Document Identifier: DSP0274

Date: 2019-05-30

Version: 0.9.0

Security Protocol and Data Model 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 This document is available for public review and comment until superseded.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: None

Document Class: Normative

Document Status: Work in Progress

Document Language: en-US

CONTENTS

Figures

Tables

Foreword

The *Security Protocol and Data Model Specification* (DSP1000) was prepared by the <DMTF Editing Body> of the DMTF.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. For information about the DMTF, see http://www.dmtf.org.

Acknowledgments

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

Editor:

- Yu-Yuan Chen – Intel Corporation
- Mahesh Natu – Intel Corporation

Contributors:

- Richelle Ahlvers – Broadcom Inc.
- Lee Ballard – Dell Technologies
- Patrick Caporale – Lenovo
- Nigel Edwards - Hewlett Packard Enterprise
- Daniil Egranov – Arm Limited
- 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
- Scott Phuong – Cisco Systems Inc.
- Jeffrey Plank – Microchip
- Viswanath Ponnuru – Dell Technologies
- Hemal Shah – Broadcom Inc.
- Srikanth Varadarajan – Intel Corporation
- Xiaoyu Ruan - Intel Corporation

Introduction

The *Security Protocol and Data Model (SPDM) Specification* defines messages, data objects and sequences for performing message exchanges between two devices within a platform over a variety of transport and physical media. The message exchanges defined in this specification includes *authentication* of hardware identities and *measurement* for firmware identities. It is designed to be a common and effective protocol and data model that enables efficient access to low-level security capabilities and operations. The protocol and the data model are generic enough and can be used in conjunction with other mechanisms including those that are not defined by PMCI or DMTF.

Document conventions

Typographical conventions

The following typographical conventions are used in this document:

- Document titles are marked in *italics*.
- Important terms that are used for the first time are marked in *italics*.
- ABNF rules are in monospaced font.

ABNF usage conventions

Format definitions in this document are specified using ABNF (see RFC5234), with the following deviations:

- Literal strings are to be interpreted as case-sensitive Unicode characters, as opposed to the definition in RFC5234 that interprets literal strings as case-insensitive US-ASCII characters.

Deprecated material

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

Deprecated material should contain references to the last published version that included the deprecated material as normative material and to a description of the favored approach.

The following typographical convention indicates deprecated material:

DEPRECATED

Deprecated material appears here.

DEPRECATED

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

Experimental material

Experimental material has yet to receive sufficient review to satisfy the adoption requirements set forth by the DMTF. Experimental material is included in this document as an aid to implementers who are interested in likely future developments. Experimental material may change as implementation experience is gained. It is likely that experimental material will be included in an upcoming revision of the document. Until that time, experimental material is purely informational.

The following typographical convention indicates experimental material:

EXPERIMENTAL

Experimental material appears here.

EXPERIMENTAL

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

Security Protocol and Data Model (SPDM) Specification

1 Scope

This specification defines the messages, data objects and sequences for performing message exchanges between two devices within a platform 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. Mapping of these messages to different transports and physical media will be defined by other specifications.

This specification is not a system-level requirements document. The mandatory requirements stated in this specification apply when a particular message exchange capability is implemented through SPDM messaging in a manner that is conformant with this specification. This specification does not specify whether a given system or device is required to implement that message exchange capability. For example, this specification does not specify whether a given device must provide firmware measurements. However, if a device does implement firmware measurement or other capabilities described in this specification, the specification defines the requirements under SPDM.

2 Normative references

The following referenced documents are indispensable for the application of this document. 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*, http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype

IETF RFC5234, *ABNF: Augmented BNF for Syntax Specifications, January 2008*, http://tools.ietf.org/html/rfc5234

USB Authentication Specification Rev 1.0 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”, Revision 1.27 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-I!!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-I!!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-I!!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-I!!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-I!!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-I!!PDF-E&type=items.)

ECDSA:

- NIST-FIPS-186-4, Section 6 (available at: http://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: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.)

RSA: Available at: https://www.emc.com/collateral/white-papers/h11300-pkcs-1v2-2-rsa-cryptography-standard-wp.pdf

SHA2-256, SHA2-384 and SHA2-512:

- NIST-FIPS-180-4 (available at: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf)

SHA3-256, SHA3-384 and SHA3-512:

- NIST-FIPS-202 (available at: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf)

3 Terms and definitions

In this document, some terms have a specific meaning beyond the normal English meaning. Those terms are defined in this clause.

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.

The following terms are used in this document.

3.1
authentication

the process of determining whether an entity is in fact who or what it claims to be.

