are fixed in higher minor versions and the fixes require change in computations or other operations. These differences are dependent on the value in the `SPDMVersion` field in the message.
- In a newer minor version of the SPDM specification, a given message can be longer, bit fields and enumerations can contain new values, and reserved fields can gain functionality. Existing numeric and bit fields retain their existing definitions.
- Errata versions (indicated by a non-zero value in the `UpdateVersionNumber` field for the `GET_VERSION` request and `VERSION` response messages) clarify existing behaviors in the SPDM specification. They maintain bitwise compatibility with the base version, except as required to fix security vulnerabilities or to correct mistakes from the base version.

For details on the version agreement process, see [GET_VERSION request and VERSION response messages](#). The detailed version encoding that the `VERSION` response message returns contains an additional byte that indicates specification bug fixes or development versions. See [Table 10 — Successful VERSION response message format](#).

5.7 Notations

SPDM specifications use the following notations:

Notation	Description
<code>Concatenate()</code>	The concatenation function <code>Concatenate(a, b, ..., z)</code> , where the first entry occupies the least-significant bits and the last entry occupies the most-significant bits.
<code>M:N</code>	In field descriptions, this notation typically represents a range of byte offsets starting from byte <code>M</code> and continuing to and including byte <code>N</code> ($M \leq N$). The lowest offset is on the left. The highest offset is on the right.
<code>[4]</code>	Square brackets around a number typically indicate a bit offset. Bit offsets are zero-based values. That is, the least significant bit (<code>[LSB]</code>) offset = 0.
<code>[M:N]</code>	A range of bit offsets where M is greater than or equal to N. The most significant bit is on the left, and the least significant bit is on the right.
<code>1b</code>	A lowercase <code>b</code> after a number consisting of <code>0</code> s and <code>1</code> s indicates that the number is in binary format.
<code>0x12A</code>	Hexadecimal, as indicated by the leading <code>0x</code> .

Notation	Description
<code>N+</code>	Variable-length byte range that starts at byte offset N.
<code>{ Payload }</code>	Used mostly in figures, this notation indicates that the payload specified in the enclosing curly brackets is encrypted and/or authenticated by the keys derived from one or more major secrets. The specific secret used is described throughout this specification. For example, <code>{ HEARTBEAT }</code> shows that the Heartbeat message is encrypted and/or authenticated by the keys derived from one or more major secrets.
<code>{ Payload }::[[S_x]]</code>	Used mostly in figures, this notation indicates that the payload specified in the enclosing curly brackets is encrypted and/or authenticated by the keys derived from major Secret X. For example, <code>{ HEARTBEAT }::[[S₂]]</code> shows that the Heartbeat message is encrypted and/or authenticated by the keys derived from major secret <code>S₂</code> .
<code>[\${message_name}] . \${field_name}</code>	Used to indicate a field in a message. <code>\${message_name}</code> is the name of the request or response message. <code>\${field_name}</code> is the name of the field in the request or response message. An asterisk (<code>*</code>) instead of a field name means all fields in that message except for any conditional fields that are empty (as for example <code>KEY_EXCHANGE . OpaqueData</code>).

5.8 Text or string encoding

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

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

[Table 3 — "spdm" encoding example](#) shows an encoding example of the string "spdm":

Table 3 — "spdm" encoding example

Offset	Character	Value
0	s	<code>0x73</code>
1	p	<code>0x70</code>
2	d	<code>0x64</code>
3	m	<code>0x6D</code>

5.9 Deprecated material

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

The following typographical convention indicates deprecated material:

DEPRECATED

Deprecated material appears here.

DEPRECATED

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

5.10 Figures

Unless otherwise specified, all figures are informative.

6 Security Protocol and Data Model (SPDM) Specification (DSP0274)

6.1 SPDM message exchanges

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

The specification-defined message exchanges enable Requesters to:

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

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.

6.1.1 Security capability discovery and negotiation

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

6.1.2 Identity authentication

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

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

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

6.1.2.1 Identity provisioning

Identity provisioning is the process that device vendors follow during or after hardware manufacturing to equip a

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

6.1.2.1.1 Certificate models

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

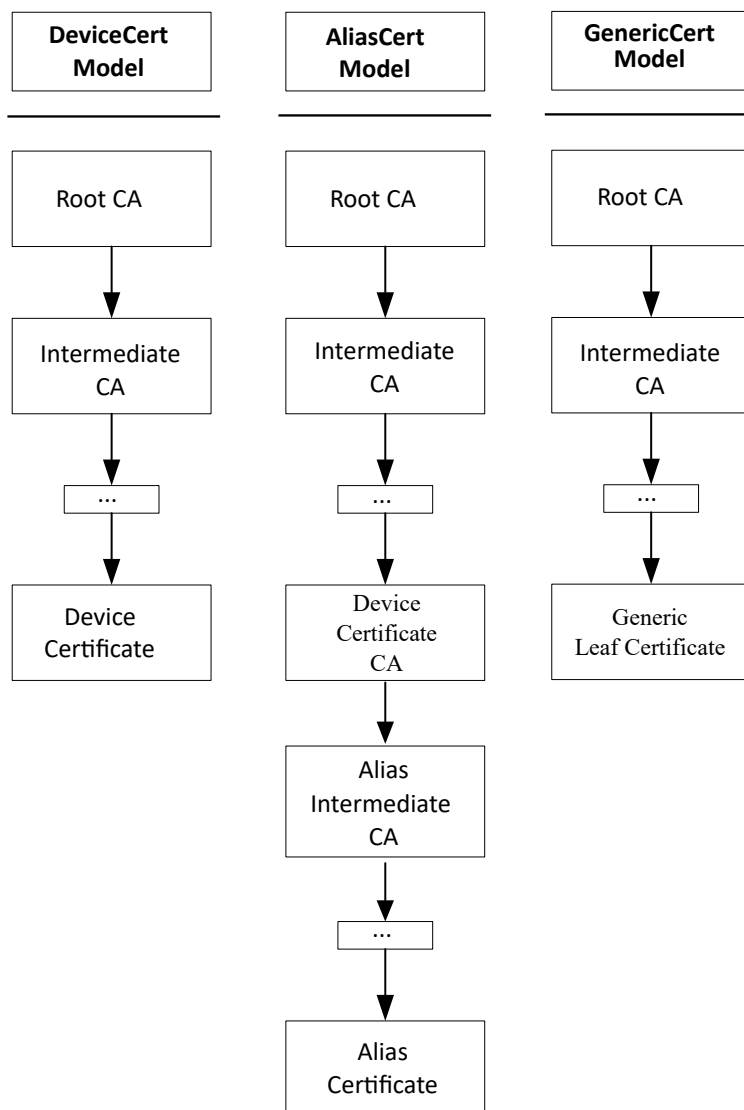
If the certificate chain uses the device certificate or alias certificate model, the certificate chain should contain at least one certificate that includes hardware identity information. One means of conveying hardware identity is by use of a public key. The [Hardware identity OID](#) should be used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

