



Document Identifier: DSP0274

Date: 2019-10-18

Version: 0.99.0a

# Security Protocol and Data Model (SPDM) Specification

## Information for Work-in-Progress version:

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 This document is available for public review and comment until superseded.

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

**Supersedes:** 0.95.0a

**Document Class:** Normative

**Document Status:** Work In Progress

**Document Language:** en-US

**Copyright Notice**

Copyright © 2019 DMTF. All rights reserved.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. Members and non-members may reproduce DMTF specifications and documents, provided that correct attribution is given. As DMTF specifications may be revised from time to time, the particular version and release date should always be noted.

Implementation of certain elements of this standard or proposed standard may be subject to third party patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose, or identify any or all such third party patent right, owners or claimants, nor for any incomplete or inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize, disclose, or identify any such third party patent rights, or for such party's reliance on the standard or incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any party implementing such standard, whether such implementation is foreseeable or not, nor to any patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a patent owner for such implementations.

For information about patents held by third-parties which have notified the DMTF that, in their opinion, such patent may relate to or impact implementations of DMTF standards, visit <http://www.dmtf.org/about/policies/disclosures.php>.

This document's normative language is English. Translation into other languages is permitted.

## CONTENTS

1 Foreword	5
2 Acknowledgments	6
3 Abstract	7
4 Document conventions	8
4.1 Scope	8
4.2 Normative references	8
4.3 Terms and definitions	10
4.4 Symbols and abbreviated terms	12
4.5 Conventions	13
4.5.1 Reserved and unassigned values	13
4.5.2 Byte ordering	13
4.5.3 SPDM data types	13
4.5.4 Version encoding	13
4.5.5 Notations	14
4.6 SPDM message exchanges	14
4.6.1 Security capability discovery and negotiation	15
4.6.2 Identity authentication	15
4.6.3 Firmware and configuration measurement	16
4.7 SPDM messaging protocol	16
4.7.1 Generic SPDM message format	18
4.7.2 SPDM request codes	18
4.7.3 SPDM response codes	19
4.8 Concurrent SPDM message processing	20
4.8.1 Requirements for Requesters	20
4.8.2 Requirements for Responders	21
4.8.3 Timing requirements	21
4.8.3.0.1 Timing measurements	21
4.8.3.1 Timing specification table	21
4.9 SPDM messages	23
4.9.1 Capability discovery and negotiation	23
4.9.1.1 GET_VERSION request message and VERSION response message	24
4.9.1.2 GET_CAPABILITIES request message and CAPABILITIES response message	26
4.9.1.3 NEGOTIATE_ALGORITHMS request message and ALGORITHMS response message	28
4.9.2 Responder identity authentication	33
4.9.2.1 Certificates and certificate chains	34
4.9.2.2 GET_DIGESTS request message and DIGESTS response message	35
4.9.2.3 GET_CERTIFICATE request message and CERTIFICATE response message	36
4.9.2.4 Leaf certificate	11
4.9.2.4.1 Required fields	38
4.9.2.4.2 Optional fields	38
4.9.2.4.3 Definition of othername using the DMTF OID	38

4.9.2.5 CHALLENGE request message and CHALLENGE_AUTH response message . . . . .	39
4.9.2.6 CHALLENGE_AUTH Signature generation . . . . .	41
4.9.2.7 CHALLENGE_AUTH Signature verification . . . . .	42
4.10 Request ordering and message transcript computation rules for M1 and M2 . . . . .	43
4.10.1 Firmware and other measurements . . . . .	44
4.10.1.1 GET_MEASUREMENTS request message and MEASUREMENTS response message . . . . .	45
4.10.1.2 Measurement block . . . . .	47
4.10.1.3 DMTF Specification for the Measurement field of a Measurement block . . . . .	47
4.10.1.4 MEASUREMENTS Signature generation . . . . .	48
4.10.1.5 MEASUREMENTS Signature verification . . . . .	49
4.10.2 ERROR response message . . . . .	51
4.10.3 RESPOND_IF_READY request message . . . . .	54
4.10.4 VENDOR_DEFINED_REQUEST request message . . . . .	55
4.10.5 VENDOR_DEFINED_RESPONSE response message . . . . .	56
4.11 SPDM messaging control and discovery examples . . . . .	56
4.12 ANNEX A (informative) . . . . .	56
4.13 ANNEX B - Leaf certificate example . . . . .	57
4.14 Change log . . . . .	58
4.15 Bibliography . . . . .	58

# 1 Foreword

---

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

## 2 Acknowledgments

---

The DMTF acknowledges these individuals' contributions to this document:

**Contributors:**

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

## 3 Abstract

---

The *Security Protocol and Data Model (SPDM) Specification* defines *messages*, data objects, and sequences for performing message exchanges between *devices* over a variety of transport and physical media. The description of message exchanges includes *authentication* of hardware identities and measurement for firmware identities. The SPDM enables efficient access to low-level security capabilities and operations. The SPDM can be used with other mechanisms, including non-PMCI- and DMTF-defined mechanisms.

## 4 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.

### 4.1 Scope

---

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

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

### 4.2 Normative references

---

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

- *ISO/IEC Directives, Part 2, Principles and rules for the structure and drafting of ISO and IEC documents*, <https://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype>
- IETF RFC5234, *Augmented BNF for Syntax Specifications: ABNF*, January 2008, <https://tools.ietf.org/html/rfc5234>
- USB Authentication Specification Rev 1.0 with ECN and Errata through January 7, 2019 <https://www.usb.org/sites/default/files/USB%20Authentication%20Specification%20Rev%201.0%20with%20ECN%20and%20Errata%20through%20January%207%2C%202019.zip>
- TCG Algorithm Registry, Family "2.0", Level 00 Revision 01.27, February 7, 2018 <https://trustedcomputinggroup.org/resource/tcg-algorithm-registry/>

**ASN.1 — ISO-822-1-4**



- ITU-T X.680

Available at: [https://www.itu.int/rec/dologin\\_pub.asp?lang=e&id=T-REC-X.680-201508-!!!PDF-E&type=items](https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.680-201508-!!!PDF-E&type=items)

- ITU-T X.681

Available at: [https://www.itu.int/rec/dologin\\_pub.asp?lang=e&id=T-REC-X.681-201508-!!!PDF-E&type=items](https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.681-201508-!!!PDF-E&type=items)

- ITU-T X.682

Available at: [https://www.itu.int/rec/dologin\\_pub.asp?lang=e&id=T-REC-X.682-201508-!!!PDF-E&type=items](https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.682-201508-!!!PDF-E&type=items)

- ITU-T X.683

Available at: [https://www.itu.int/rec/dologin\\_pub.asp?lang=e&id=T-REC-X.683-201508-!!!PDF-E&type=items](https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.683-201508-!!!PDF-E&type=items)

#### **DER — ISO-8825-1**

- ITU-T X.690

Available at: [https://www.itu.int/rec/dologin\\_pub.asp?lang=e&id=T-REC-X.690-201508-!!!PDF-E&type=items](https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.690-201508-!!!PDF-E&type=items)

#### **X509v3 — ISO-9594-8**

- ITU-T X.509

Available at: [https://www.itu.int/rec/dologin\\_pub.asp?lang=e&id=T-REC-X.509-201210-!!!PDF-E&type=items](https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-201210-!!!PDF-E&type=items)

#### **ECDSA**

- NIST-FIPS-186-4, Section 6

Available at: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>

- NIST P256, secp256r1; NIST P384, secp384r1; NIST P521, secp521r1: NIST-FIPS-186-4, Appendix D

Available at: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>

#### **RSA**

- As defined in TCG Algorithm Registry, Family "2.0", Level 00 Revision 01.27, February 7, 2018, Table 3:  
[https://trustedcomputinggroup.org/wp-content/uploads/TCG\\_Algorithm\\_Registry\\_Rev\\_1.22.pdf](https://trustedcomputinggroup.org/wp-content/uploads/TCG_Algorithm_Registry_Rev_1.22.pdf)

#### **SHA2-256, SHA2-384, and SHA2-512**

- FIPS PUB 180-4 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Secure Hash Standard (SHS)

Available at: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>

#### SHA3-256, SHA3-384, and SHA3-512

- FIPS PUB 202 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions

Available at: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>

## 4.3 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, or annexes labeled "(informative)" do not contain normative content. Notes and examples are always informative elements.

This specification uses these terms:

Term	Definition
authentication	Process of determining whether an entity is who or what it claims to be.
authentication initiator	Endpoint that initiates the authentication process by challenging another endpoint.
byte	Eight-bit quantity. Also known as an <i>octet</i> . <b>Note:</b> SPDM specifications shall use the term byte, not octet.
certificate	Digital form of identification that provides information about an entity and certifies ownership of a particular asymmetric key-pair.

Term	Definition
certificate authority (CA)	Trusted third-party entity that issues certificates.
certificate chain	Series of two or more certificates. Each certificate is signed by the preceding certificate in the chain.
certificate signing request (CSR)	One of the first steps towards getting a certificate.
component	Similar to the PCI Express specification's definition, a physical entity.
device	Physical entity such as a network card or a fan.
endpoint	Logical entity that communicates with other endpoints over one or more transport protocol.
intermediate certificate	Certificate that is neither a root certificate nor a leaf certificate.
leaf certificate	Last certificate in a certificate chain.
message	See <a href="#">SPDM message</a> .
message body	Portion of an SPDM message that carries additional data.
message originator	Original transmitter, or source, of an SPDM message.
message transcript	The concatenation of a sequence of messages in the order in which they are sent and received by an endpoint. The final message included in the message transcript may be truncated to allow inclusion of a signature in that message which is computed over the message transcript.
most significant byte (MSB)	Highest order <a href="#">byte</a> in a number consisting of multiple bytes.
Negotiated State	A set of parameters that represent the state of the communication between a corresponding pair of Requester and Responder at the successful completion of the <code>NEGOTIATE_ALGORITHMS</code> messages. 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 in order to continue or preserve communication with each other.
nibble	Computer term for a four-bit aggregation, or half of a byte.
nonce	Number that is unpredictable to entities other than its generator. The probability of the same number occurring more than once is negligible. Nonce may be generated by combining a pseudo random number of at least 64 bits, optionally concatenated with a monotonic counter of size suitable for the application.
payload	Information-bearing fields of a message. These fields are separate from the fields and elements, such as address fields, framing bits, checksums, and so on, that transport the message from one point to another. In some instances, a field can be both a payload field and a transport field.
physical transport binding	Specifications that define how a base messaging protocol is implemented on a particular physical transport type and medium, such as SMBus/I <sup>2</sup> C, PCI Express™ Vendor Defined Messaging, and so on.
SPDM message	Unit of communication in SPDM communications.

Term	Definition
SPDM message payload	Portion of the message body of an SPDM message. This portion of the message is separate from those fields and elements that identify the SPDM version, the SPDM request and response codes, and the two parameters.
SPDM request message	Message that is sent to an endpoint to request a specific SPDM operation. A corresponding SPDM response message acknowledges receipt of an SPDM request message.
SPDM response message	Message that is sent in response to a specific SPDM request message. This message includes a <code>Response Code</code> field that indicates whether the request completed normally.
Platform Management Component Intercommunications (PMCI)	Name of a working group under the Distributed Management Task Force that defines standardized communication protocols, low-level data models, and transport definitions that support communications with and between management controllers and management devices that form a platform management subsystem within a managed computer system.
Requester	Original transmitter, or source, of an SPDM request message. It is also the ultimate receiver, or destination, of an SPDM response message.
Responder	Ultimate receiver, or destination, of an SPDM request message. It is also the original transmitter, or source of an SPDM response message.
root certificate	First certificate in a certificate chain, which is self-signed.
Trusted Computing Base (TCB)	(Reference: <a href="https://en.wikipedia.org/wiki/Trusted_computing_base">https://en.wikipedia.org/wiki/Trusted_computing_base</a> ) The trusted computing base (TCB) of a computer system is the set of all hardware, firmware, and/or software components that are critical to its security, in the sense that bugs or vulnerabilities occurring inside the TCB might jeopardize the security properties of the entire system. By contrast, parts of a computer system outside the TCB must not be able to misbehave in a way that would leak any more privileges than are granted to them in accordance to the security policy.

## 4.4 Symbols and abbreviated terms

This specification uses these abbreviations:

Abbreviation	Definition
CA	<i>certificate authority</i>
CSR	<i>certificate signing request</i>
MSB	<i>most significant byte</i>
PMCI	Platform Management Component Intercommunications
SPDM	Security Protocol and Data Model

## 4.5 Conventions

The following conventions apply to all SPDM specifications.