3.2
authentication initiator

the endpoint that initiates the authentication process by challenging another endpoint.

3.3

byte

an 8-bit quantity. Also referred to as an *octet*.

NOTE SPDM specifications shall use the term byte, not octet.

3.4

certificate

a digital form of identification that provides information about an entity and certifies ownership of a particular an asymmetric key-pair.

3.5

certificate authority

a trusted third-party entity that issues certificates.

3.6

certificate chain

a series of two or more certificates where each certificate is signed by the preceding certificate in the chain.

3.7

device

a physical entity such as a network card or a fan.

3.8

endpoint

a logical entity that communicates with other endpoints over one or more transport protocol.

3.9

intermediate certificate

a certificate that is neither a Root certificate nor a leaf certificate.

3.10

leaf certificate

the last certificate in a certificate chain.

3.11

message

see *SPDM message.*

3.12

message body

the portion of a SPDM message that carries data associated with the message.

3.13

message originator

the original transmitter (source) of a SPDM message.

3.14

most significant byte
MSB

the highest order byte in a number consisting of multiple bytes.

3.15

nonce

a 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.

3.16

nibble

the computer term for a four-bit aggregation, or half of a byte.

3.17

payload

the 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 are used to transport the message from one point to another. In some instances, a given field may be both a payload field and a transport field.

3.18

physical transport binding

refers to specifications that define how a base messaging protocol is implemented on a particular physical transport type and medium, such as SMBus/I^2C, PCI Express™ Vendor Defined Messaging, and so on.

3.19

SPDM endpoint

a SPDM endpoint is defined as the point of communication termination for SPDM messages and the SPDM functions associated with those messages.

3.20

SPDM message

a unit of communication that is used for SPDM communications.

3.21

SPDM message payload

a portion of the message body of a SPDM message
This portion of the message is separate from those fields and elements that are used to identify the SPDM version, the SPDM request/response codes, and the two parameters.

3.22

SPDM request message

a message that is sent to a SPDM endpoint to request a specific SPDM operation
A SPDM request message is acknowledged with a corresponding SPDM response message.

3.23

SPDM response message

a message that is sent in response to a specific SPDM request message

This message includes a "Response Code" field that indicates whether the requested operation completed normally.

3.24

Platform Management Component Intercommunications

PMCI

the name of a working group under the Distributed Management Task Force that is chartered to define 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.

3.25

requestor

the original transmitter (source) of an SPDM message.

3.26

responder

the ultimate receiver (destination) of an SPDM message.

3.27

Root Certificate

the first certificate in a certificate chain. This certificate is self-signed.

4 Symbols and abbreviated terms

The following abbreviations are used in this document.

4.1

MSB

most significant byte

4.2

SPDM

Security Protocol and Data Model

4.3

PMCI

Platform Management Component Intercommunications

5 Conventions

The conventions described in the following clauses apply to all of the SPDM specifications.

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, numeric or bit fields that are designated as reserved shall be written as 0 (zero) and ignored when read.

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).

5.3 SPDM data types

Table 1 lists the abbreviations and descriptions for common data types that are used in SPDM message fields and data structure definitions. These definition follow DSP0240 – PLDM Base Specification.

Table 1 – SPDM data types

Data Type	Interpretation
ver8	An eight-bit encoding of the SPDM version number. The encoding of the version number is defined in Section 5.4. [7:4] = major version number [3:0] = minor version number
bitfield8	A byte with 8 bit fields. Each of these bit fields can be separately defined.
bitfield16	A 2-byte word with 16 bit fields. Each of these bit fields can be separately defined.

5.4 Version encoding

The version field represents the version of the specification and is comprised of two bytes referred to as the "major" and "minor" nibbles and one byte of detailed version. The major and minor nibbles shall be encoded as follows:

- Major and Minor version fields in such a representation match corresponding major and minor version fields in the SPDMVersion field in the SPDM message header.
- Minor version is incremented when the protocol is modified while maintaining backward compatibility.
- Major version is incremented when the protocol is modified in a manner that breaks 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, but the available functionality is limited to what is defined in SPDM specification Version 1.0.

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

The detailed version byte resides in the CAPABILITIES response message payload and is incremented to indicate specification bug fixes.

5.5 Notations

The following notations are used for SPDM specifications:

- M:N In field descriptions, this will typically be used to represent a range of byte offsets starting from byte M and continuing to and including byte N (M ≤ N). The lowest offset is on the left, and the highest is on the right.
- rsvd Abbreviation for Reserved. Case insensitive.
- [4] Square brackets around a number are typically used to indicate a bit offset. Bit offsets are given as zero-based values (that is, the least significant bit [LSb] offset = 0).
- [7:5] A range of bit offsets. The most-significant is on the left, and the least-significant is on the right.
- `1b` A lowercase "`b`" after a number consisting of `0`s and `1`s indicates that the number is in binary format.
- `0x12A` A leading "`0x`" indicates that the number is in hexadecimal format.