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

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

Figure 1 — SPDM certificate chain models shows the SPDM certificate chain models:

Figure 1 — SPDM certificate chain models



6.1.2.1.1.1 Device certificate model

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

6.1.2.1.1.2 Alias certificate model

When the alias certificate (`AliasCert`) model is in use, the leaf certificate is an Alias Certificate, in which case there may be one or more intermediate `AliasCert` certificates between the Device Certificate and the leaf Alias Certificate. In the `AliasCert` model, the device private key signs the next level Alias Certificate, and then the private key associated with the public key in each Alias Intermediate CA signs the Alias Certificate below it. When the `AliasCert` model is in use, the Device Certificate is referred to as a Device Certificate CA, indicating

that the certificate both contains device hardware identity information and functions as a certificate authority to sign an additional certificate. In this model, the Device Certificate CA should contain the [Hardware identity OID](#).

A device that implements the `AliasCert` model might factor some mutable information, such as the measurement of a firmware image, into the derivation of the public/private key pairs for the intermediate and leaf alias certificates. Therefore, the asymmetric public/private key pairs for each Alias Certificate should be treated as mutable.

Through the certificate chain, the root CA indirectly endorses the device public key in the Device Certificate. When the `AliasCert` model is in use, the Alias Certificates are endorsed by the device private key, meaning that the Alias Certificates are also indirectly endorsed by the root CA.

When the `AliasCert` model is used, the device creates and endorses one or more certificates. The certificates from the root certificate to the Device Certificate are considered immutable because the Responder cannot change them, as they can only be changed through the `SET_CERTIFICATE` command or an equivalent capability. The certificates below the Device Certificate can be created on the device and are mutable certificates in that they can change when the device state changes, such as a device [reset](#). The [Mutable certificate OID](#) should be used to indicate mutable certificates.

In addition, when the `AliasCert` model is used, one or more Alias Certificates can contain firmware identity information. Other standards bodies might define the format of the firmware identity information. Such definitions are outside the scope of this specification.

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

6.1.2.1.1.3 Generic certificate model

With the support of [multiple asymmetric keys](#), the need for another certificate model arises to accommodate varying use cases that `DeviceCert` and `AliasCert` models cannot fulfill. Thus, the generic certificate model offers the greatest flexibility to the device manufacturer, a manufacturer in the supply chain, and the users of the SPDM endpoint.

As [Figure 1 — SPDM certificate chain models](#) illustrates, much like the other certificate models, the generic certificate model, too, is composed of a chain of certificates starting with the root and ending with the leaf. The root CA, too, either directly certifies the leaf certificate or indirectly certifies the leaf certificate (`GenericCert`) through one or more intermediate certificate authorities. In other words, this model is the most flexible (or least restrictive) of the certificate models in this specification. The main difference between this model and the other models is that SPDM shall not impose any requirements on the contents of each certificate in the chain in a generic certificate model other than the key pair and related information associated in the leaf certificate.

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

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

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

6.1.2.2 Raw public keys

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

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

6.1.2.3 Runtime authentication

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

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

6.1.3 Firmware and configuration measurement

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

6.1.4 Secure sessions

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

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

Finally, many SPDM requests and their corresponding responses can also be afforded the same protection. For more details, see [Table 7 — SPDM request and response messages validity](#) and the [SPDM request and response code issuance allowance](#) clause.