### 4.5.1 Reserved and unassigned values

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

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

### 4.5.2 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).

### 4.5.3 SPDM data types

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

**Table 1: SPDM data types**

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

### 4.5.4 Version encoding

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

Version	Matches	Incremented when
Major	Major version field in the <code>SPDMVersion</code> field in the SPDM message header.	Protocol modification breaks backward compatibility.

Version	Matches	Incremented when
Minor	Minor version field in the <code>SPDMVersion</code> field in the SPDM message header.	Protocol modification maintains backward compatibility.

EXAMPLE:

Version 3.7 → `0x37`

Version 1.0 → `0x10`

Version 1.2 → `0x12`

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

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

The detailed version encoding returned by the `VERSION` response message contains an additional byte indicating specification bug fixes or development versions. See [VersionNumberEntry definition](#).

### 4.5.5 Notations

The following notations are used for SPDM specifications:

Notation	Description
<code>M:N</code>	In field descriptions, this notation typically represents a range of byte offsets starting from byte M and continuing to and including byte N ( $M \leq N$ ). 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 [LSb] offset = 0.
<code>[7:5]</code>	A range of bit offsets. The most significant is on the left, and the least significant is on the right.
<code>1b</code>	A lowercase <code>b</code> after a number consisting of 0s and 1s indicates that the number is in binary format.
<code>0x12A</code>	A leading <code>0x</code> indicates that the number is in hexadecimal format.

## 4.6 SPDM message exchanges

The message exchanges defined in this specification are between two endpoints and are performed and exchanged through sending and receiving of SPDM messages defined in [SPDM messages](#). The SPDM message exchanges are

defined in a generic fashion that allows the messages to be communicated across different physical mediums and over different transport protocols.

The two endpoints have a role of either a Requester or Responder. All messages are paired as command/response with the Requester initiating all communication and the Responder replying to the communication.

Endpoints may implement both Requester and Responder capabilities. It is possible for a pair of endpoints to be involved with two SPDM message streams between each other with each endpoint having a Requester role and a Responder role. These two streams are mutually exclusive.

The message exchanges defined in this specification include Requesters that:

1. Discover and negotiate the security capabilities of a Responder.
2. Authenticate the identity of a Responder.
3. Retrieve the firmware measurement of a Responder.

These message exchange capabilities are built on top of well-known and established security practices across the computing industry. A brief overview for each of the message exchange capabilities is described in the following sections. Some of the message exchange capabilities are based on the security model defined in USB Authentication Specification Rev 1.0.

#### 4.6.1 Security capability discovery and negotiation

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

#### 4.6.2 Identity authentication

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

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

- **Identity provisioning**

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

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

- **Runtime authentication**

The process by which an authentication initiator (Requester) interacts with a Responder in a running system. The authentication initiator can retrieve the certificate chain(s) from the Responder and send a unique challenge to the Responder. The Responder then signs the challenge with the private key. The authentication initiator verifies the signature using the public key of the Responder as well as any intermediate public keys within the certificate chain using the root certificate as the trusted anchor.

### 4.6.3 Firmware and configuration measurement

Measurement is a term that describes the process of calculating the cryptographic hash value of a piece of firmware/software or configuration data and tying the cryptographic hash value with the endpoint identity through the use of digital signatures. This allows an authentication initiator to establish that the identity and measurement of the firmware/software or configuration running on the endpoint.

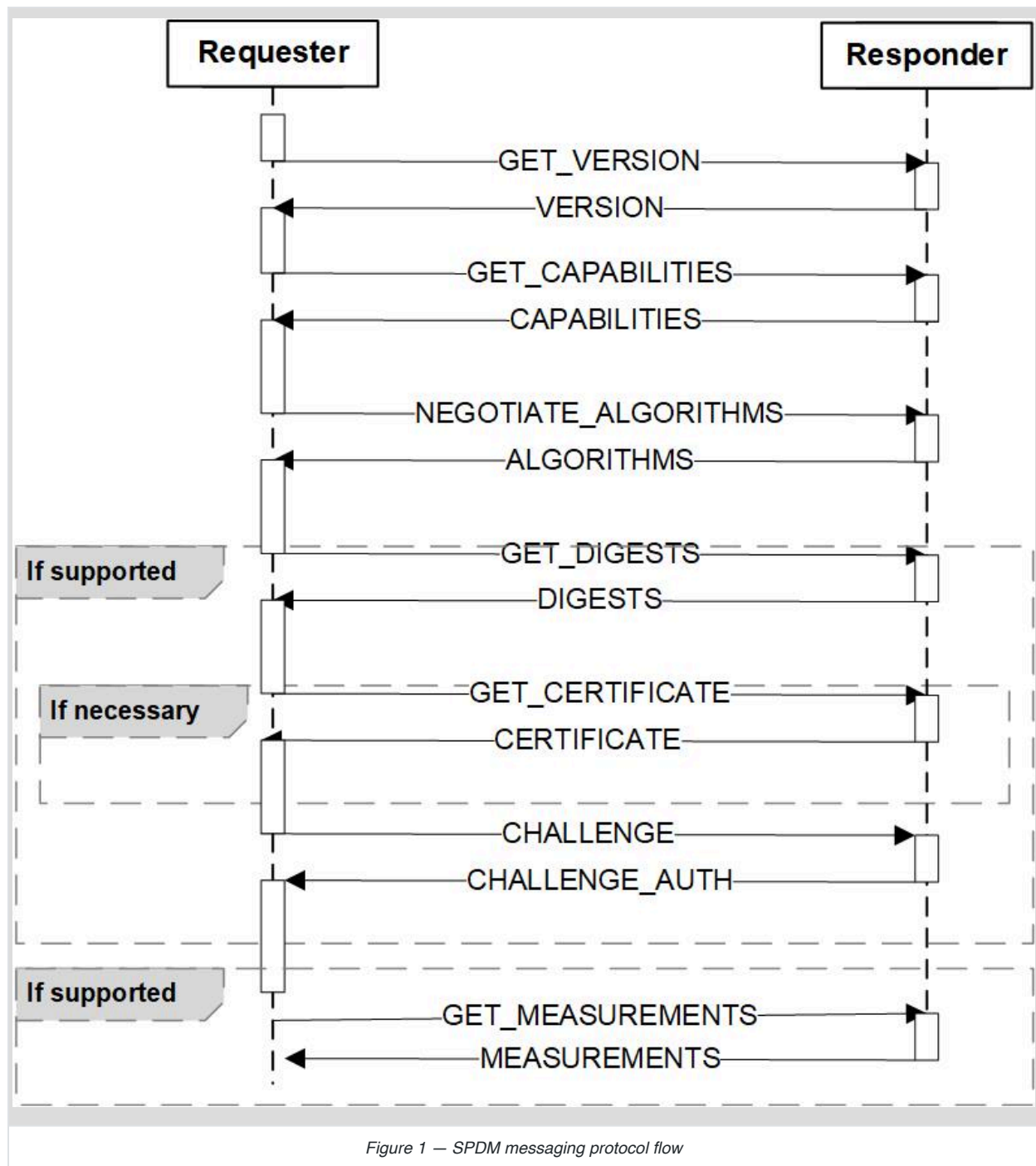
## 4.7 SPDM messaging protocol

---

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 defined in this specification unless otherwise stated in this specification.

[Figure 1](#) depicts the high-level request-response flow diagram for SPDM. An endpoint that acts as the Requester sends an SPDM request message to another endpoint that acts as the [Responder](#), and the Responder returns an SPDM response message to the Requester.





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

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

### 4.7.1 Generic SPDM message format

[Table 2](#) defines the fields that constitute a generic SPDM message, including the message header and payload. The fields within the SPDM messages are transferred from the lowest offset first.

**Table 2 — Generic SPDM message formats**

Byte 1								Byte 2								Byte 3								Byte 4							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
SPDM Major Version				SPDM Minor Version				Request Response Code								Param1				Param2											
SPDM message payload (zero or more bytes)																															

[Table 3](#) defines the fields that are part of a generic SPDM message.

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

Field name	Field size (bits)	Description
SPDM Major Version	4	The major version of the SPDM Specification. An endpoint shall not communicate by using an incompatible SPDM version value. See <a href="#">Version encoding</a> .
SPDM Minor Version	4	The minor version of the SPDM Specification. A specification with a given minor version extends a specification with a lower minor version as long as they share the major version. See <a href="#">Version encoding</a> .
Request Response Code	8	The request message code or response code, which are enumerated in <a href="#">Table 4</a> and <a href="#">Table 5</a> . <code>0x00</code> through <code>0x7F</code> represent response codes and <code>0x80</code> through <code>0xFF</code> represent request codes.
Param1	8	The first one-byte parameter. The contents of the parameter is specific to the Request Response Code.
Param2	8	The second one-byte parameter. The contents of the parameter is specific to the Request Response Code.
SPDM message payload	Variable	Zero or more bytes that are specific to the Request Response Code.

### 4.7.2 SPDM request codes

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

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

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

**Table 4 — SPDM request codes**

Request	Code value	Implementation Requirement	Message format
GET_DIGESTS	0x81	Optional	See <a href="#">Table 16</a> .
GET_CERTIFICATE	0x82	Optional	See <a href="#">Table 18</a> .
CHALLENGE	0x83	Optional	See <a href="#">Table 20</a> .
GET_VERSION	0x84	Required	See <a href="#">Table 7</a> .
GET_MEASUREMENTS	0xE0	Optional	See <a href="#">Table 22</a> .
GET_CAPABILITIES	0xE1	Required	See <a href="#">Table 10</a> .
NEGOTIATE_ALGORITHMS	0xE3	Required	See <a href="#">Table 13</a> .
RESPOND_IF_READY	0xFF	Required	See <a href="#">Table 31</a> .
VENDOR_DEFINED_REQUEST	0xFE	Optional	See <a href="#">Table 104</a> .
Reserved	0x80 , 0x85 - 0xDF , 0xE2 , 0xE4 - 0xFD	SPDM implementations compatible with this version shall not use the reserved request codes.	

### 4.7.3 SPDM response codes

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

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

**Table 5 — SPDM response codes**

Response	Value	Implementation Requirement	Message format
DIGESTS	0x01	Optional	See <a href="#">Table 16</a> .

Response	Value	Implementation Requirement	Message format
CERTIFICATE	0x02	Optional	See <a href="#">Table 18</a> .
CHALLENGE_AUTH	0x03	Optional	See <a href="#">Table 20</a> .
VERSION	0x04	Required	See <a href="#">Table 8</a> .
MEASUREMENTS	0x60	optional	See <a href="#">Table 24</a> .
CAPABILITIES	0x61	Required	See <a href="#">Table 11</a> .
ALGORITHMS	0x63	Required	See <a href="#">Table 14</a> .
VENDOR_DEFINED_RESPONSE	0x7E	Optional	See <a href="#">Table 105</a> .
ERROR	0x7F	See <a href="#">Table 26</a> .	
Reserved	0x00 , 0x05 - 0x5F , 0x62 , 0x64 - 0x7D	SPDM implementations compatible with this version shall not use the reserved response codes.	

## 4.8 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.

### 4.8.1 Requirements for Requesters

A Requester shall not have multiple outstanding requests to the same Responder, with the exception of `GET_VERSION` addressed in [GET\\_VERSION request message and VERSION response message](#). If the Requester has sent a request to a Responder and wants to send a subsequent request to the same Responder, then the Requester shall wait to send the subsequent request until after the Requester completes one of the following actions:

- Receives the response from the Responder for the outstanding request.
- Times out waiting for a response.
- Receives an indication, from the transport layer, that transmission of the request message failed.

A Requester may send simultaneous request messages to different Responders.

## 4.8.2 Requirements for Responders

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

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

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

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

## 4.8.3 Timing requirements

Table 6 shows the timing specifications for Requesters and Responders.

If the Requester does not receive a response within **T1** or **T2** time accordingly, the Requester may retry a request message. A retry of a request message shall be a complete retransmission of the original SPDM request message.

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

### 4.8.3.0.1 Timing measurements

A Requester shall measure timing parameters, applicable to it, from the end of a successful transmission of an SPDM request to the beginning of the reception of the corresponding SPDM response. A Responder shall measure timing parameters, applicable to it, from the end of the reception of the SPDM request to the beginning of transmission of the response.

### 4.8.3.1 Timing specification table

In Table 6, the **Ownership** column specifies whether the timing parameter applies to the Responder or Requester.