6 SPDM message exchanges

The message exchanges defined in this specification include:

1) an endpoint discovering and negotiating the security capabilities of another endpoint.

2) an endpoint authenticating the hardware identity of another endpoint.

3) an endpoint retrieving the firmware measurement for another endpoint's firmware identity.

These message exchange capabilities are built on top of well-known and established security practices across the computing industry. Brief overview for each of the message exchange capabilities are 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.

All message exchanges between two endpoints are performed and exchanged through sending and receiving of the SPDM messages defined in Section 8. 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.

6.1 Security capability discovery and negotiation

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

6.2 Hardware identity authentication

In this specification, the authenticity of an endpoint is determined by digital signatures using well-established techniques based on public key cryptography. An endpoint proves its hardware identity by generating digital signatures using a private key that is known only to that particular endpoint, and the signature can be verified by another endpoint using the public key associated with that private key. The authentication initiator can cryptographically verify the uniqueness of the endpoint, given that the private key is known only to that particular endpoint,

At a high-level, the authentication of an endpoint's hardware identity involves two processes—*identity provisioning* and *runtime authentication*. Identity provisioning is a 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 hardware. 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 is the process by which an authentication initiator interacts with an endpoint in a running system. The authentication initiator can retrieve the certificate(s) from the endpoint and send a unique challenge to the endpoint. The endpoint then signs the challenge with the private key. The authentication initiator verifies the signature using the public keys of the endpoint and the root CA, as well as any intermediate public keys within the certificate chain.

6.3 Firmware identity through measurement

In this specification, measurement is a term that describes the process of calculating the cryptographic hash value of a piece of firmware/software and tying the cryptographic hash value with the hardware identity through the use of digital signatures. Therefore, not only the identity of a piece of firmware/software can be established, the generation of the identity can be guaranteed to originate from a particular hardware endpoint.

7 SPDM messaging protocol

The SPDM messaging protocol defines a request-response messaging model between two endpoints to perform the message exchanges outlined in Section 6. Each SPDM request message shall be responded to with a SPDM response message as defined in this specification.

Figure 1 depicts the high-level request-response flow diagram for SPDM. As shown in Figure 1, an endpoint acting as the requestor sends a SPDM request message to another endpoint acting as the responder, and the responder sends back a SPDM response message to the requestor. The requestor repeats the process by issuing different request messages to

1. Discover and negotiate the security capabilities of the responder
2. Authenticate the responder's hardware identity
3. Retrieve the responder's firmware measurements.

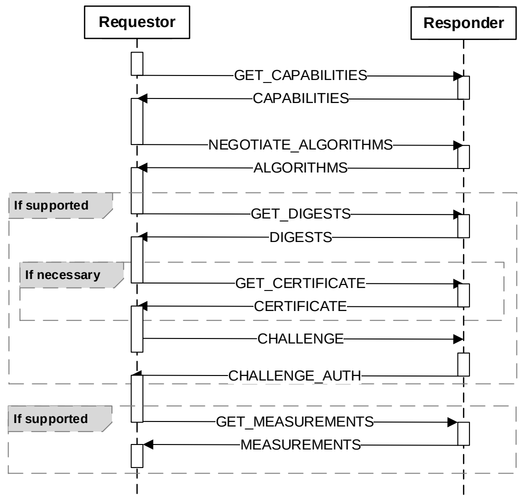


Figure 1 – SPDM messaging protocol flow

All SPDM request-response messages share a common data format, consisting of a 4-byte message header and zero or more bytes message payload that is message-dependent. The following sections describe the common message format and Section 8 details each of the request and response messages.

The requestor shall issue GET_CAPABILTIES followed by NEGOTIATE_ALGORITHMS request messages prior to issuing any other request messages.

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 format

<table>
<tr><td colspan="8">Byte 1</td><td colspan="8">Byte 2</td><td colspan="8">Byte 3</td><td colspan="8">Byte 4</td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="4">SPDM major version</td><td colspan="4">SPDM minor version</td><td colspan="8">Request Response Code</td><td colspan="8">Param1</td><td colspan="8">Param2</td></tr>
<tr><td colspan="32">SPDM message payload (zero or more bytes)</td></tr>
</table>

Table 3 defines the fields that are part of a generic SPDM message.