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

6.1.5 Mutual authentication overview

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

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

6.1.6 Multiple asymmetric key support

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

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

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

6.1.7 Custom environments

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

6.1.8 Notification overview

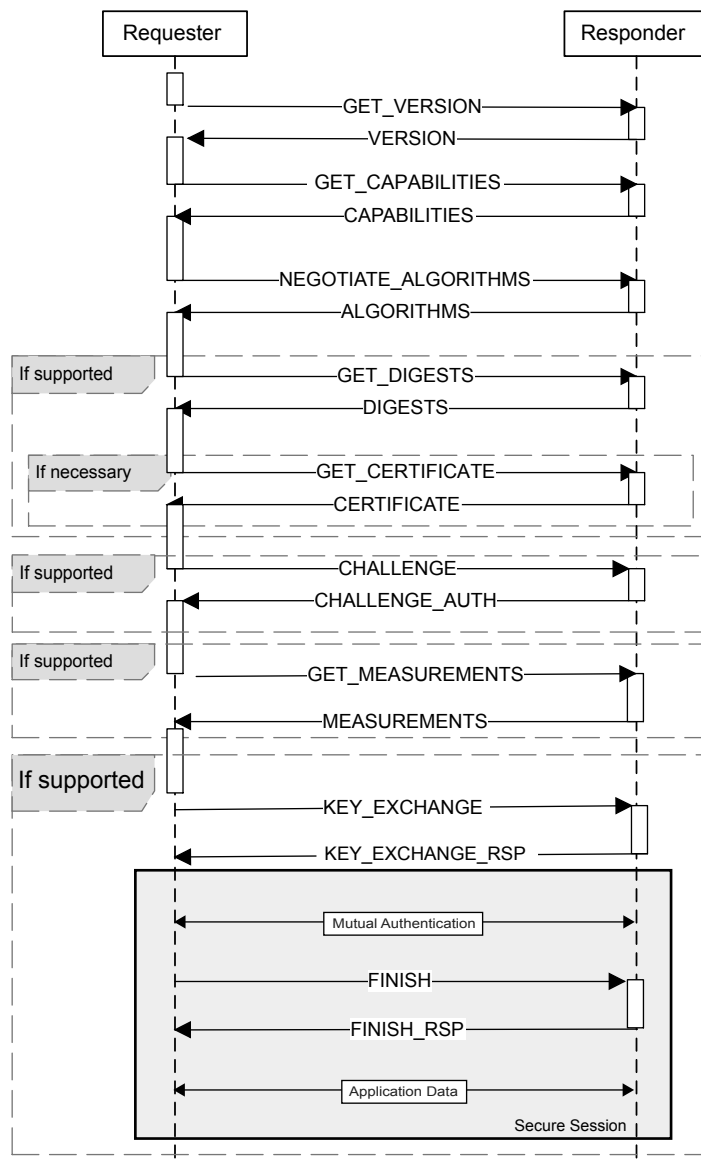
To aid an SPDM endpoint in enforcing its security policy requirements in an efficient, reliable, and timely manner, the [SPDM event mechanism](#) provides a method to asynchronously deliver a notification to or receive a notification from the interested SPDM endpoint. This mechanism allows an interested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

6.2 SPDM messaging protocol

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

Figure 2 — 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.

Figure 2 — SPDM messaging protocol flow



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

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

6.2.1 SPDM connection model

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

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

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

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

6.2.2 SPDM bits-to-bytes mapping

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

Figure 3 — One-byte field bit map shows the one-byte field bit map:

Figure 3 — One-byte field bit map

Example:

A One-Byte Field Starting at Byte Offset 3

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

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

Figure 4 — Two-byte field bit map

Example:

A Two-Byte Field Starting at Byte Offset 5

Byte Offset 6								Byte Offset 5							
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

6.2.3 Generic SPDM message format

Table 4 — Generic SPDM message field definitions defines the fields that constitute a generic SPDM message, including the message header and payload:

Table 4 — Generic SPDM message field definitions

Byte offset	Bit offset	Size (bits)	Field	Description
0	[7:4]	4	SPDM Major Version	Shall be the major version of the SPDM Specification. An endpoint shall not communicate by using an incompatible SPDM version value. See Version encoding .
0	[3:0]	4	SPDM Minor Version	Shall be the minor version of the SPDM Specification. A specification with a given minor version extends a specification with a lower minor version as long as they share the major version. See Version encoding .
1	[7:0]	8	Request Response Code	Shall be the request message code or response code, which Table 5 — SPDM request codes and Table 6 — SPDM response codes enumerate. 0x00 through 0x7F represent response codes and 0x80 through 0xFF represent request codes. In request messages, this field is considered the request code. In response messages, this field is considered the response code.
2	[7:0]	8	Param1	Shall be the first one-byte parameter. The contents of the parameter are specific to the Request Response Code.
3	[7:0]	8	Param2	Shall be the second one-byte parameter. The contents of the parameter are specific to the Request Response Code.

Byte offset	Bit offset	Size (bits)	Field	Description
4	See the description.	Variable	<i>SPDM message payload</i>	Shall be zero or more bytes that are specific to the Request Response Code .

6.2.3.1 SPDM version

The `SPDMVersion` field, present as the first field in all SPDM messages, indicates the version of the SPDM specification that the format of an SPDM message adheres to. The format of this field shall be the same as byte 0 in [Table 4 — Generic SPDM message field definitions](#). The value of this field shall be the same value as the version of this specification except for `GET_VERSION` and `VERSION` messages.

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

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

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

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

6.2.4 SPDM request codes

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

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

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

Table 5 — SPDM request codes

Request	Code value	Implementation requirement	Message format
<code>GET_DIGESTS</code>	0x81	Optional	Table 35 — GET_DIGESTS request message format
<code>GET_CERTIFICATE</code>	0x82	Optional	Table 39 — GET_CERTIFICATE request message format
<code>CHALLENGE</code>	0x83	Optional	Table 45 — CHALLENGE request message format
<code>GET_VERSION</code>	0x84	Required	Table 9 — GET_VERSION request message format
<code>CHUNK_SEND</code>	0x85	Optional	Table 98 — CHUNK_SEND request format
<code>CHUNK_GET</code>	0x86	Optional	Table 102 — CHUNK_GET request format
<code>GET_ENDPOINT_INFO</code>	0x87	Optional	Table 121 — GET_ENDPOINT_INFO request format