**Table 6 — Timing specification for SPDM Messages**

Timing Parameter	Ownership	Value	Units	Description
RTT	Requester	See Description	See Description	<p>This is the worst case round trip transport timing.</p> <p>The max 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/media specific.</p>
ST1	Responder	100	ms	<p>This shall be the maximum amount of time the Responder has to provide a response to requests that do not require cryptographic processing, such as <a href="#">GET_CAPABILITIES</a> , <a href="#">GET_VERSION</a> or <a href="#">NEGOTIATE_ALGORITHMS</a> .</p>
T1	Requester	RTT + ST1	ms	<p>This shall be the minimum amount of time the Requester shall wait before issuing a retry for requests that do not require cryptographic processing.</p> <p>For details, see <a href="#">ST1</a> .</p>
CT	Responder	$2^{CTExponent}$	us	<p>This is the cryptographic timeout in microseconds. <code>CTExponent</code> is reported in the <a href="#">CAPABILITIES</a> message.</p> <p>This timing parameter shall be the maximum amount of time the Responder has to provide any response requiring cryptographic processing, such as <a href="#">GET_MEASUREMENTS</a> and <a href="#">CHALLENGE</a> .</p>
T2	Requester	RTT + CT	us	<p>This shall be the minimum amount of time the Requester shall wait before issuing a retry for requests that require cryptographic processing.</p> <p>For details, see <a href="#">CT</a> .</p>
RDT	Responder	$2^{RDTEExponent}$	us	<p>This is the Recommended Delay in microseconds. When the Responder is unable to complete cryptographic processing response within the <code>CT</code> time, it shall provide <code>RDTEExponent</code> as part of the <a href="#">ERROR</a> Response. See <a href="#">Table 28</a> for the <code>RDTEExponent</code> value.</p> <p>For details, see <a href="#">ErrorCode=ResponseNotReady</a> .</p>
WT	Requester	RDT	us	<p>This is the amount of time the Requester should wait before issuing <a href="#">RESPOND_IF_READY</a> request. The Requester shall measure this time parameter from the reception of the <code>ERROR</code> response to the transmission of <a href="#">RESPOND_IF_READY</a> request. The Requester may take into account the transission time of the <code>ERROR</code> from the Responder to Requester when calculating <code>WT</code> .</p> <p>For details, see <a href="#">RDT</a> .</p>

Timing Parameter	Ownership	Value	Units	Description
WT <sub>Max</sub>	Requester	(RDT * RDTM) - RTT	us	<p>This is the maximum wait time the Requester has to to issue <a href="#">RESPOND_IF_READY</a> request unless the Requester issued a successful <a href="#">RESPOND_IF_READY</a> earlier. After this time the Responder is allowed to drop the response. The Requester shall take into account the transmission time of the <a href="#">ERROR</a> from the Responder to Requester when calculating WT<sub>Max</sub>. The value of RDTM is given in <a href="#">Table 28</a>. The Responder should ensure <math>WT_{Max}</math> does not result less than WT in determination of RDTM.</p> <p>For details, see <a href="#">ErrorCode=ResponseNotReady</a>.</p>

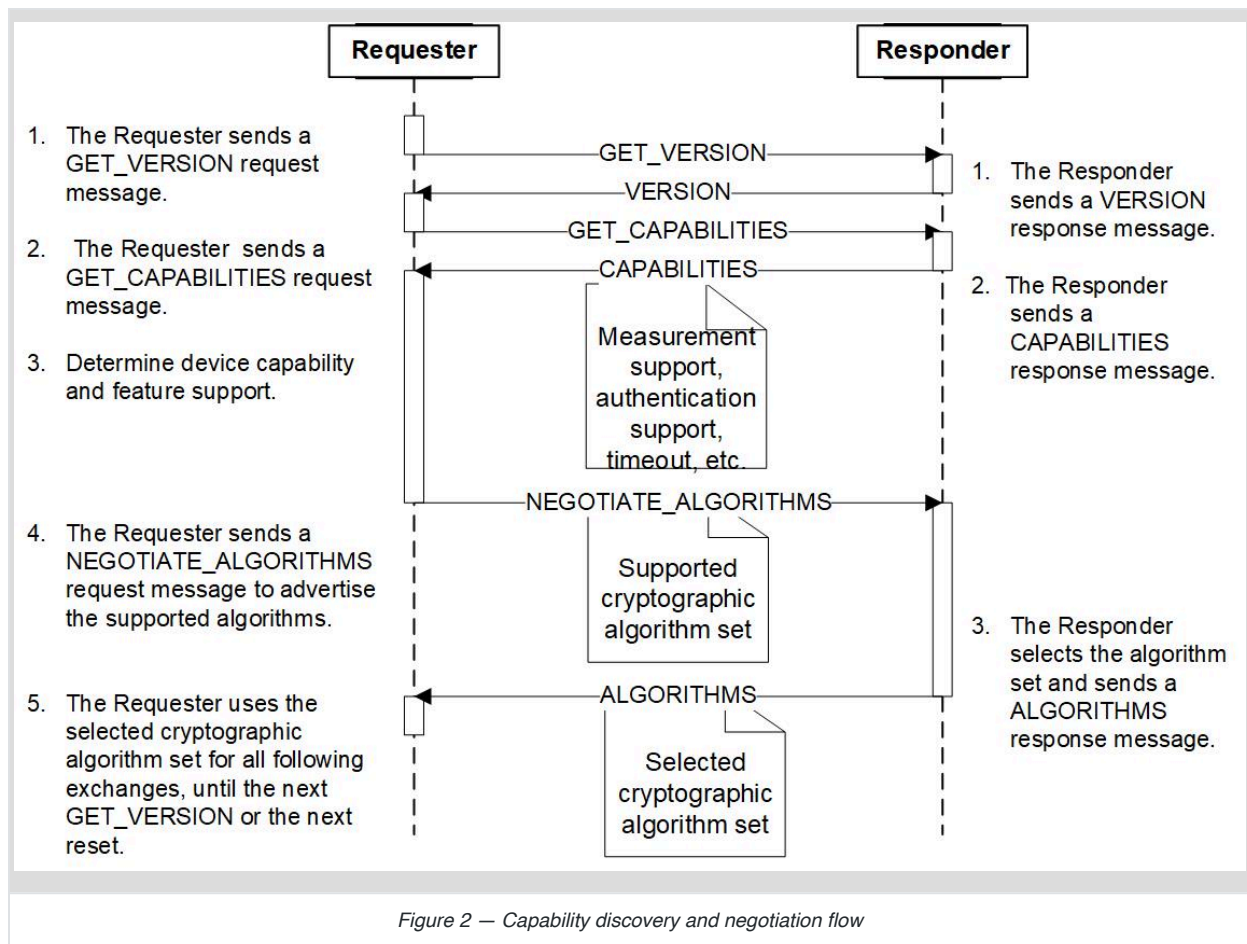
## 4.9 SPDM messages

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

1. Capability discovery and negotiation.
2. Hardware identity authentication.
3. Firmware measurement.

### 4.9.1 Capability discovery and negotiation

All Requesters and Responders shall support [GET\\_VERSION](#), [GET\\_CAPABILITIES](#) and [NEGOTIATE\\_ALGORITHMS](#). [Figure 2](#) shows the high-level request-response flow and sequence for the capability discovery and negotiation.



#### 4.9.1.1 GET\_VERSION request message and VERSION response message

This request message shall retrieve an endpoint's SPDM version. Table 7 shows the `GET_VERSION` request message format and Table 8 shows the `VERSION` response message format.

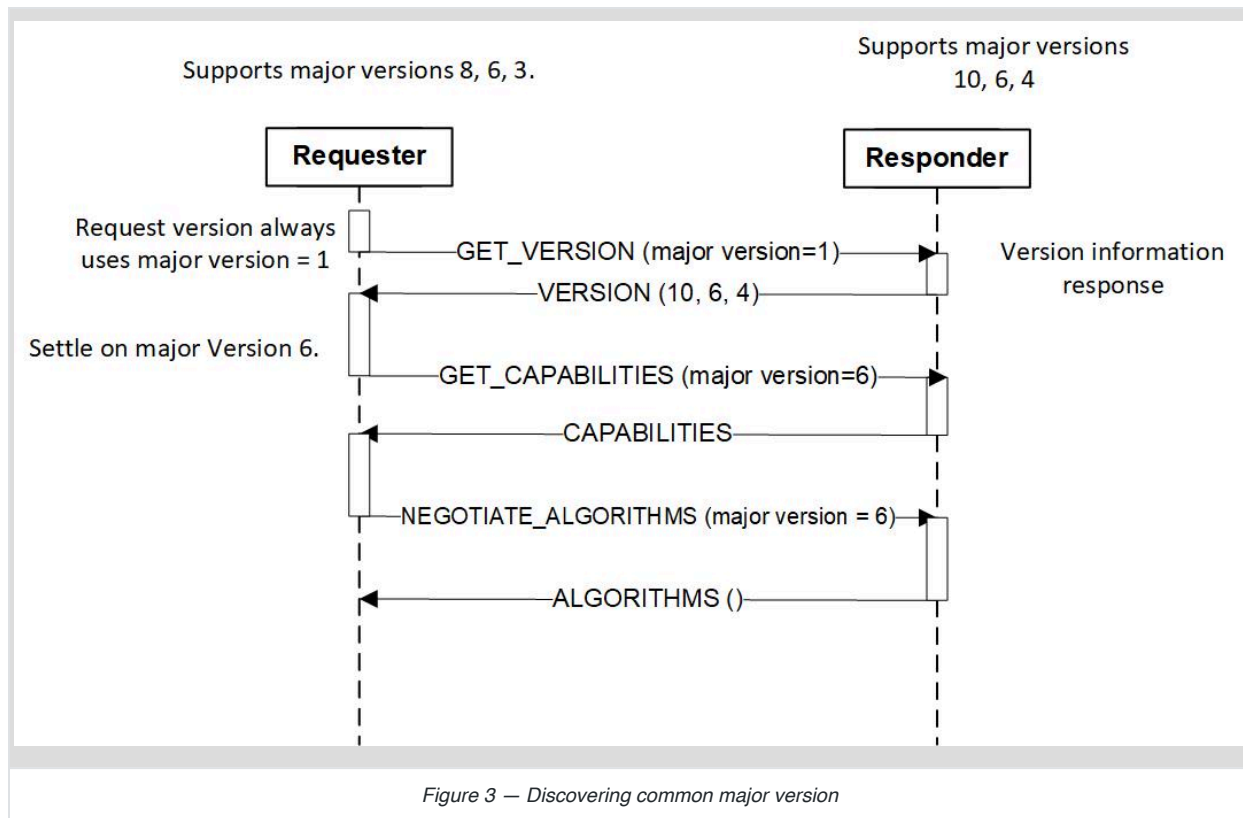
In all future SPDM versions, the `GET_VERSION` and `VERSION` response messages will be backward compatible with all previous versions.

The Requester shall begin the discovery process by sending a `GET_VERSION` request message with major version 0x1. All Responders must always support `GET_VERSION` request message with major version 0x1 and provide a `VERSION` response containing all supported versions as described in Table 7.

The Requester shall consult the `VERSION` response to select a common (typically highest) version supported. The Requester shall use the selected version in all future communication of other requests. A Requester shall not issue other requests until it has received a successful `VERSION` response and has identified a common version supported by both sides. A Responder shall not respond to `GET_VERSION` request message with `ErrorCode=ResponseNotReady`.



A Requester may issue `GET_VERSION` request message at any time to a Responder, which is as an exception to the rules in [Requirements for Requesters](#) for the case where a Requester must restart the protocol because of internal error or reset. After receiving a `GET_VERSION` request the Responder shall cancel all previous requests from the same Requester. Additionally, this message shall clear or reset the previously Negotiated State, if any, in both the Requester and its corresponding Responder.



**Table 7 — `GET_VERSION` request message**

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

**Table 8 — Successful `VERSION` response message**

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x04 = VERSION
2	Param1	1	Reserved
3	Param2	1	Reserved
4	Reserved	1	Reserved
5	VersionNumberEntryCount	1	Number of version entries present in this table (=n).
6	VersionNumberEntry1:n	2 x n	16-bit version entry. See <a href="#">Table 7</a> .

**Table 9 — VersionNumberEntry definition**

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

#### 4.9.1.2 GET\_CAPABILITIES request message and CAPABILITIES response message

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

[Table 10](#) shows the GET\_CAPABILITIES request message format.

[Table 11](#) shows the CAPABILITIES response message format.

[Table 12](#) shows the flag fields definitions.

A Responder shall not respond to GET\_CAPABILITIES request message with ErrorCode= ResponseNotReady .

**Table 10 — GET\_CAPABILITIES request message**

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0xE1 = GET_CAPABILITIES
2	Param1	1	Reserved
3	Param2	1	Reserved

Table 11 — Successful CAPABILITIES response message

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x61 = CAPABILITIES
2	Param1	1	Reserved
3	Param2	1	Reserved
4	Reserved	1	Reserved
5	CTExponent	1	The value of this shall be the exponent of base 2. Used to calculate CT, as described in Table 6. The equation for CT shall be $2^{CT}$ microseconds (us). For example, if CTExponent is 10, CT is $2^{10} = 1024$ us.
6	Reserved	2	Reserved
8	Flags	4	See Table 1.