Table 3 – Generic SPDM message field definitions

Field Name	Field Size	Description
SPDM major version	4 bits	This field identifies which major version of the SPDM Specification is being used. An endpoint shall not communicate using an incompatible SPDM Version value. See section 5.4.
SPDM minor version	4 bits	This field identifies which minor version of the SPDM Specification is being used. A specification with a given minor version extends a specification with a lower minor version as long as they share the major version. See section 5.4.
Request Response Code	8 bits	Describes the request message code or response code. Enumerated in Table 4 and Table 5. 0x00 – 0x7F are used for response codes and 0x80 – 0xFF are used for request codes.
Param1	8 bits	This field is used to pass a first 1-byte parameter. The contents of the parameter is specific to the Request Response Code.
Param2	8 bits	This field is used to pass a second 1-byte parameter. The contents of the parameter is specific to the Request Response Code.
SPDM message payload	Variable	The SPDM message payload is zero or more bytes that are specific to the Request Response Code.

7.2 SPDM Request Codes

Table 4 defines the SPDM request codes for SPDM. All SPDM-compatible implementations shall use the following request codes.

Table 4 – SPDM request codes

Request	Code Value	Requirement	Message Format
GET_DIGESTS	0x81	Optional	See Table 12
GET_CERTIFICATE	0x82	Optional	See Table 14.
CHALLENGE	0x83	Optional	See Table 16.
GET_MEASUREMENTS	0xE0	Optional	See Table 18.
GET_CAPABILITIES	0xE1	Mandatory	See Table 7.
SET_CERTIFICATE	0xE2	Optional	To be defined in a future version.
NEGOTIATE_ALGORITHMS	0xE3	Mandatory	See Table 10.
RESPOND_IF_READY	0xFF	Mandatory for all requestors. Mandatory for all responders that return ERROR code of *ResponseNotReady*	See Table 24
Reserved	0x80, 0x84 – 0xDF, 0xE4 – 0xFE	SPDM implementations compatible with this version shall not use the reserved request codes.	

7.3 SPDM response codes

The Request Response Code field in the SPDM response message shall be used to specify the appropriate response code for a given 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, ERROR response message shall be returned.

Table 5 defines the response codes for SPDM.

Table 5 – SPDM response codes

Response	Value	Description	Message Format
DIGESTS	0x01	Successful response to GET_DIGESTS request message. Mandatory for endpoints that support GET_DIGESTS request message.	See Table 13.
CERTIFICATE	0x02	Successful response to GET_CERTIFICATE request message. Mandatory for endpoints that support GET_CERTIFICATE request message.	See Table 15.
CHALLENGE_AUTH	0x03	Successful response to CHALLENGE. Mandatory for endpoints that support CHALLENGE request message.	See Table 17.

Response	Value	Description	Message Format
MEASUREMENTS	0x60	Successful response to GET_MEASUREMENTS request message. Mandatory for endpoints that support GET_MEASUREMENTS request message.	See Table 19.
CAPABILITIES	0x61	Successful response to GET_CAPABILITIES request message. Mandatory for all SPDM endpoints.	See Table 7.
SET_CERT_RESPONSE	0x62	Successful response to SET_CERTIFICATE request message. Mandatory for endpoints that support SET_CERTIFICATE request message.	To be defined in a future version.
ALGORITHMS	0x63	Successful response to NEGOTIATE_ALGORITHMS request message. Mandatory for all SPDM endpoints.	See Table 11.
ERROR	0x7F	Response to any unsuccessful request message. Mandatory for all SPDM endpoints.	See Table 21 and Table 22.
Reserved	0x00, 0x04–0x5F, 0x64 – 0x7E	SPDM implementations compatible with this version shall not use the reserved response codes.	

7.4 Concurrent SPDM command processing

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

7.4.1 Requirements for responders

A responder shall process SPDM message requests from a given requestor in order.

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.

An SPDM endpoint 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 an SPDM endpoint is working on a request message from a requestor, then the SPDM endpoint shall be able to process (or queue up processing) and send the response message independently from sending its own request message.

If an SPDM endpoint is working on a request message from a requestor, then the SPDM endpoint shall be allowed to respond with ErrorCode=*ResponseNotReady.*

If a responder allows simultaneous communications with multiple requestors, the responder shall use the following fields to track a SPDM request message:

- the transport address (which is transport-binding specific) of the requestor
- SPDM request code
- Param1
- Param2.

7.4.2 Requirements for requestors

.

An SPDM endpoint requestor shall not issue another request message to the same endpoint with the exception of GET_CAPABILITIES request message until it either gets the response message to a particular request message, times out waiting for the response message, or receives an indication that transmission of the particular request message failed, before issuing a new SPDM request message. An SPDM requestor may issue GET_CAPABILITIES request message at any time.