Request	Code value	Implementation requirement	Message format
GET_MEASUREMENTS	0xE0	Optional	Table 50 — GET_MEASUREMENTS request message format
GET_CAPABILITIES	0xE1	Required	Table 12 — GET_CAPABILITIES request message format
GET_SUPPORTED_EVENT_TYPES	0xE2	Optional	Table 111 — GET_SUPPORTED_EVENT_TYPES request message format
NEGOTIATE_ALGORITHMS	0xE3	Required	Table 16 — NEGOTIATE_ALGORITHMS request message format
KEY_EXCHANGE	0xE4	Optional	Table 71 — KEY_EXCHANGE request message format
FINISH	0xE5	Optional	Table 74 — FINISH request message format
PSK_EXCHANGE	0xE6	Optional	Table 76 — PSK_EXCHANGE request message format
PSK_FINISH	0xE7	Optional	Table 78 — PSK_FINISH request message format
HEARTBEAT	0xE8	Optional	Table 80 — HEARTBEAT request message format
KEY_UPDATE	0xE9	Optional	Table 82 — KEY_UPDATE request message format
GET_ENCAPSULATED_REQUEST	0xEA	Optional	Table 85 — GET_ENCAPSULATED_REQUEST request message format
DELIVER_ENCAPSULATED_RESPONSE	0xEB	Optional	Table 87 — DELIVER_ENCAPSULATED_RESPONSE request message format
END_SESSION	0xEC	Optional	Table 89 — END_SESSION request message format
GET_CSR	0xED	Optional	Table 92 — GET_CSR request message format
SET_CERTIFICATE	0xEE	Optional	Table 95 — SET_CERTIFICATE request message format
GET_MEASUREMENT_EXTENSION_LOG	0xEF	Optional	Table 128 — GET_MEASUREMENT_EXTENSION_LOG message format
SUBSCRIBE_EVENT_TYPES	0xF0	Optional	Table 115 — SUBSCRIBE_EVENT_TYPES request message format
SEND_EVENT	0xF1	Optional	Table 118 — SEND_EVENT request message format

Request	Code value	Implementation requirement	Message format
GET_KEY_PAIR_INFO	0xFC	Optional	Table 104 — GET_KEY_PAIR_INFO request message format
SET_KEY_PAIR_INFO	0xFD	Optional	Table 108 — SET_KEY_PAIR_INFO request message format
VENDOR_DEFINED_REQUEST	0xFE	Optional	Table 68 — VENDOR_DEFINED_REQUEST request message format
RESPOND_IF_READY	0xFF	Required	Table 67 — RESPOND_IF_READY request message format
Reserved	All other values		SPDM implementations compatible with this version shall not use the reserved request codes.

6.2.5 SPDM response codes

The Request Response Code field in the SPDM response message shall specify the appropriate response code for a request. All SPDM-compatible implementations shall use [Table 6 — SPDM response codes](#).

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

[Table 6 — SPDM response codes](#) defines the response codes for SPDM. The **Implementation requirement** column indicates requirements on the Responder.

Table 6 — SPDM response codes

Response	Value	Implementation requirement	Message format
DIGESTS	0x01	Optional	Table 36 — Successful DIGESTS response message format
CERTIFICATE	0x02	Optional	Table 41 — Successful CERTIFICATE response message format
CHALLENGE_AUTH	0x03	Optional	Table 46 — Successful CHALLENGE_AUTH response message format
VERSION	0x04	Required	Table 10 — Successful VERSION response message format
CHUNK_SEND_ACK	0x05	Optional	Table 100 — CHUNK_SEND_ACK response message format

Response	Value	Implementation requirement	Message format
CHUNK_RESPONSE	0x06	Optional	Table 103 — CHUNK_RESPONSE response format
ENDPOINT_INFO	0x07	Optional	Table 124 — ENDPOINT_INFO response format
MEASUREMENTS	0x60	Optional	Table 53 — Successful MEASUREMENTS response message format
CAPABILITIES	0x61	Required	Table 13 — Successful CAPABILITIES response message format
SUPPORTED_EVENT_TYPES	0x62	Optional	Table 112 — SUPPORTED_EVENT_TYPES response message format
ALGORITHMS	0x63	Required	Table 22 — Successful ALGORITHMS response message format
KEY_EXCHANGE_RSP	0x64	Optional	Table 73 — Successful KEY_EXCHANGE_RSP response message format
FINISH_RSP	0x65	Optional	Table 75 — Successful FINISH_RSP response message format
PSK_EXCHANGE_RSP	0x66	Optional	Table 77 — PSK_EXCHANGE_RSP response message format
PSK_FINISH_RSP	0x67	Optional	Table 79 — Successful PSK_FINISH_RSP response message format
HEARTBEAT_ACK	0x68	Optional	Table 81 — HEARTBEAT_ACK response message format
KEY_UPDATE_ACK	0x69	Optional	Table 83 — KEY_UPDATE_ACK response message format
ENCAPSULATED_REQUEST	0x6A	Optional	Table 86 — ENCAPSULATED_REQUEST response message format
ENCAPSULATED_RESPONSE_ACK	0x6B	Optional	Table 88 — ENCAPSULATED_RESPONSE_ACK response message format
END_SESSION_ACK	0x6C	Optional	Table 91 — END_SESSION_ACK response message format

Response	Value	Implementation requirement	Message format
CSR	0x6D	Optional	Table 94 — CSR response message format
SET_CERTIFICATE_RSP	0x6E	Optional	Table 97 — Successful SET_CERTIFICATE_RSP response message format
MEASUREMENT_EXTENSION_LOG	0x6F	Optional	Table 129 — Successful MEASUREMENT_EXTENSION_LOG message format
SUBSCRIBE_EVENT_TYPES_ACK	0x70	Optional	Table 116 — SUBSCRIBE_EVENT_TYPES_ACK response message format
EVENT_ACK	0x71	Optional	Table 120 — EVENT_ACK response message format
KEY_PAIR_INFO	0x7C	Optional	Table 105 — KEY_PAIR_INFO response message format
SET_KEY_PAIR_INFO_ACK	0x7D	Optional	Table 110 — SET_KEY_PAIR_INFO_ACK response message format
VENDOR_DEFINED_RESPONSE	0x7E	Optional	Table 69 — VENDOR_DEFINED_RESPONSE response message format
ERROR	0x7F	Required	Table 59 — ERROR response message format
Reserved	All other values		SPDM implementations compatible with this version shall not use the reserved response codes.