Table 12 — Flag fields definitions

Byte	Bit	Field	Value
0	0	CACHE_CAP	If set, the Responder supports the ability to cache the Negotiated State across a reset. This allows the Requester to skip reissuing the GET_VERSION, GET_CAPABILITIES and NEGOTIATE_ALGORITHMS requests after a reset. The Responder shall cache the selected cryptographic algorithms as one of the parameters of the Negotiated State. If the Requester chooses to skip issuing these requests after the reset, the Requester shall also cache the same selected cryptographic algorithms.
0	1	CERT_CAP	If set, Responder supports GET_DIGESTS and GET_CERTIFICATE messages.
0	2	CHAL_CAP	If set, Responder supports CHALLENGE request message.

Byte	Bit	Field	Value
0	4:3	MEAS_CAP	<p>The Responder's MEASUREMENT capabilities.</p> <ul style="list-style-type: none"> <li>• 00b . The Responder does not support MEASUREMENTS capabilities.</li> <li>• 01b . The Responder supports MEASUREMENTS but cannot perform signature generation.</li> <li>• 10b . The Responder supports MEASUREMENTS and can generate signatures.</li> <li>• 11b . Reserved</li> </ul>
0	5	MEAS_FRESH_CAP	<ul style="list-style-type: none"> <li>• 0 . As part of MEASUREMENTS response message, the Responder may return MEASUREMENTS that were computed during the last Responder's reset.</li> <li>• 1 . The Responder can recompute all MEASUREMENTS in a manner that is transparent to the rest of the system and shall always return fresh MEASUREMENTS as part of MEASUREMENTS response message.</li> </ul>
0	7:6	Reserved	Reserved
1	7:0	Reserved	Reserved
2	7:0	Reserved	Reserved
3	7:0	Reserved	Reserved

#### 4.9.1.3 NEGOTIATE\_ALGORITHMS request message and ALGORITHMS response message

This request message shall negotiate cryptographic algorithms. A Requester shall not issue a NEGOTIATE\_ALGORITHMS request message until it receives a successful CAPABILITIES response message.

A Requester shall not issue any other SPDM requests, with the exception of GET\_VERSION until it receives a successful ALGORITHMS response message with exactly one asymmetric algorithm and exactly one hashing algorithm.

A Responder shall not respond to NEGOTIATE\_ALGORITHMS request message with ErrorCode=ResponseNotReady .

Table 13 shows the NEGOTIATE\_ALGORITHMS request message format.

Table 14 shows the ALGORITHMS response message format.

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

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0xE3 = NEGOTIATE_ALGORITHMS
2	Param1	1	Reserved

Offset	Field	Size (bytes)	Value
3	Param2	1	Reserved
4	Length	2	Length of the entire request message, in bytes. Length shall be less than 64 bytes.
6	MeasurementSpecification	1	This field is a bitmask. The values for this field shall be those defined in the MeasurementSpecification field of <a href="#">GET_MEASUREMENTS request message</a> and <a href="#">MEASUREMENTS response message</a> . The Requester may set more than one bit to indicate multiple measurement specification support.
7	Reserved	1	Reserved
8	BaseAsymAlgo	4	<p>Bit mask listing Requester-supported SPDM-enumerated asymmetric key signature algorithms for the purposes of signature verification.</p> <ul style="list-style-type: none"> <li>• Byte 0 Bit 0. <a href="#">TPM_ALG_RSASSA_2048</a></li> <li>• Byte 0 Bit 1. <a href="#">TPM_ALG_RSAPSS_2048</a></li> <li>• Byte 0 Bit 2. <a href="#">TPM_ALG_RSASSA_3072</a></li> <li>• Byte 0 Bit 3. <a href="#">TPM_ALG_RSAPSS_3072</a></li> <li>• Byte 0 Bit 4. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P256</a></li> <li>• Byte 0 Bit 5. <a href="#">TPM_ALG_RSASSA_4096</a></li> <li>• Byte 0 Bit 6. <a href="#">TPM_ALG_RSAPSS_4096</a></li> <li>• Byte 0 Bit 7. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P384</a></li> <li>• Byte 1 Bit 0. <a href="#">TPM_ALG_ECDSA_ECC_NIST_P521</a></li> </ul> <p>All other values reserved.</p>
12	BaseHashAlgo	4	<p>Bit mask listing Requester-supported SPDM-enumerated cryptographic hashing algorithms.</p> <ul style="list-style-type: none"> <li>• Byte 0 Bit 0. <a href="#">TPM_ALG_SHA_256</a></li> <li>• Byte 0 Bit 1. <a href="#">TPM_ALG_SHA_384</a></li> <li>• Byte 0 Bit 2. <a href="#">TPM_ALG_SHA_512</a></li> <li>• Byte 0 Bit 3. <a href="#">TPM_ALG_SHA3_256</a></li> <li>• Byte 0 Bit 4. <a href="#">TPM_ALG_SHA3_384</a></li> <li>• Byte 0 Bit 5. <a href="#">TPM_ALG_SHA3_512</a></li> </ul> <p>All other values reserved.</p>
16	Reserved	12	Reserved
28	ExtAsymCount	1	Number of Requester-supported extended asymmetric key signature algorithms (=A). A + E shall be less than or equal to 8.
29	ExtHashCount	1	Number of Requester-supported extended hashing algorithms (=E). A + E shall be less than or equal to 8.
30	Reserved	2	Reserved for future use
32	ExtAsym	4*A	List of Requester-supported extended asymmetric key signature algorithms. The format of this field is described in <a href="#">Extended Algorithm Field Format Table</a> .

Offset	Field	Size (bytes)	Value
32+4*A	<i>ExtHash</i>	4*E	List of the extended hashing algorithms supported by Requester. The format of this field is described in <a href="#">Extended Algorithm Field Format Table</a> .

**Table 14 — Successful ALGORITHMS response message**

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	<a href="#">RequestResponseCode</a>	1	0x63 = ALGORITHMS
2	Param1	1	Reserved
3	Param2	1	Reserved
4	Length	2	Length of the response message, in bytes.
6	MeasurementSpecificationSel	1	Bit mask. The Responder shall select one of the measurement specifications supported by the Requester. Thus, no more than one bit shall be set. The values in this field shall be those defined in the <code>MeasurementSpecification</code> field of <a href="#">Table 25</a> .
7	Reserved	1	Reserved
8	MeasurementHashAlgo	4	<p>Bit mask listing SPDM-enumerated hashing algorithm for measurements. M represents the length of the measurement hash field in measurement block structure (<a href="#">Table 20</a>). The Responder shall ensure the length of measurement hash field during all subsequent MEASUREMENT response messages to the Requester until the next <code>ALGORITHMS</code> response message is M.</p> <ul style="list-style-type: none"> <li>• Bit 0. Raw Bit Stream Only, M=0</li> <li>• Bit 1. <a href="#">TPM_ALG_SHA_256</a>, M=32</li> <li>• Bit 2. <a href="#">TPM_ALG_SHA_384</a>, M=48</li> <li>• Bit 3. <a href="#">TPM_ALG_SHA_512</a>, M=64</li> <li>• Bit 4. <a href="#">TPM_ALG_SHA3_256</a>, M=32</li> <li>• Bit 5. <a href="#">TPM_ALG_SHA3_384</a>, M=48</li> <li>• Bit 6. <a href="#">TPM_ALG_SHA3_512</a>, M=64</li> </ul> <p>If the Responder supports <code>GET_MEASUREMENTS</code>, exactly one bit in this bit field shall be set. Otherwise, the Responder shall set this field to 0.</p> <p>A Responder shall only select Bit 0 if the Responder supports Raw Bit Streams as the only form of measurement; otherwise, it shall select one of the other bits.</p>
12	BaseAsymSel	4	Bit mask listing the SPDM-enumerated asymmetric key signature algorithm selected. A Responder that returns <code>CHAL_CAP=0</code> and <code>MEAS_CAP != 2</code> shall set this field 0. Other Responders shall set no more than one bit.

Offset	Field	Size (bytes)	Value
16	BaseHashSel	4	Bit mask listing the SPDM-enumerated hashing algorithm selected. A Responder that returns <code>CHAL_CAP=0</code> and <code>MEAS_CAP != 2</code> shall set this field 0. Other Responders shall set no more than one bit.
20	Reserved	12	Reserved.
32	ExtAsymSelCount	1	The number of extended asymmetric key signature algorithms selected. Shall be either 0 or 1 (=A'). A Requester that returns <code>CHAL_CAP=0</code> and <code>MEAS_CAP != 2</code> shall set this field 0.
33	ExtHashSelCount	1	The number of extended hashing algorithms selected. Shall be either 0 or 1 (=E'). A Requester that returns <code>CHAL_CAP=0</code> and <code>MEAS_CAP != 2</code> shall set this field 0.
34	Reserved	2	Reserved
36	ExtAsymSel	4*A'	<p>The extended asymmetric key signature algorithm selected.</p> <p>Responder must be able to sign a response message using this algorithm and Requester must have listed this algorithm in the request message indicating it can verify a response message using this algorithm. The Responder shall use this asymmetric signature algorithm for all subsequent applicable response messages to the Requester.</p> <p>The format of this field is described in <a href="#">Extended Algorithm Field Format Table</a>.</p>
36+4*A	ExtHashSel	4*E'	<p>The extended Hashing algorithm selected.</p> <p>The Responder shall use this hashing algorithm during all subsequent response messages to the Requester. The Requester shall use this hashing algorithm during all subsequent applicable request messages to the Responder.</p> <p>The format of this field is described in <a href="#">Extended Algorithm Field Format Table</a>.</p>

**Extended Algorithm Field Format Table**

Offset	Field	Description
0	Registry ID	This field shall represent the registry or standards body. This field's value shall be one listed in the <b>ID</b> column of <a href="#">Table 29</a> .
1	Reserved	Reserved
[2:3]	Algorithm ID	This field shall indicate the desired algorithm. The value of this field is owned by the registry or standards body. For details, see <a href="#">Table 29</a>

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

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

This clause illustrates how two endpoints negotiate base hashing algorithm.

In [Figure 4](#), endpoint A issues `NEGOTIATE_ALGORITHMS` request message and endpoint B selects an algorithm of which both endpoints are capable.

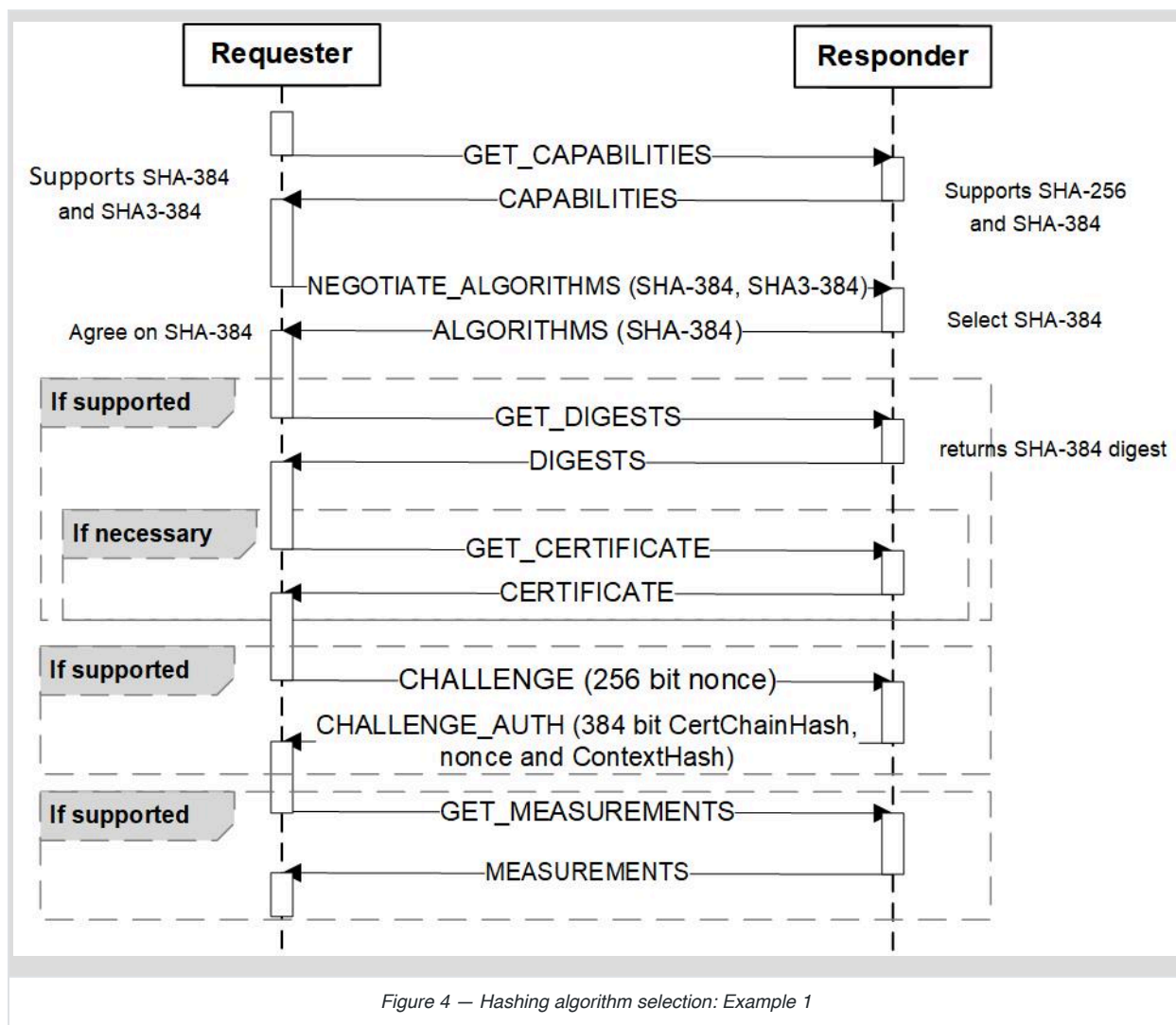


Figure 4 — Hashing algorithm selection: Example 1

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



### 4.9.2 Responder identity authentication

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

Figure 5 shows the high-level request-response message flow and sequence for Responder's identity authentication for *certificate* retrieval.