An SPDM endpoint is permitted to send multiple simultaneous request messages outstanding to different SPDM endpoints.

The timing specifications shown in Table 6 are specific to SPDM request messages. The SPDM response messages are not retried. A “try” or “retry” of a request message is defined as a complete transmission of the SPDM request message. All timeout reports in Table 6 are worst case values.

Table 6 – Timing and retry specifications for SPDM messages

Timing Specification	Symbol	Min	Max	Description
Number of request retries	SN1	2	See "Description"	If the requestor does not receive a response within the request-to-response time, the requestor shall try a request message at least three times - the original attempt try plus two retries, prior to treating it as an error condition. The maximum number of retries for a given request message may be further limited by the underlying transport specification.
Request-to-response time for GET_CAPABILITIES request message	ST1	–	100 ms	If the underlying media or other layers have more stringent timeout requirements, SPDM responder should obey those.
Request-to-response time for all request messages except GET_CAPABILITIES request	ST2	-	CT	CT is reported via CAPABILITIES response message. The duration CT may exceed the timeout values associated with the underlying transport or media layers. The responder should avoid such timeouts by responding with ERROR with ErrorCode=*ResponseNotReady* response message if necessary. Requestor may respond by sending RESPOND_IF_READY request message until request to response message timeout is reached.

8 SPDM messages

SPDM messages can be divided into three categories, supporting different aspects of security exchanges between two endpoints

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

8.1 Capability discovery and negotiation

All SPDM endpoints shall support GET_CAPABILITIES and NEGOTIATE_ALGORITHMS both as a requestor and as a responder. The high-level request-response flow and sequence for the capability discovery and negotiation are shown in Figure 2.

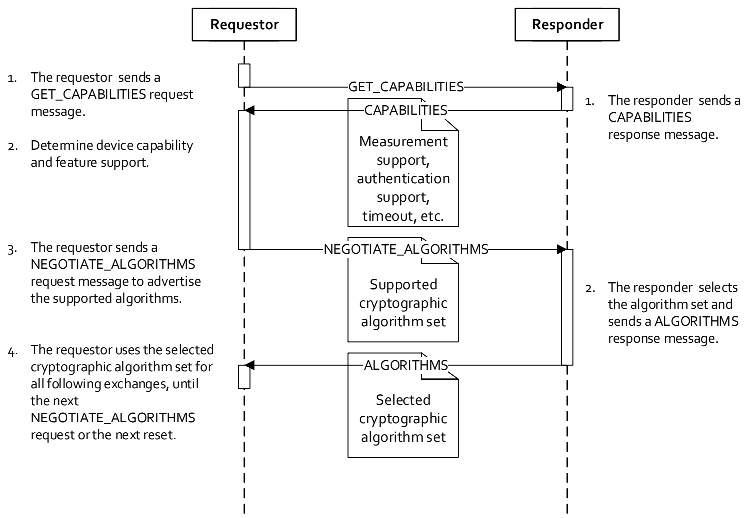


Figure 2 – Capability discovery and negotiation flow

8.1.1 GET_CAPABILITIES request message and CAPABILITIES response message

This request message shall be used to retrieve an endpoint's security capabilities. The request message format is shown in Table 7 and the response message format is shown in Table 8. GET_CAPABILITIES request message and CAPABILITIES response message in all future SPDM major versions will be backward compatible with all previous major versions.

If the requestor supports multiple SPDM major versions, the requestor shall begin the discovery process by sending a GET_CAPABILITIES request message that advertises the highest supported major version. If the responder does not support this major version, it shall return ERROR response with ErrorCode of *MajorVersionMismatch* along with a bitmap of supported major versions. The requestor shall consult the bitmap to select the highest common major version supported and issue GET_CAPABILITIES request message. A requestor is not permitted to issue NEGOTIATE_ALGORITHMS request until it has received a successful CAPABILITIES response and identified a common major version supported by both sides.

A responder is not permitted to respond to GET_CAPABILITIES request message with ErrorCode=*ResponseNotReady*.

An SPDM requestor may issue GET_CAPABILITIES request message at any time.