6.2.6 SPDM request and response code issuance allowance

Table 7 — SPDM request and response messages validity describes the conditions under which a request and response can be issued.

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.

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.

A request and its corresponding response can have an "Allowed" value in both the **Session** and **Outside of a**

session columns, in which case they can be sent and received in both scenarios but might have additional restrictions. For details, see the respective request and response clauses.

A request and its corresponding response that has an *"Allowed"* value in the **Session** and *"Prohibited"* in the **Outside of a session** columns are commands used by the session. These commands only operate on the session that they were sent under, which is outside the scope of this specification. The session ID is implicit from the session used to transmit the commands.

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.

If, during the session handshake phase, an unexpected request is received using a valid session ID, the Responder shall either send an **ERROR** message in the session with `ErrorCode=UnexpectedRequest` or silently discard the request.

`Vendor-defined` shall indicate whether a `VENDOR_DEFINED_REQUEST` and `VENDOR_DEFINED_RESPONSE` is *"Allowed"* or *"Prohibited"* for use in the **Session**, **Outside of a session**, and the applicable **Session phases**.

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

Table 7 — SPDM request and response messages validity

Request	Response	Outside of a session	Session	Session phases
GET_MEASUREMENTS	MEASUREMENTS	Allowed	Allowed	Application Phase
FINISH	FINISH_RSP	Conditional (**)	Allowed	Session Handshake
PSK_FINISH	PSK_FINISH_RSP	Prohibited	Allowed	Session Handshake
HEARTBEAT	HEARTBEAT_ACK	Prohibited	Allowed	Application Phase
KEY_UPDATE	KEY_UPDATE_ACK	Prohibited	Allowed	Application Phase
END_SESSION	END_SESSION_ACK	Prohibited	Allowed	Application Phase
Not Applicable	ERROR	Allowed	Allowed	All Phases
GET_ENCAPSULATED_REQUEST	ENCAPSULATED_REQUEST	Allowed	Allowed	All Phases
DELIVER_ENCAPSULATED_RESPONSE	ENCAPSULATED_RESPONSE_ACK	Allowed	Allowed	All Phases
VENDOR_DEFINED_REQUEST	VENDOR_DEFINED_RESPONSE	Vendor-defined	Vendor-defined	Vendor-defined
CHUNK_SEND	CHUNK_SEND_ACK	Allowed	Allowed	All Phases
CHUNK_GET	CHUNK_RESPONSE	Allowed	Allowed	All Phases
GET_ENDPOINT_INFO	ENDPOINT_INFO	Allowed	Allowed	Application Phase
GET_CSR	CSR	Allowed	Allowed	Application Phase
SET_CERTIFICATE	SET_CERTIFICATE_RSP	Allowed	Allowed	Application Phase
GET_DIGESTS	DIGESTS	Allowed	Allowed	Application Phase
GET_CERTIFICATE	CERTIFICATE	Allowed	Allowed	Application Phase

Request	Response	Outside of a session	Session	Session phases
GET_KEY_PAIR_INFO	KEY_PAIR_INFO	Allowed	Allowed	Application Phase
SET_KEY_PAIR_INFO	SET_KEY_PAIR_INFO_ACK	Allowed	Allowed	Application Phase
GET_MEASUREMENT_EXTENSION_LOG	MEASUREMENT_EXTENSION_LOG	Allowed	Allowed	Application Phase
GET_SUPPORTED_EVENT_TYPES	SUPPORTED_EVENT_TYPES	Prohibited	Allowed	Application Phase
SUBSCRIBE_EVENT_TYPES	SUBSCRIBE_EVENT_TYPES_ACK	Prohibited	Allowed	Application Phase
SEND_EVENT	EVENT_ACK	Prohibited	Allowed	Application Phase
RESPOND_IF_READY	Response to Original Request (*)	Allowed (*)	Allowed (*)	All Phases (*)
All others	All others	Allowed	Prohibited	Not Applicable

(*) See [RESPOND_IF_READY request description for details](#) (**) Prohibited when `HANDSHAKE_IN_THE_CLEAR_CAP` = 0 , Allowed when `HANDSHAKE_IN_THE_CLEAR_CAP` = 1 .

6.2.7 Concurrent SPDM message processing

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

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

6.2.8 Requirements for Requesters

A Requester shall not have multiple outstanding requests to the same Responder within a connection, with the following exceptions:

- As the [GET_VERSION request and VERSION response messages](#) clause describes, a Requester can issue a `GET_VERSION` to a Responder to reset the connection at any time, even if the Requester has existing outstanding requests to the same Responder.
- In the [large SPDM message transfer mechanism](#), a single large SPDM request message and a single `CHUNK_SEND` request can be outstanding at the same time.

An outstanding request is a request where the request message has begun transmission, the corresponding response has not been fully received, and the request is not a retry as described in [Timing Requirements](#).

If the Requester has sent a request to a Responder and wants to send a subsequent request to the same Responder, then the Requester shall wait to send the subsequent request until after the Requester completes one of the following actions:

- Receives the response from the Responder for the outstanding request.
- Times out waiting for a response.
- Receives an indication from the transport layer that transmission of the request message failed.
- The Requester encounters an internal error or Reset.
- The Requester sends a `GET_VERSION` to reinitialize the session.

A Requester might send simultaneous request messages to different Responders.

6.2.9 Requirements for Responders

A Responder is not required to process more than one request message at a time, even across connections, with the following exceptions:

- As the [GET_VERSION request and VERSION response messages](#) clause describes, a Requester can issue a `GET_VERSION` to a Responder to reset a connection at any time, even if the Requester has existing outstanding requests to the same Responder.
- In the [large SPDM message transfer mechanism](#), a single large SPDM request message and a single `CHUNK_SEND` request can be outstanding at the same time.
- Retries can be issued multiple times to the same Responder, as [Timing requirements](#) defines.