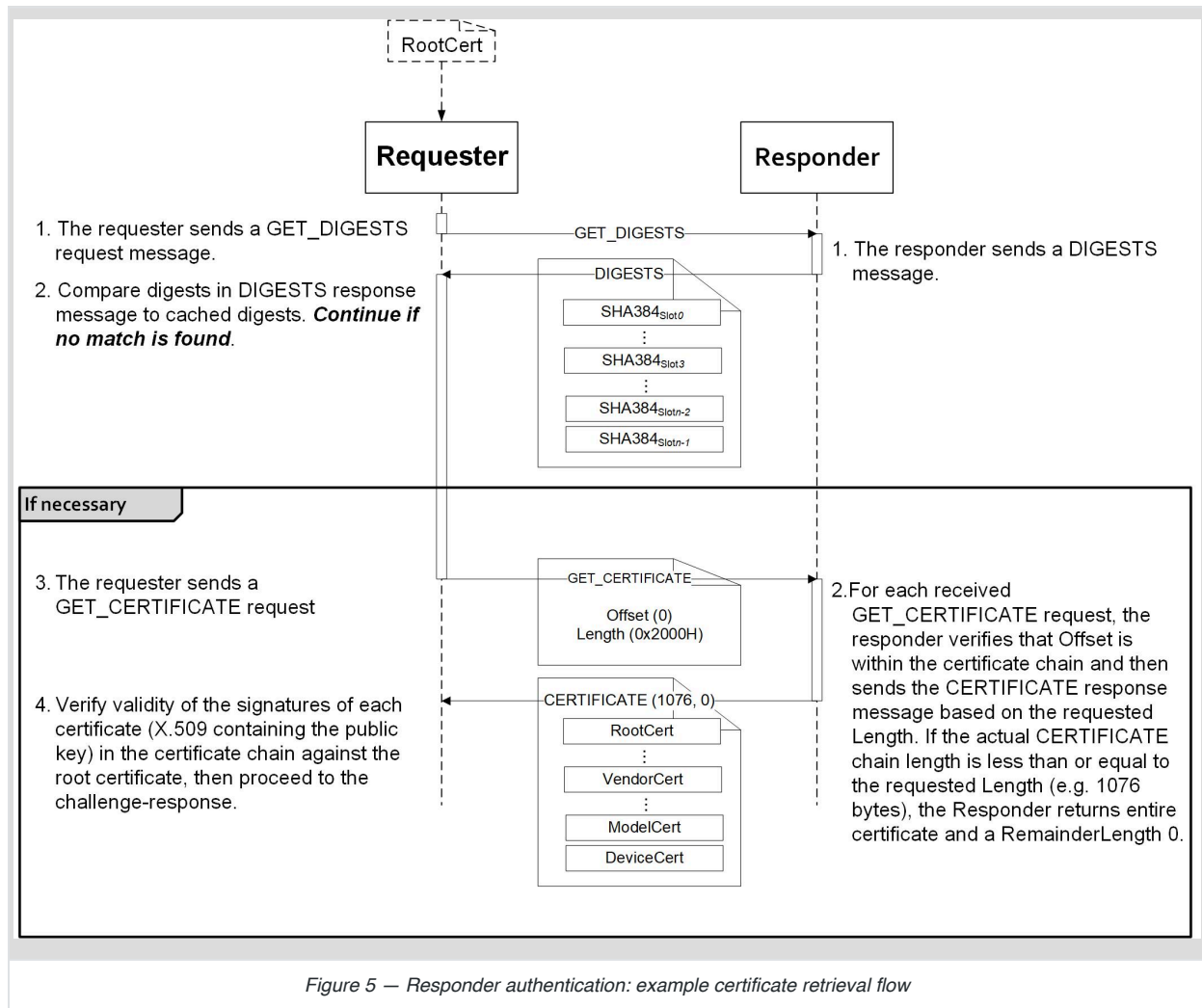


Figure 5 — Responder authentication: example certificate retrieval flow

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

detect any change to the certificate chains stored on the device before issuing the `GET_CERTIFICATE` request message.

For the runtime challenge-response flow, the signature field in the `CHALLENGE_AUTH` response message payload shall be signed using the device private key over the hash of the message transcript defined in [Table 103](#). This ensures cryptographic binding between a specific request message from a specific Requester and a specific response message from a specific Responder and allows the Requester to detect the presence of an active adversary attempting to downgrade cryptographic algorithms or SPDM Versions. Furthermore, a nonce generated by the Requester protects the challenge-response from replay attacks, whereas a nonce generated by the Responder prevents the Responder from signing over arbitrary data dictated by the Requester. The signature computation is restarted with the latest `GET_VERSION` request received.

#### 4.9.2.1 Certificates and certificate chains

Each Responder that supports identity authentication shall carry at least one certificate chain. A certificate chain contains an ordered list of certificates, presented as the binary (byte) concatenation of the fields shown in [Table 11](#). Each certificate shall be in ASN.1 DER-encoded X509v3 format. The ASN.1 DER encoding of each individual certificate can be analyzed to determine its length. The minimum number of certificates within a chain shall be one, in which case the single certificate is the device-specific certificate. The Responder shall contain a single public-private key pair per supported algorithm for its hardware identity, regardless of how many certificate chains are stored on the device. The Responder selects a single asymmetric key signature algorithm per Requester.

Certificate chains are stored in locations called slots. Each slot shall either be empty or contain one complete certificate chain. A Product shall not contain more than 8 slots. Slot 0 is populated by default. Additional slots may be populated through the supply chain such as by a platform integrator or by an end user such as the IT administrator. A slot mask is used to identify the certificate chains from the 8 slots.

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

**Table 15 — Certificate chain format**

Offset	Field	Size	Description
0	<code>Length</code>	2	Total length of the certificate chain, in bytes, including all fields in this table. This field is little endian.
2	<code>Reserved</code>	2	Reserved.
4	<code>RootHash</code>	H	Digest of the Root Certificate. Note that Root Certificate is ASN.1 DER-encoded for this digest. This field is big endian.
4 + H	<code>Certificates</code>	Length - (4 + H)	One or more ASN.1 DER-encoded X509v3 certificates where the first certificate is signed by the Root Certificate or is the Root Certificate itself and each subsequent certificate is signed by the preceding certificate. The last certificate is the <a href="#">Leaf Certificate</a> . This field is big endian.

#### 4.9.2.2 GET\_DIGESTS request message and DIGESTS response message

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

Table 16 shows the GET\_DIGESTS request message format.

Table 17 shows the DIGESTS response message format.

The digests in Table 13 are in big endian.

**Table 16 — GET\_DIGESTS request message**

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

**Table 17 — Successful DIGESTS response message**

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x01 = DIGESTS
2	Param1	1	Reserved
3	Param2	1	Slot mask. The bit in position K of this byte shall be set to 1b if and only if slot number K contains a certificate chain for the protocol version in the SPDMVersion field. (Bit 0 is the least significant bit of the byte.) The number of digests returned shall be equal to the number of bits set in this byte. The digests shall be returned in order of increasing slot number.
4	Digest[0]	H	Digest of the first certificate chain.
...	...	...	...
4 + (H * (n - 1))	Digest[n-1]	H	Digest of the last (n <sup>th</sup> ) certificate chain.

#### 4.9.2.3 GET\_CERTIFICATE request message and CERTIFICATE response message

This request message shall retrieve the certificate chains.

Table 18 shows the GET\_CERTIFICATE request message format.

Table 19 shows the CERTIFICATE response message format.

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

**Table 18 — GET\_CERTIFICATE request message**

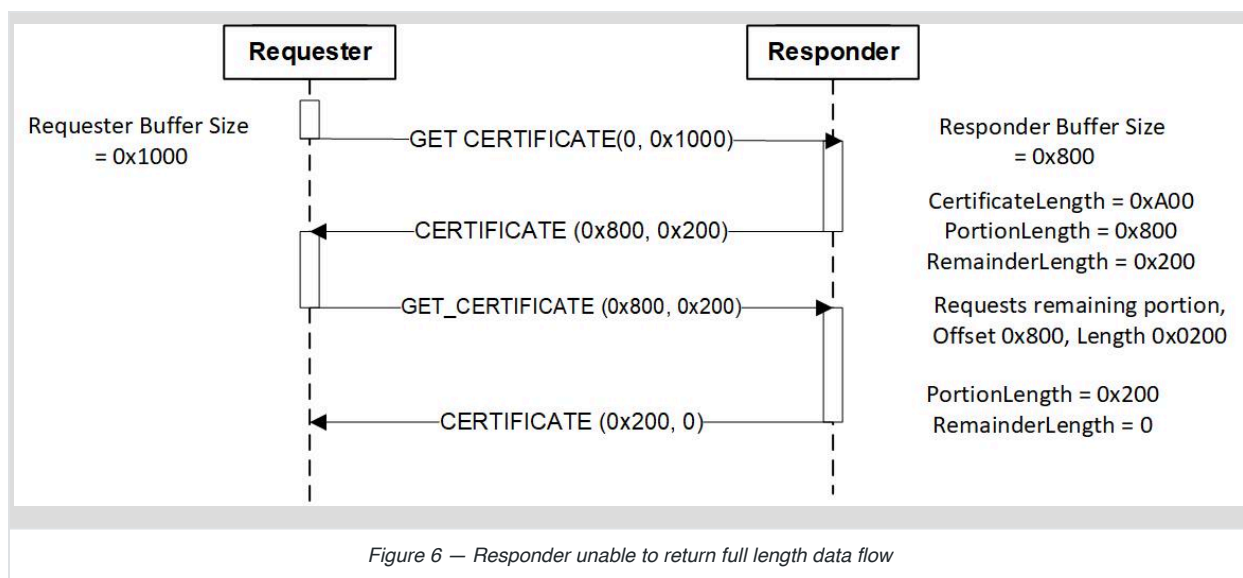
Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x82 = GET_CERTIFICATE
2	Param1	1	Slot number of the target certificate chain to read from. The value in this field shall be between 0 and 7 inclusive.
3	Param2	1	Reserved
4	Offset	2	Offset in bytes from the start of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

**Table 19 — Successful CERTIFICATE response message**

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

Offset	Field	Size (bytes)	Value
3	Param2	1	Reserved.
4	PortionLength	2	Number of bytes of this portion of certificate chain. This should be less than or equal to <code>Length</code> received as part of the request. For example, the Responder might set this field to a value less than <code>Length</code> received as part of the request due limitations on the Responder's internal buffer.
6	RemainderLength	2	Number of bytes of the certificate chain that have not been sent yet after the current response. For the last response, this field shall be 0 as an indication to the Requester that the entire certificate chain has been sent.
8	CertChain	PortionLength	Requested contents of target certificate chain, formatted in DER. This field is big endian.

Figure 6-1 shows the high-level request-response message flow for Responder response when it cannot return the entire data requested by the Requester in the first response.