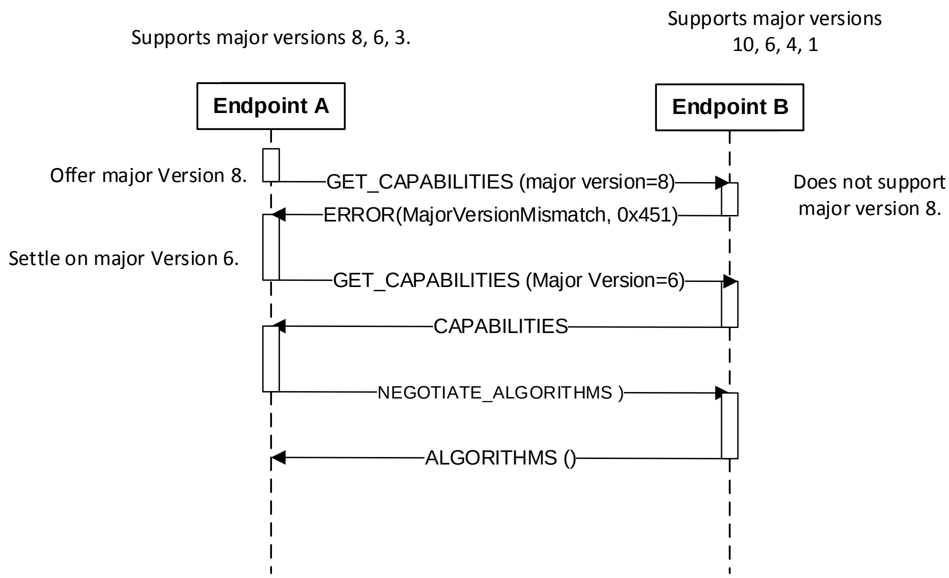


Figure 3 – Discovering common major version

Table 7 – GET_CAPABILITIES request message

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

Table 8 – Successful CAPABILITIES response message

Offset	Field	Size in bytes	Value
0	*SPDMVersion*	1	V1.0 = 0x10
1	*RequestResponseCode*	1	0x61 = CAPABILITIES
2	*Param1*	1	Reserved
3	*Param2*	1	Reserved

Offset	Field	Size in bytes	Value
4	*DetailedVersion*	1	Detailed version. See Section 5.4.
5	*CryptographicTimeout (CT)*	1	The requestor shall add this value is base timeout value when deriving request-to-response timeout for request messages other than GET_CAPABILITIES. See Table 6. For example, CT=10 implies the worst case duration of 2^{10} =1024 uS. Calculation of CT shall account of the possibility that the responder may receive such requests from multiple endpoints.
6	*Reserved*	2	Reserved
8	*Flags*	4	See Table 9.
12	*SPDMMajorVersions*	2	Bitmap representing the SPDM major version supported by the responder. For example, return value of 0x24 implies responder supports SPDM major versions 5 and 2.
14	*Reserved*	2	Reserved

Table 9 – Flags Fields Definition

Byte	Bit Position	Field	Value
0	0	*Reserved*	Reserved
0	1	*AUTH_CAP*	1 - Supports GET_DIGESTS, GET_CERTIFICATE and CHALLENGE request messages 0 - otherwise
0	2	*Reserved*	Reserved
0	3	*MEAS_CAP*	1 - Supports GET_MEASUREMENTS request message 0 - otherwise
0	4	*MEAS_FRESH_CAP*	0: As part of MEASUREMENTS response message, the responder may return measurements that were computed during the last responder's reset 1: The responder is capable of recomputing 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.
0	7:5	*Reserved*	Reserved
1	7:0	*Reserved*	Reserved
2	7:0	*Reserved*	Reserved
3	7:0	*Reserved*	Reserved

8.1.2 NEGOTIATE_ALGORITHMS request message and ALGORITHMS response message

This request message shall be used to negotiate cryptographic algorithms. A requestor is not permitted to issue NEGOTIATE_ALGORITHMS request message until it has received a successful CAPABILITIES response. A requestor is not permitted to issue any other SPDM requests with the exception of

GET_CAPABILITIES until it has received a successful ALGORITHMS response with exactly one asymmetric and exactly one hashing algorithm.

The request message format is shown in Table 10 and the response message format is shown in Table 11.