A Responder that is not ready to accept a new request message or process more than one outstanding request at a time from the same Requester shall either respond with an `ERROR` message of `ErrorCode=Busy` or silently discard the request message.

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

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

6.2.10 Transcript and transcript hash calculation rules

The transcript is a concatenation of the prescribed full messages or message fields in order. In the case where a message is transferred in chunks, only the complete message that is built by the concatenation of chunk payloads shall be added to the transcript. Consequently, the transcript hash is the hash of the transcript using the negotiated hash algorithm (`BaseHashSel` or `ExtHashSel` of `ALGORITHMS`). For messages that are encrypted, the plaintext messages are used in the transcript. Where a transcript indicates that the hash of the specified certificate chain is used, the hash of the certificate chain is calculated over the specified certificate chain, as [Table 34 — Certificate chain format](#) describes. Messages that contribute to a transcript may be optional and/or conditional and will only contribute to a transcript if issued. Such messages are identified by the text "if issued" in the transcript definition. For a given message, if it does not have the "if issued" text in the transcript definition, then it is required to be present in the transcript. When an endpoint calculates the transcript hash over a series of messages, the endpoint shall ensure both the existence and the order of the messages as specified by each transcript hash calculation rule.

6.3 Timing requirements

[Table 8 — Timing specification for SPDM messages](#) shows the timing specifications for Requesters and Responders.

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

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.

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

6.3.1 Timing measurements

Unless otherwise stated, a Requester shall measure timing parameters applicable to it from the end of a successful transmission of an SPDM request to the beginning of the reception of the corresponding SPDM response. With the exception of `RD_T`, 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.

6.3.2 Timing parameters

In [Table 8 — Timing specification for SPDM messages](#), timing parameters are differentiated into two categories: the timing parameters for non-cryptographic operations (`T1`) and the timing parameters for cryptographic operations (`T2`). The timing parameters are differentiated in this manner to allow a Responder to request additional time for cryptographic operations. The timing parameters apply to normal conditions, and some operations may take additional time in some situations. For instance, a Responder may need additional time to process a non-cryptographic operation because of another operation in progress or some other condition. In this case, the Responder shall respond with an `ERROR` message of `ErrorCode=ResponseNotReady` to indicate that it needs more time.

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

6.3.3 Timing specification table

The **Ownership** column of [Table 8 — Timing specification for SPDM messages](#) specifies whether the timing parameter applies to the Responder or Requester. For *encapsulated requests*, the Requester shall comply with the timing parameters where the **Ownership** indicates a Responder.

Table 8 — Timing specification for SPDM messages

Timing parameter	Ownership	Value	Units	Description
RTT	Requester	See the description.	µs	This value shall be the worst-case round-trip transport timing. The value shall be the worst-case total time for the complete transmission and delivery of an SPDM message round trip at the transport layer(s). The actual value for this parameter is transport- or media-specific. Both the actual value and how an endpoint obtains this value are outside the scope of this specification. A Requester shall measure this timing parameter from the end of a successful transmission of an SPDM request to the beginning of the reception of the corresponding SPDM response less <code>ST1</code> or <code>CT</code>, depending on the Request.
ST1	Responder	100,000	µs	This value shall be the maximum amount of time the Responder has to provide a response under normal conditions to requests that do not require cryptographic processing, such as the <code>GET_CAPABILITIES</code>, <code>GET_VERSION</code>, or <code>NEGOTIATE_ALGORITHMS</code> request messages. See Table 12 — <code>GET_CAPABILITIES</code> request message format, Table 9 — <code>GET_VERSION</code> request message format, and Table 16 — <code>NEGOTIATE_ALGORITHMS</code> request message format.
T1	Requester	<code>RTT</code> + <code>ST1</code>	µs	This value shall be the minimum amount of time the Requester shall wait before issuing a retry for requests that do not require cryptographic processing. For details, see the <code>ST1</code> timing parameter.

Timing parameter	Ownership	Value	Units	Description
				CTExponent is reported in the GET_CAPABILITIES request message and CAPABILITIES response message. This parameter is applicable to both a Responder and Requester as the Ownership column shows. Specifically for a Requester, this field is applicable when the Requester provides a response that requires cryptographic processing such as in the mutual authentication portion of a KEY_EXCHANGE flow. When the Requester provides a response that requires cryptographic processing, the Requester shall measure timing just as a Responder would.
CT	Requester and Responder	2 ^{CTExponent}	µs	This timing parameter shall be the maximum amount of time the endpoint has to provide any response requiring cryptographic processing under normal conditions, such as the GET_MEASUREMENTS or CHALLENGE request messages. If the Responder cannot respond within CT, the Responder shall respond with an ERROR message of ErrorCode=ResponseNotReady to indicate that it needs more time. See Table 12 — GET_CAPABILITIES request message format, Table 13 — Successful CAPABILITIES response message format, Table 50 — GET_MEASUREMENTS request message format, and Table 45 — CHALLENGE request message format.

Timing parameter	Ownership	Value	Units	Description
T2	Requester	$RTT + CT$	μs	This value shall be the minimum amount of time the Requester shall wait before issuing a retry for requests that require cryptographic processing. For details, see the <code>CT</code> timing parameter.
RDT	Responder	$2^{RDTE\text{Exponent}}$	μs	This value shall be the 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 <code>CT</code> time, it shall provide <code>RDTEExponent</code> as part of the <code>ERROR</code> response as Table 59 — <code>ERROR</code> response message format shows. For details, see <code>ErrorCode=ResponseNotReady</code> in Table 61 — <code>ResponseNotReady</code> extended error data for the <code>RDTEExponent</code> value. An SPDM Responder measures the <code>RDT</code> value from the end of the transmission of the <code>ERROR</code> message of <code>ErrorCode=ResponseNotReady</code>, to the beginning of the reception of the next <code>RESPOND_IF_READY</code> request message.