#### 4.9.2.4 Leaf certificate

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

## 4.9.2.4.1 Required fields

Field	Description
Version	The version of the encoded certificate shall be present and shall be 3, or value 2.
Serial Number	The CA assigned serial number shall be present with a positive integer value.
Signature Algorithm	The signature algorithm used by CA shall be present.
Issuer	The CA distinguished name shall be specified.
Subject Name	The subject name shall be present and shall represent the distinguished name associated with the leaf certificate.
Validity	The certificate may include this attribute. If the validity attribute is present, the value for <i>notBefore</i> field should be assigned the generalized 19700101000000Z time value and <i>notAfter</i> field should be assigned the generalized 99991231235959Z time value.
Subject Public Key Info	The device public key and the algorithm shall be present.
Extended Key Usage	The Extended Key Usage field shall be present and key usage bit for digital signature shall be set.

## 4.9.2.4.2 Optional fields

Field	Description
Basic Constraints	If present, the CA value shall be FALSE.
Subject Alternative Name otherName	In some cases it might be desirable to provide device specific information as part of the device certificate. DMTF chose the otherName field to be used with a specific format to represent the device information. The use of othername field also provides flexibility for other alliances to be able to provide device specific information as part of the device certificate.

## 4.9.2.4.3 Definition of othername using the DMTF OID

```
DMTFOtherName ::= SEQUENCE {
    type-id  [0] id-DMTF-device-info
    value    [1] ub-DMTF-device-info
}
-- OID for DMTF device info --
```

```

id-DMTF-device-info  OBJECT IDENTIFIER ::= { 1 3 6 1 4 1 412 274 1 }

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

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

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

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

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

```

Annex B shows an example leaf certificate.

#### 4.9.2.5 CHALLENGE request message and CHALLENGE\_AUTH response message

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

Table 20 shows the CHALLENGE request message format.

Table 21 shows the CHALLENGE\_AUTH response message format.

**Table 20 — CHALLENGE request message**

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

Offset	Field	Size (bytes)	Value
3	Param2	1	<p>Requested Measurement Summary Hash Type:</p> <p>0x0 = No Measurement Summary Hash,</p> <p>0x1 = TCB Component Measurement Hash,</p> <p>0xFF = All measurements Hash.</p> <p>All other values reserved.</p> <p>When Responder does not support any measurements, Requester shall set this value to 0x0 .</p>
4	Nonce	32	The Requester should choose a random value.

Table 21 — Successful CHALLENGE\_AUTH response message

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x03 = CHALLENGE_AUTH
2	Param1	1	Shall contain the Slot number in the Param1 field of the corresponding CHALLENGE request. This value can be used, by the Requester, to check that the certificate matched what was requested.
3	Param2	1	Slot mask. The bit in position K of this byte shall be set to 1b if and only if slot number K contains a certificate chain for the protocol version in the SPDMVersion field. (Bit 0 is the least significant bit of the byte.)
4	CertChainHash	H	Hash of the certificate chain used for authentication. This field is big endian. This value can be used, by the Requester, to check that the certificate matched what was requested.
4 + H	Nonce	32	Responder-selected random value.



Offset	Field	Size (bytes)	Value
36 + H	MeasurementSummaryHash	H	<p>When the Responder does not support measurement or requested <code>param2</code> = 0, the field shall be absent.</p> <p>When the requested <code>param2</code> = 1, this field shall be the combined hash of all measurements of all measurable components considered to be in the TCB required to generate this response.</p> <p>When the requested <code>param2</code> = 1 and there are no measurable components in the TCB required to generate this response, this field shall be 0.</p> <p>When requested <code>param2</code> = 0xFF, this field is computed as the hash(Concatenation(Measurement 1, Measurement 2, ..., Measurement N)) of all supported measurements.</p>
36 + 2H	OpaqueLength	2	Size of the <code>OpaqueData</code> field. The value shall not be greater than 1024 bytes.
38 + 2H	OpaqueData	OpaqueLength	Free-form field, if present. The Responder may include Responder-specific information and/or information defined by its transport.
38 + 2H + OpaqueLength	Signature	S	<p>S is the size of the asymmetric signing algorithm output the Responder selected via the last <code>ALGORITHMS</code> response message to the Requester. Signature generation and verification processes are defined in the <a href="#">CHALLENGE_AUTH Signature generation</a> and <a href="#">CHALLENGE_AUTH Signature verification</a> clauses, respectively.</p>

#### 4.9.2.6 CHALLENGE\_AUTH Signature generation

1. The Responder shall construct M1 and the Requester shall construct M2, message transcripts, as defined in [Table 103](#)

where:

- `Concatenate( )` is the standard concatenation function.
- `Concatenate( )` operation is performed only after a successful completion response on the entire contents of the request and the response.
- If a response contains `ErrorCode`= `ResponseNotReady`, the `Concatenate( )` operation is performed on contents of the original request and the contents of the response received during `RESPOND_IF_READY`.
- If a response contains `ErrorCode`  $\neq$  `ResponseNotReady`, no concatenate operation is performed on the original request and the response.

2. The Responder shall generate:

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

where:

- `Sign` is the asymmetric signing algorithm the Responder selected via the last `ALGORITHMS` response message sent by the Responder. See `BaseAsymSel` or `ExtAsymSel` fields in [Table 11](#).
- `Hash` is the hashing algorithm the Responder selected via the last `ALGORITHMS` response message sent by the Responder. See `BaseHashSel` or `ExtHashSel` fields in [Table 11](#).
- `SK` is the private Key associated with the Responder's leaf certificate in slot=Param1 of `CHALLENGE` request message.

#### 4.9.2.7 CHALLENGE\_AUTH Signature verification

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

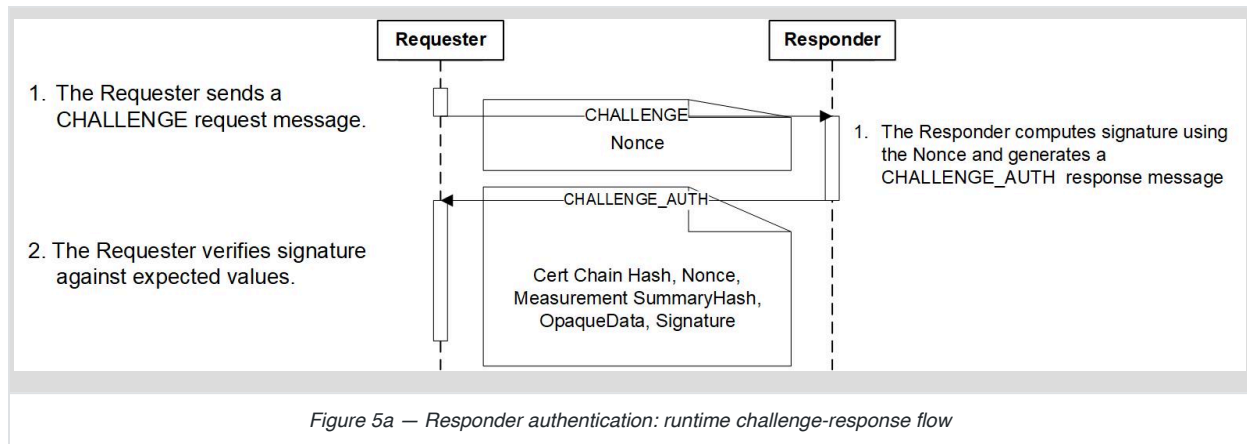
1. The Requester shall perform:

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

where:

- `PK` is the public key associated with the leaf certificate of the Responder with `slot=Param1` of the `CHALLENGE` request message.
- `Verify` is the asymmetric verification algorithm the Responder selected through the last `ALGORITHMS` response message as received by the Requester. See the `BaseAsymSel` or `ExtAsymSel` field in [Table 11](#).
- `Hash` is the hashing algorithm the Responder selected via the last `ALGORITHMS` response message sent as received by the Requester. See the `BaseHashSel` or `ExtHashSel` field in [Table 11](#).

[Figure 5a](#) shows the high-level request-response message flow and sequence for Responder's authentication for runtime challenge-response.



## 4.10 Request ordering and message transcript computation rules for M1 and M2

Table 103 defines how the message transcript is constructed for M1 and M2 which are used in signature calculation and verification in the CHALLENGE\_AUTH response message. The possible request orderings after Power on Reset are listed below explicitly:

- GET\_VERSION, GET\_CAPABILITY, NEGOTIATE\_ALGORITHMS, GET\_DIGESTS, GET\_CERTIFICATE, CHALLENGE
- GET\_VERSION, GET\_CAPABILITY, NEGOTIATE\_ALGORITHMS, GET\_DIGESTS, CHALLENGE
- GET\_VERSION, GET\_CAPABILITY, NEGOTIATE\_ALGORITHMS, CHALLENGE
- GET\_DIGESTS, GET\_CERTIFICATE, CHALLENGE
- GET\_DIGESTS, CHALLENGE
- GET\_DIGESTS
- CHALLENGE

After a successful CHALLENGE\_AUTH response is received by the Requestor, or the Requestor sends a GET\_MEASUREMENTS request, M1 and M2 shall be set to null. Immediately after Power on Reset M1 and M2 shall be null. If a Requestor sends a GET\_VERSION message the Requestor and Responder shall reset M1 and M2 to null and recommence construction of M1 and M2 starting with the new GET\_VERSION message.

**Table 103 — Request ordering and message transcript computation rules for M1/M2**

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

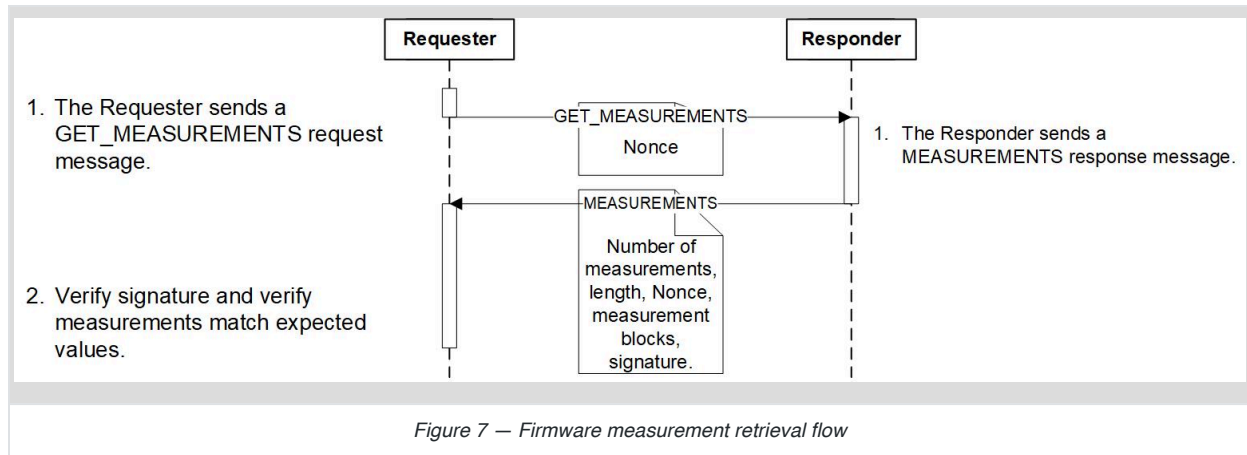
Requests	Implementation Requirements	M1/M2 = Concatenate (A, B, C)
GET_VERSION issue	The Requester may choose to issue this request any time, to allow Requester / Responder to determine an agreed upon Negotiated State. A Requester may detect out of synch condition typically when signature verification fails or when the Responder provides an unexpected error response.	M1/M2 = null
GET_VERSION , GET_CAPABILITIES , NEGOTIATE_ALGORITHMS	Requester shall always issue these requests in the order shown.	A = Concatenate (GET_VERSION, VERSION, GET_CAPABILITIES, CAPABILITIES, NEGOTIATE_ALGORITHMS, ALGORITHMS)
GET_VERSION , GET_CAPABILITIES , NEGOTIATE_ALGORITHMS	Requester may skip issuing these requests after a new Power on Reset, if the Responder has previously indicated CACHE_CAP = 1. In this case the Requester and Responder shall proceed with the previously Negotiated State	A = null
GET_DIGESTS , GET_CERTIFICATE	Requester shall always issue these requests in the order shown after NEGOTIATE_ALGORITHMS request completion or immediately after Power on Reset, if it chose to skip the previous three requests.	B = Concatenate (GET_DIGEST, DIGEST, GET_CERTIFICATE, CERTIFICATE)
GET_DIGESTS , GET_CERTIFICATE	Requester may choose to skip both requests after a new Power on Reset if it is capable of using previously cached response to these requests.	B = Null
GET_DIGESTS , GET_CERTIFICATE	Requester may choose to skip GET_CERTIFICATE request after a new Power on Reset if it is capable of using previously cached CERTIFICATE response.	B = (GET DIGESTS, DIGEST)
CHALLENGE	Requester shall issue this request to complete security verification of current requests and responses. The Signature bytes of CHALLENGE_AUTH shall not be included in C.	C = (CHALLENGE, CHALLENGE_AUTH\Signature).
CHALLENGE completion	Completion of CHALLENGE resets M1 and M2	M1/M2 = null
CHALLENGE	Requester may choose to skip this request and forgo security verification of previous requests and responses. Requester may typically skip CHALLENGE when it issues GET_DIGESTS directly after Power on Reset.	NA
GET_MEASUREMENTS	If the Requester chooses to issue GET_MEASUREMENTS and skips CHALLENGE completion, M1 and M2 are reset to null	M1/M2 = null

#### 4.10.1 Firmware and other measurements

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

Figure 7 shows the high-level request-response flow and sequence for endpoint measurement. If MEAS\_FRESH\_CAP bit in the CAPABILITIES response message returns 0, and the Requester requires fresh measurements, the

Responder must be reset before `GET_MEASUREMENTS` is resent. The mechanisms employed for resetting the Responder are outside the scope of this specification.