Table 10 – NEGOTIATE_ALGORITHMS request message

Offset	Field	Size in bytes	Value
0	*SPDMVersion*	1	V1.0 = 0x10
1	*RequestResponseCode*	1	0xE3 = NEGOTIATE_ALGORITHMS
2	*Param1*	1	Reserved
3	*Param2*	1	Reserved
4	*Length*	2	Length of the entire request message packet in bytes
6	*MeasurementSpecification*	1	Bit Mask – Bit position based on "Measurement Specification as defined in section 8.3.2. Bit 7 Reserved for extension indication.
7	*Reserved*	1	Reserved
8	*BaseAsymAlgo*	4	Bit mask listing SPDM enumerated asymmetric algorithms supported by requestor for the purposes of signature verification. Bit 0 – TPM_ALG_RSASSA_2048 Bit 1 – TPM_ALG_RSASSA_3072 Bit 2 – TPM_ALG_ECDSA_ECC_NIST_P256 Bit 3 – TPM_ALG_RSASSA_4096 Bit 4 – TPM_ALG_ECDSA_ECC_NIST_P384 Bit 5 – TPM_ALG_ECDSA_ECC_NIST_P521 All RSA based algorithms shall use PSS padding and exponent of 65537.
12	*BaseHashAlgo*	4	Bit mask listing SPDM enumerated cryptographic hashing algorithms supported by requestor. Bit 0 – SHA2-256 Bit 1 – SHA3-256 Bit 2 – SHA2-384 Bit 3 – SHA3-384 Bit 4 – SHA2-512 Bit 5 – SHA3-512
16	*Reserved*	8	Reserved
24	*ExtAsymCount*	1	Number of extended asymmetric algorithms supported by requestor (=A)
25	*ExtHashCount*	1	Number of extended hashing algorithms supported by requestor (=H)
26	*Reserved*	2	Reserved for future use
28	*ExtAsym*	4*A	List of the extended asymmetric algorithms supported by requestor. First byte in each entry is enumeration for the encoding for ExtAsym 0 – DMTF; 1 – TCG The second byte is reserved and the other two

Offset	Field	Size in bytes	Value
			bytes represent the algorithm encoding. At this time, the DMTF namespace has no algorithms defined. TCG algorithms are enumerated in TCG Algorithm Registry.
28+4*A	*ExtHash*	4*H	List of the extended hashing algorithms supported by requestor. First byte in each entry is enumeration for the encoding for ExtHash 0 – DMTF; 1 – TCG The second byte is reserved and the other two bytes represent the algorithm encoding. At this time, the DMTF namespace has no algorithms defined. TCG algorithms are enumerated in TCG Algorithm Registry.
28+4*A+ 4*H	*Reserved*	Length – 28 – 4* A – 4*H	Reserved for future expansion. Consult the Length field (offset 4) to determine the number of bytes in the request message.

598 **Table 11 –Successful ALGORITHMS response message**

Offset	Field	Size in bytes	Value
0	*SPDMVersion*	1	V1.0 = 0x10
1	*RequestResponseCode*	1	0x63 = ALGORITHMS
2	*Param1*	1	Reserved
3	*Param2*	1	Reserved
4	*Length*	2	Length of the response message packet in bytes
6	*MeasurementSpecification*	1	The specification that governs the format of Measurement Block. 0 – DMTF. All other encodings are reserved
7	*MeasurementHashAlgo*	1	Bit mask listing SPDM enumerated hashing algorithm for measurements. M represents the length of the measurement hash field in Measurement Block structure (Table 20). The responder shall ensure the length of measurement hash field during all subsequent MEASUREMENT response messages to the requestor until the next ALGORITHMS response message is M. Bit 0 – SHA2-256, M=32 Bit 1 – SHA3-256, M=32 Bit 2 – SHA2-384, M=48 Bit 3 –SHA3-384, M=48 Bit 4 – SHA2-512, M=64 Bit 5 – SHA3-512, M=64 If the responder supports GET_MEASUREMENT, exactly 1 bit in this bit field shall be set. Otherwise, the responder shall set this field to 0.
8	*BaseAsymSel*	4	Bit mask listing SPDM enumerated asymmetric algorithm selected. Responder must be able to sign a response message using this algorithm and requestor must have listed this algorithm in the Request indicating it can verify a response message using this algorithm. The responder shall use this asymmetric signature algorithm during all subsequent applicable response