Timing parameter	Ownership	Value	Units	Description
WT	Requester	RDT	µs	This value shall be the amount of time that the Requester should wait before issuing the RESPOND_IF_READY request message as Table 67 — RESPOND_IF_READY request message format shows. The Requester shall measure this time parameter from the reception of the ERROR response to the transmission of the 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. For details, see the RDT timing parameter.

Timing parameter	Ownership	Value	Units	Description
WT _{Max}	Requester	$(RDT * RDTM) - RTT$	μs	This value shall be the maximum wait time the Requester has to issue the RESPOND_IF_READY request message, as Table 67 — RESPOND_IF_READY request message format shows, unless the Requester issued a successful RESPOND_IF_READY request message, as Table 67 — RESPOND_IF_READY request message format shows, earlier. The Requester shall start measuring time from the reception of the first ERROR message of ErrorCode=ResponseNotReady with the same Token until WT_{Max} μs elapses or the corresponding Response is successfully received. After this time has passed, the Responder is allowed to drop the response. The RESPOND_IF_READY message follows the most recently received ERROR message with ErrorCode = ResponseNotReady, which shall specify the wait time for that cycle. The Requester shall take into account the transmission time of the ERROR response, as Table 59 — ERROR response message format shows, from the Responder to Requester when calculating WT_{Max}. The RDTM value appears in Table 61 — ResponseNotReady extended error data. The Responder should ensure that WT_{Max} does not result in less than WT in determination of RDTM. See ErrorCode=ResponseNotReady in Table 61 — ResponseNotReady extended error data.

Timing parameter	Ownership	Value	Units	Description
HeartbeatPeriod	Requester and Responder	Variable	s	See the HEARTBEAT request and HEARTBEAT_ACK response clause.

6.4 SPDM messages

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

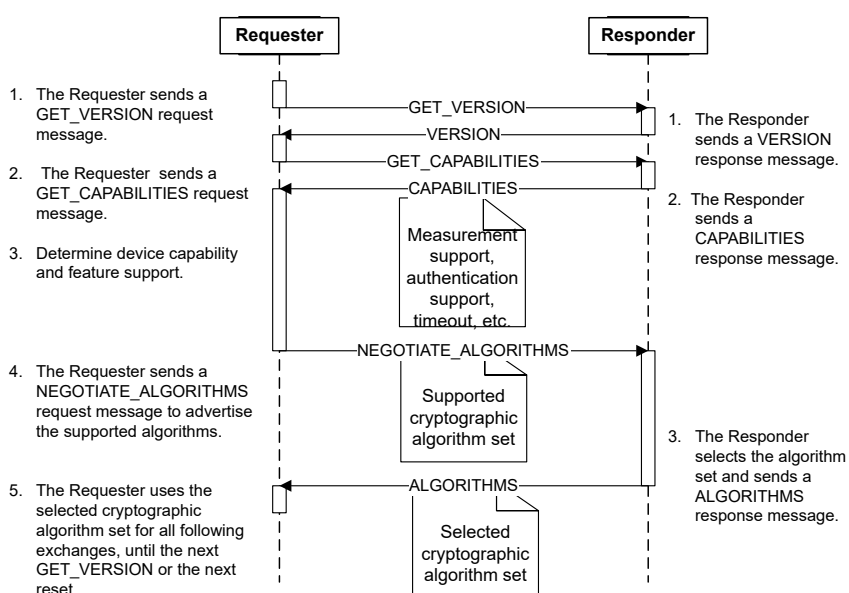
- [Capability discovery and negotiation](#)
- [Responder identity authentication](#)
- [Measurement](#)
- [Key agreement for secure-channel establishment](#)

6.4.1 Capability discovery and negotiation

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

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

Figure 5 — Capability discovery and negotiation flow



6.4.1.1 Negotiated state preamble

The [vca](#) ([Version-Capabilities-Algorithms](#)) shall be the concatenation of messages `GET_VERSION`, `VERSION`, `GET_CAPABILITIES`, `CAPABILITIES`, `NEGOTIATE_ALGORITHMS`, and `ALGORITHMS` last exchanged between the Requester and the Responder.

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

two endpoints support PSK but the negotiated capabilities and algorithms are not provisioned to both endpoints alongside the PSK, then the Requester shall issue `GET_VERSION`, `GET_CAPABILITIES`, and `NEGOTIATE_ALGORITHMS` to construct `VCA`.

If the Responder supports caching the negotiated state (`CACHE_CAP=1`), the Requester might not issue `GET_VERSION`, `GET_CAPABILITIES`, and `NEGOTIATE_ALGORITHMS`. In this case, the Requester and the Responder shall store the most recent `VCA` as part of the Negotiated State.

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

6.4.2 `GET_VERSION` request and `VERSION` response messages

This request message shall retrieve the SPDM version of an endpoint. [Table 9 — `GET\_VERSION` request message format](#) shows the `GET_VERSION` request message format and [Table 10 — Successful `VERSION` response message format](#) shows the `VERSION` response message format.

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

The Requester shall begin the discovery process by sending a `GET_VERSION` request message with the value of the `SPDMVersion` field set to `0x10`. All Responders shall always support the `GET_VERSION` request message with major version `0x1` and provide a `VERSION` response containing all supported versions, as [Table 9 — `GET\_VERSION` request message format](#) describes.