#### 4.10.1.1 `GET_MEASUREMENTS` request message and `MEASUREMENTS` response message

This request message shall retrieve firmware measurements. A Requester should not send this message until it has received at least one successful `CHALLENGE_AUTH` response message from the responder. The successful `CHALLENGE_AUTH` response may have been received before the last Power on Reset.

Table 22 shows the `GET_MEASUREMENTS` request message format.

Table 23 shows the `GET_MEASUREMENTS` request message attributes.

Table 24 shows the `MEASUREMENTS` response message format.

**Table 22 — `GET_MEASUREMENTS` request message**

Offset	Field	Size (bytes)	Value
0	<code>SPDMVersion</code>	1	<code>V1.0 = 0x10</code>
1	<code>RequestResponseCode</code>	1	<code>0xE0 = GET_MEASUREMENTS</code>
2	<code>Param1</code>	1	Request attributes. See Table 23.
3	<code>Param2</code>	1	Measurement operation.  A value of 0x0 shall query the Responder for the total number of measurements available. A value of 0xFF shall request all measurements. A value between 0x1 and 0xFE inclusively shall request the measurement at the index corresponding to that value.

Offset	Field	Size (bytes)	Value
4	Nonce	32	The Requester should choose a random value. This field is only present if a signature is required on the response. See <a href="#">Table 23</a> .

Table 23 — GET\_MEASUREMENTS request attributes

Bit(s)	Value	Description
0	1	If the Responder can generate a signature as indicated in <a href="#">CAPABILITIES</a> message, this bit's value shall indicate to the Responder to generate a signature. The Responder shall generate a signature in the corresponding response. The <code>Nonce</code> field shall be present in the request.
0	0	This bit's value shall be used for Responders incapable of generating a signature as indicated in <a href="#">CAPABILITIES</a> message. For Responders capable of signature generation, this bit's value shall indicate the Requester does not want a signature. The Responder shall not generate a signature in the response. The <code>Nonce</code> field shall be absent in the request.
[7:1]	Reserved	Reserved

Table 24 — Successful MEASUREMENTS response message

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	<a href="#">RequestResponseCode</a>	1	0x60 = MEASUREMENTS
2	Param1	1	When Param2 in the requested measurement operation is 0, this parameter shall return the total number of measurement indices on the device. Otherwise, this field is reserved.
3	Param2	1	Reserved
4	NumberOfBlocks	1	Number of measurement blocks (N) in <b>MeasurementRecord</b> . This field shall reflect the number of measurement blocks in <b>MeasurementRecord</b> . If Param2 in the requested measurement operation is 0, this field shall be 0.
5	MeasurementRecordLength	3	Size of the <code>MeasurementRecord</code> field in bytes. If Param2 in the requested measurement operation is 0, this field shall be 0.
8	MeasurementRecord	L= MeasurementRecordLength	Concatenation of all Measurement Blocks that correspond to the requested Measurement operation. The Measurement Block structure is defined in <a href="#">Measurement block</a> .
8 + L	Nonce	32	The Responder should choose a random value.

Offset	Field	Size (bytes)	Value
40 + L	OpaqueLength	2	Size of the <code>OpaqueData</code> field in bytes. The value shall not be greater than 1024 bytes.
42 + L	OpaqueData	OpaqueLength	Free-form field, if present. The Responder may include Responder-specific information and/or information defined by its transport.
42 + L + OpaqueLength	Signature	S	Signature of the <code>GET_MEASUREMENTS</code> Request and <code>MEASUREMENTS</code> Response messages, excluding the Signature field and signed using the device private key (slot 0 leaf certificate private key). The Responder shall use the asymmetric signing algorithm it selected during the last <code>ALGORITHMS</code> response message to the Requester and S is the size of that asymmetric signing algorithm output.

#### 4.10.1.2 Measurement block

Each Measurement block defined in the `MEASUREMENTS` response message shall contain a four-byte descriptor (offsets 0-3), followed by the Measurement Data corresponding to a particular *Measurement Index* and *Measurement Type*. The blocks are ordered by `Index`.

The following table shows the format for a measurement block:

**Table 25 — Measurement block format**

Offset	Field	Size (bytes)	Value
0	Index	1	Index. This field shall represent the index of the measurement.
1	MeasurementSpecification	1	This field is a bitmask. The value shall indicate the measurement specification that the requested <code>Measurement</code> follows and shall match the selected measurement specification in <code>Algorithms</code> message (See <a href="#">Table 14</a> ). Only one bit shall be set in the Measurement Block. <ul style="list-style-type: none"> <li>Bit 0 = DMTF, as specified in (See <a href="#">Table 25a</a>)</li> </ul> All other bits are reserved.
2	MeasurementSize	2	Size of <code>Measurement</code> , in bytes.
4	Measurement	MeasurementSize	For format of this field is defined by <code>MeasurementSpecification</code>

#### 4.10.1.3 DMTF Specification for the Measurement field of a Measurement block

The present clause is the specification for the format of the `Measurement` field in a Measurement block when the `MeasurementSpecification` field selects Bit 0 = DMTF. This format is specified in [Table 25a](#).

**Table 25a — Format of the Measurement field in a Measurement block when the MeasurementSpecification field selects Bit 0 = DMTF**

Offset	Field	Size (bytes)	Value
0	DMTFSpecMeasurementValueType	1	<p>This field is composed of two parts: bit [7] indicating the representation in DMTFSpecMeasurementValue, and bits [6:0] indicating what is being measured by DMTFSpecMeasurementValue. These values are set independently. These values are interpreted as follows:</p> <ul style="list-style-type: none"> <li>• [7] = 0b: Hash</li> <li>• [7] = 1b: Raw Bit Stream</li> <li>• [6:0] = 00h: immutable ROM</li> <li>• [6:0] = 01h: mutable firmware</li> <li>• [6:0] = 02h: hardware configuration, such as straps, debug modes</li> <li>• [6:0] = 03h: firmware configuration, e.g., configurable firmware policy</li> </ul> <p>All other values reserved.</p>
1	DMTFSpecMeasurementValueSize	2	<p>Size of DMTFSpecMeasurementValue, in bytes. When DMTFSpecMeasurementValueType[7] = 0b: Hash, the DMTFSpecMeasurementValueSize shall be derived from the measurement hash algorithm returned in the ALGORITHM response message.</p>
3	DMTFSpecMeasurementValue	DMTFSpecMeasurementValueSize	<p>DMTFSpecMeasurementValueSize bytes of cryptographic hash or Raw Bit Stream, as indicated in DMTFSpecMeasurementType[7].</p>

**4.10.1.4 MEASUREMENTS Signature generation**

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

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

where:

- Concatenate ( ) is the standard concatenation function.
- GET\_MEASUREMENTS\_REQUEST1 is the entire first GET\_MEASUREMENTS request message under



consideration, where the Requester has not requested a signature on that specific `GET_MEASUREMENTS` request.

- `MEASUREMENTS_RESPONSE1` is the entire `MEASUREMENTS` response message without the signature bytes, as sent by the Responder in response to `GET_MEASUREMENT_REQUEST1`.
- `GET_MEASUREMENTS_REQUESTn-1` is the entire last consecutive `GET_MEASUREMENTS` request message under consideration, where the Requester has not requested a signature on that specific `GET_MEASUREMENTS` request.
- `MEASUREMENTS_RESPONSEn-1` is the entire `MEASUREMENTS` response message without the signature bytes, as sent by the Responder in response to `GET_MEASUREMENT_REQUESTn-1`.
- `GET_MEASUREMENTS_REQUESTn` is the entire first `GET_MEASUREMENTS` request message under consideration, where the Requester has requested a signature on that specific `GET_MEASUREMENTS` request. (n is a number greater than equal to 1. When n equals 1, the Requester has not made any `GET_MEASUREMENTS` request without signature prior to issuing `GET_MEASUREMENTS` request with signature.)
- `MEASUREMENTS_RESPONSEn` is the entire `MEASUREMENTS` response message without the signature bytes, as sent by the Responder in response to `GET_MEASUREMENT_REQUESTn`.
- Any communication between Requester and Responder other than `GET_MEASUREMENTS` request or response resets L1/L2 computation to null.

2. The Responder shall generate:

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

where:

- `Sign` is the asymmetric signing algorithm the Responder selected through the last `ALGORITHMS` response message sent by the Responder. See `BaseAsymSel` or `ExtAsymSel` fields in [Table 11](#).
- `Hash` is the hashing algorithm the Responder selected through the last `ALGORITHMS` response message sent by the Responder. See `BaseHashSel` or `ExtHashSel` fields in [Table 11](#).
- `SK` is the private Key associated with the Responder's slot 0 leaf certificate.

#### 4.10.1.5 MEASUREMENTS Signature verification

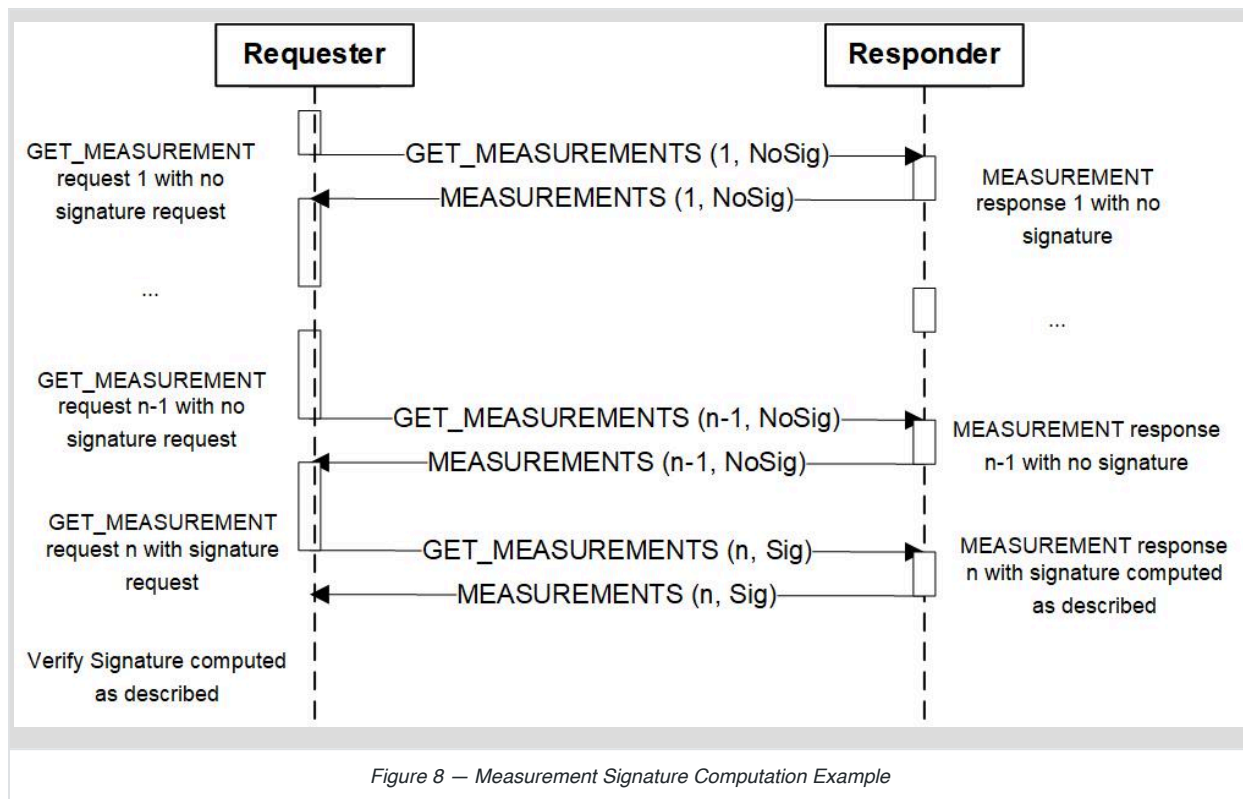
1. The Requester shall perform:

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

where:

- **PK** is the public key associated with the slot 0 certificate of the Responder. **PK** is extracted from the **CERTIFICATES** response.
- **Verify** is the asymmetric verification algorithm the Responder selected through the last **ALGORITHMS** response message as received by the Requester. See **BaseAsymSel** or **ExtAsymSel** fields in [Table 11](#).
- **Hash** is the hashing algorithm the Responder selected through the last **ALGORITHMS** response message sent as received by the Requester. See the **BaseHashSel** or **ExtHashSel** fields in [Table 11](#).