The Requester shall consult the `VERSION` response to select a common supported version, which should be the latest supported common version. The Requester shall use the selected version in all future communication of other requests. A Requester shall not issue other requests until it receives a successful `VERSION` response and identifies a common version that both sides support. A Responder shall not respond to the `GET_VERSION` request message with an `ERROR` message except for `ErrorCode`s specified in this clause. The selected version shall be the version in the `SPDMVersion` field of the Request (other than `GET_VERSION`) immediately following the `GET_VERSION` request. If the Requester uses a version other than the selected version in a Request, the Responder should either return an `ERROR` message of `ErrorCode=VersionMismatch` or silently discard the Request.

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

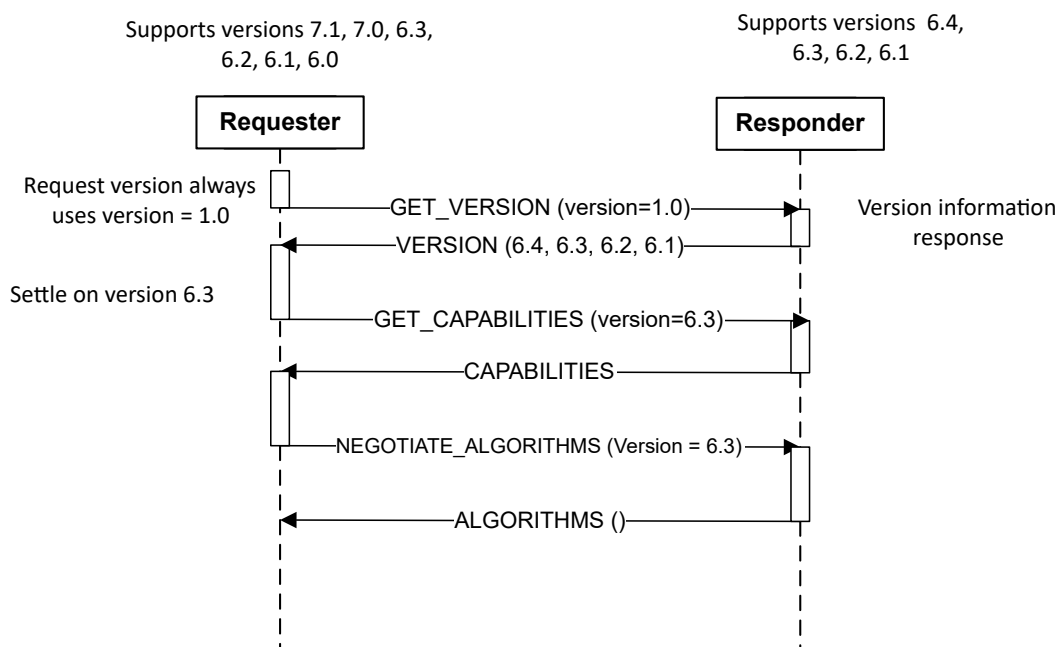
After receiving a valid `GET_VERSION` request, the Responder shall invalidate state and data associated with all previous requests from the same Requester. All active sessions between the Requester and the Responder are terminated, and information (such as session keys and session IDs) for those sessions should not be used anymore. Additionally, this message shall clear the previously [Negotiated State](#), if any, in both the Requester and its corresponding Responder. An invalid `GET_VERSION` request that results in the Responder returning an error to the Requester shall not affect the connection state. The `ERROR` message resulting from an invalid `GET_VERSION` request shall have the value of the `SPDMVersion` field set to `0x10`.

After sending the `VERSION` response for a `GET_VERSION` request, if the Responder completes a runtime code or configuration change for its hardware or firmware measurement and the change has taken effect, then the Responder shall either silently discard any request received outside of a session or respond with an `ERROR` message of `ErrorCode=RequestResynch` to any request received outside of a session, until a `GET_VERSION`

request is received. For requests received within a session, the Responder shall follow the selected session policy that the Requester selects in [Table 72 — Session policy](#) at the time of session establishment.

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

Figure 6 — Discovering the common major version



[Table 9 — GET_VERSION request message format](#) shows the `GET_VERSION` request message format:

Table 9 — GET_VERSION request message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be <code>0x10</code> (V1.0).
1	RequestResponseCode	1	Shall be <code>0x84</code> = <code>GET_VERSION</code> . See Table 5 — SPDM request codes .
2	Param1	1	Reserved.
3	Param2	1	Reserved.

[Table 10 — Successful VERSION response message format](#) shows the successful `VERSION` response message format:

Table 10 — Successful VERSION response message format

Byte offset	Field	Size (bytes)	Description
0	SPDMVersion	1	Shall be <code>0x10</code> (V1.0).
1	RequestResponseCode	1	Shall be <code>0x04</code> = <code>VERSION</code> . See Table 6 — SPDM response codes .

Byte offset	Field	Size (bytes)	Description
2	Param1	1	Reserved.
3	Param2	1	Reserved.
4	Reserved	1	Reserved.
5	VersionNumberEntryCount	1	Number of version entries present in this table (=n).
6	VersionNumberEntry1:n	2 * n	16-bit version entry. See Table 11 — VersionNumberEntry definition . Each entry should be unique.

[Table 11 — VersionNumberEntry definition](#) shows the `VersionNumberEntry` definition. See [Version encoding](#) for more details.

Table 11 — VersionNumberEntry definition

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

6.4.3 GET_CAPABILITIES request and CAPABILITIES response messages

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

[Table 12 — GET_CAPABILITIES request message format](#) shows the `GET_CAPABILITIES` request message format.

[Table 13 — Successful CAPABILITIES response message format](#) shows the `CAPABILITIES` response message format.

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

Likewise, [Table 15 — Flag fields definitions for the Responder](#) shows the flag fields definitions for the Responder.

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

- One for receiving a single SPDM transfer called `DataTransferSize`
- One for indicating their maximum internal buffer size for processing a single assembled received SPDM