[Figure 8](#) shows an example of a typical Requester Responder protocol where the Requester issues 0 to n-1 GET\_MEASUREMENT requests without signature followed by a single GET\_MEASUREMENT request n with signature.



### 4.10.2 ERROR response message

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

Table 26 shows the `ERROR` response format.

Table 27 shows the detailed error code, error data, and extended error data.

Table 28 shows the `ResponseNotReady` extended error data.

Table 29 shows the registry or standards body ID.

Table 30 shows the `ExtendedErrorData` format definition for vendor or other standards-defined `ERROR` response message.

**Table 26 — ERROR response message**

Offset	Field	Size (bytes)	Value
0	<code>SPDMVersion</code>	1	V1.0 = 0x10
1	<code>RequestResponseCode</code>	1	0x7F = <code>ERROR</code>
2	<code>Param1</code>	1	Error Code. See Table 27.
3	<code>Param2</code>	1	Error Data. See Table 27.
4	<code>ExtendedErrorData</code>	0-32	Optional extended data. See Table 27.

**Table 27 — Error code and error data**

Error code	Value	Description	Error data	ExtendedErrorData
Reserved	00h	Reserved	Reserved	Reserved
<code>InvalidRequest</code>	01h	One or more request fields are invalid	0x00	No extended error data is provided.
Reserved	02h	Reserved	Reserved	Reserved
<code>Busy</code>	03h	The Responder received the request message and the Responder decided to ignore the request message, but the Responder may be able to process the request message if the request message is sent again in the future.	0x00	No extended error data is provided.

Error code	Value	Description	Error data	ExtendedErrorData
UnexpectedRequest	04h	The Responder received an unexpected request message. For example, CHALLENGE before NEGOTIATE_ALGORITHMS .	0x00	No extended error data is provided.
Unspecified	05h	Unspecified error occurred.	00h	No extended error data is provided.
Reserved	06h	Reserved	00h	Reserved
UnsupportedRequest	07h	The RequestResponseCode in the Request message is unsupported.	RequestResponseCode in the Request message.	No extended error data is provided
Reserved	08h - 40h	Reserved	Reserved	Reserved
MajorVersionMismatch	41h	Requested SPDM Major Version is not supported.	00h	No extended error data provided.
ResponseNotReady	42h	See RESPOND_IF_READY clause.	00h	See Table 28.
RequestResynch	43h	Responder is requesting Requester to reissue GET_VERSION in order to resynch.	0x00	No extended error data provided.
Reserved	44h - FEh	Reserved	Reserved.	Reserved
Vendor/Other Standards Defined	FFh	Vendor or Other Standards defined	This field shall indicate the registry or standard body using one of the values in the ID column of Table 29.	See Table 30 for format definition.

Table 28 — ResponseNotReady extended error data

Offset	Field	Size (bytes)	Value
0	RDTExponent	1	Exponent expressed in logarithmic (base 2 scale) to calculate RDT time in uS after which the Responder will be able to provide successful completion response. For example, the raw value 8 indicates that the Responder will be ready in $2^8 = 256$ uS. Responder should use RDT to avoid continuous ping and issue RESPOND_IF_READY after RDT time. For timing requirement details, see Table 6.
1	RequestCode	1	The request code that triggered this response.
2	Token	1	The opaque handle that the Requester shall pass in with the RESPOND_IF_READY request message.

Offset	Field	Size (bytes)	Value
3	RDTM	1	Multiplier used to compute $WT_{Max}$ in uS to indicate the response may be dropped after this delay. The multiplier shall always be greater than 1. The Responder may also stop processing the initial request if the same Requester issues a different request. For timing requirement details, see <a href="#">Table 6</a> .

**Table 29 — Registry or standards body ID**

Unless otherwise specified, for algorithm encoding used in extended algorithm fields, consult the respective registry or standards body.

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

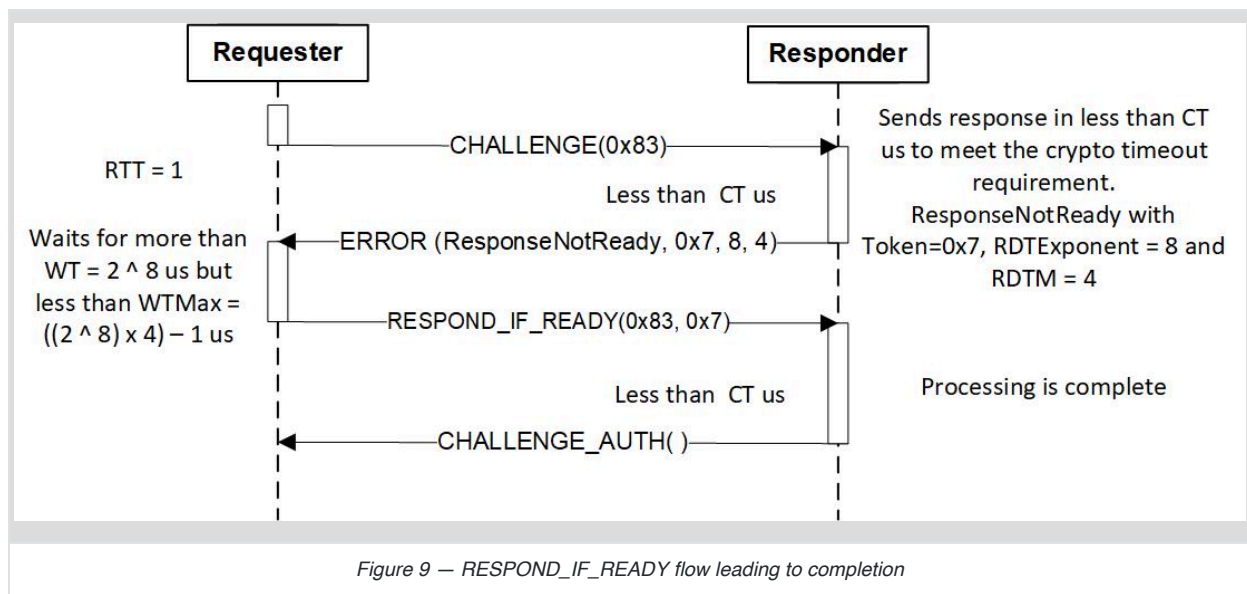
**Table 30 — ExtendedErrorData format definition for vendor or other standards-defined ERROR response message**

Byte Offset	Len	Field name	Description
0	1	Len	Length of the <code>VendorID</code> field. If the <code>ERROR</code> is vendor defined, the value of this field shall equal the <code>Vendor ID Len</code> as described in <a href="#">Table 29</a> of the corresponding registry or standard body name. If the <code>ERROR</code> is defined by a registry or a standard, this field shall be zero, which also indicates the <code>VendorID</code> field is not present. The registry or standards body name in the <code>ERROR</code> is indicated in the <code>Error Data</code> field, such as <code>Param2</code> , and is one of the values in the <code>ID</code> column of <a href="#">Table 29</a> .

Byte Offset	Len	Field name	Description
1	Len	VendorID	The value of this field shall indicate the Vendor ID, as assigned by the registry or standards body. The length of this field is provided in <a href="#">Table 29</a> . This field shall be in little endian format. The registry or standards body name in the <code>ERROR</code> is indicated in the <code>Error Data</code> field, such as <code>Param2</code> , and is one of the values in the <code>ID</code> column of <a href="#">Table 29</a> .
1 + Len	Variable	OpaqueErrorData	Defined by the vendor or other standards.

### 4.10.3 RESPOND\_IF\_READY request message

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



[Table 30](#) shows the `RESPOND_IF_READY` request message format.

**Table 31 — `RESPOND_IF_READY` request message**

Offset	Field	Size (bytes)	Value
0	<code>SPDMVersion</code>	1	<code>V1.0</code> = <code>0x10</code>

Offset	Field	Size (bytes)	Value
1	<a href="#">RequestResponseCode</a>	1	0xFF = RESPOND_IF_READY
2	<a href="#">RequestCode</a>	1	The original request code that triggered the <a href="#">ResponseNotReady</a> error code response. Shall match the request code returned as part of the <a href="#">ResponseNotReady</a> extended error data.
3	<a href="#">Token</a>	1	The token that was returned as part of the <a href="#">ResponseNotReady</a> extended error data.

#### 4.10.4 VENDOR\_DEFINED\_REQUEST request message

This request message can be used by a Requester intending to define a unique request to meet its need. The format is defined in [Table 104](#). The Requester should send this request message only after sending [GET\\_VERSION](#), [GET\\_CAPABILITIES](#) and [NEGOTIATE\\_ALGORITHMS](#) request sequence.

[Table 104](#) shows the [VENDOR\\_DEFINED\\_REQUEST](#) request message format.

**Table 104 — VENDOR\_DEFINED\_REQUEST request message**

Offset	Field	Size (bytes)	Value
0	<a href="#">SPDMVersion</a>	1	V1.0 = 0x10
1	<a href="#">RequestResponseCode</a>	1	0xFE = VENDOR_DEFINED_REQUEST
2	<a href="#">Reserved</a>	1	Reserved
3	<a href="#">Reserved</a>	1	Reserved
4	<a href="#">StandardID</a>	2	This field shall indicate the registry or standard body using one of the values in the ID column of <a href="#">Table 29</a> name.
6	<a href="#">Len</a>	1	Length of the <a href="#">Vendor ID</a> field. If the <a href="#">VendorDefinedRequest</a> is standard defined, Len shall be 0. If the <a href="#">VendorDefinedRequest</a> is vendor defined, Len shall equal <a href="#">Vendor ID Len</a> as described in <a href="#">Table 29</a> .
7	<a href="#">VendorID</a>	Len	The value of this field shall indicate Vendor ID, as assigned by the registry or standards body. This field shall be in little endian format.
7 + Len	<a href="#">ReqLength</a>	2	The length of the <a href="#">VendorDefinedReqPayload</a>
7 + Len + 2	<a href="#">VendorDefinedReqPayload</a>	ReqLength	This field shall be used by the standard/vendor to send the request payload

#### 4.10.5 VENDOR\_DEFINED\_RESPONSE response message

This response message can be used by a Responder in response to `VENDOR_DEFINED_REQUEST`. The format is defined in [Table 105](#).

[Table 105](#) shows the `VENDOR_DEFINED_RESPONSE` response message format.

**Table 105 — VENDOR\_DEFINED\_RESPONSE` response message**

Offset	Field	Size (bytes)	Value
0	SPDMVersion	1	V1.0 = 0x10
1	RequestResponseCode	1	0x7E = VENDOR_DEFINED_RESPONSE
2	Reserved	1	Reserved
3	Reserved	1	Reserved
4	StandardID	2	This field shall indicate the registry or standard body using one of the values in the ID column of <a href="#">Table 29</a> name.
6	Len	1	Length of the Vendor ID field. If the VendorDefinedRequest is standard defined, Len shall be 0. If the VendorDefinedRequest is vendor defined, Len shall equal Vendor ID Len as described in <a href="#">Table 29</a> .
7	VendorID	Len	The value of this field shall indicate Vendor ID, as assigned by the registry or standards body. This field shall be in little endian format.
7 + Len	RespLength	2	The length of the VendorDefinedRespPayload
7 + Len + 2	VendorDefinedRespPayload	ReqLength	This field shall be used by the standard/vendor to send the response payload

### 4.11 SPDM messaging control and discovery examples

#### 4.12 ANNEX A (informative)

The 1.0 revision of this specification will address scenarios where `GET_VERSION`, `VERSION`, `GET_CAPABILITIES`, `CAPABILITIES`, `NEGOTIATE_ALGORITHMS` and `ALGORITHMS` messages can be optional.

In the v1.0 release, figures and tables will be renumbered.



## 4.13 ANNEX B - Leaf certificate example

Certificate:

```

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

```

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

## 4.14 Change log

---

Version	Date	Description
0.9.0	2019-05-30	First draft version.

## 4.15 Bibliography

---

DMTF DSP4014, *DMTF Process for Working Bodies 2.6*, [https://www.dmtf.org/sites/default/files/standards/documents/DSP4014\\_2.6.pdf](https://www.dmtf.org/sites/default/files/standards/documents/DSP4014_2.6.pdf)