Offset	Field	Size in bytes	Value
			messages to the requestor until the next ALGORITHMS response message. A requestor that returns AUTH_CAP=0 and MEAS_CAP=0 shall set this field 0. Other requestors shall set no more than 1 bit.
12	*BaseHashSel*	4	Bit mask listing SPDM enumerated hashing algorithm selected. The responder shall use this hashing algorithm during all subsequent response messages to the requestor until the next ALGORITHMS response message. The requestor shall use this hashing algorithm during all subsequent applicable request messages to the responder until the next ALGORITHMS response message. The length of the nonce and salt fields exchanged during subsequent request messages and response messages, and any other fields specified in the request message and response message format, shall match the length of the selected hash, until the next ALGORITHM response message. A requestor that returns AUTH_CAP=0 and MEAS_CAP=0 shall set this field 0. Other requestors shall set no more than 1 bit.
16	*Reserved*	8	Reserved.
24	*ExtAsymSelCount*	1	The number of extended asymmetric algorithms selected. Shall be either 0 or 1. (=A) A requestor that returns AUTH_CAP=0 and MEAS_CAP=0 shall set this field 0.
25	*ExtHashSelCount*	1	The number of extended hashing algorithms selected. Shall be either 0 or 1. (=H) A requestor that returns AUTH_CAP=0 and MEAS_CAP=0 shall set this field 0.
26	*Reserved*	2	Reserved
28	*ExtAsymSel*	4*A	The extended asymmetric algorithm selected. Responder must be able to sign a response message using this algorithm and requestor 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 during all subsequent applicable response messages to the requestor until the next ALGORITHMS response message. First byte is enumeration for the encoding for ExtAsymSel 0 – DMTF; 1 – TCG The second byte is reserved and the other two bytes represent the algorithm encoding. At this time, the DMTF namespace has no algorithms defined. TCG algorithms are enumerated in TCG Algorithm Registry.
28+4*A	*ExtHashSel*	4*H	The extended Hashing algorithm selected. The responder shall use this hashing algorithm during all subsequent response messages to the requestor until the next ALGORITHMS response message. The requestor shall use this hashing algorithm during all subsequent applicable request messages to the responder until the next ALGORITHMS response message. The length of the nonce and salt fields exchanged during subsequent applicable request messages and response messages shall match the length of the selected hash, until the

Offset	Field	Size in bytes	Value
			next ALGORITHM response message. First byte is enumeration for the encoding for ExtHashSel 0 – DMTF; 1 – TCG The second byte is reserved and the other two bytes represent the algorithm encoding. At this time, the DMTF namespace has no algorithms defined. TCG algorithms are enumerated in TCG Algorithm Registry.
28+4*A +4*H	*Reserved*	Length – 28 – 4*A – 4*H	Reserved for future expansion. Consult the length field (offset 4) to determine the total number of bytes in the response message.

8.1.3 Algorithm negotiation rules

Under certain usage models, it may be possible to guarantee that a single SPDM endpoint in any given pair will be the one issuing NEGOTIATE_ALGORITHMS request message. However, this assumption may not hold under all usage models. Therefore, SPDM architecture accounts for the possibility that both endpoints may issue NEGOTIATE_ALGORITHMS request message independent of each other. SPDM specification defines specific rules to ensure both endpoints select consistent algorithms regardless of which or how many endpoints in a given pair initiate the negotiation. These rules ensure that

1. The two SPDM endpoints shall agree on a single hashing algorithm.
2. The two SPDM endpoints shall agree on a single asymmetric algorithm in each direction. It is permitted for the asymmetric signature algorithm employed when endpoint A is acting as the challenger to be different from the asymmetric signature algorithm employed when its peer is acting as the challenger.

SPDM endpoints shall follow the below rules during construction of ALGORITHMS response message.

1. The following priority is established within asymmetric signature algorithms. ALGORITHMS response message shall select the highest priority algorithm if the responder is able to sign using multiple algorithms out of those specified in NEGOTIATE_ALGORITHMS request message. The priority order for the currently defined asymmetric algorithms shall be (from highest to lowest priority).
    1. TPM_ALG_ECDSA_ECC_NIST_P521
    2. TPM_ALG_ECDSA_ECC_NIST_P384
    3. TPM_ALG_RSASSA_4096
    4. TPM_ALG_ECDSA_ECC_NIST_P256
    5. TPM_ALG_RSASSA_3072
    6. TPM_ALG_RSASSA_2048
2. The following priority is established within hashing algorithms. ALGORITHMS response message shall select the highest priority algorithm if the responder is capable of hashing using multiple algorithms out of those specified in NEGOTIATE_ALGORITHMS request message. The priority order for the currently defined hashing algorithms shall be (from highest to lowest priority).
    1. SHA3-512
    2. SHA2-512

3. SHA3-384
4. SHA2-384
5. SHA3-256
6. SHA2-256

2. If common base hashing algorithm(s) are available, ALGORITHMS response message shall never select an extended hashing algorithm. If common base asymmetric signature algorithm(s) are available, ALGORITHMS response message shall never select an extended asymmetric signature algorithm.

3. If extended algorithms within more than one namespace are supported by the two negotiating endpoints, ALGORITHMS response message shall select an algorithm in TCG namespace. The namespace encoding is defined by the first byte of *ExtAsymSel* and *ExtHashSel*.

4. If more than one extended algorithms within a given namespace are supported by the negotiating endpoints, ALGORITHMS response message shall select the one with numerically higher encoding.

8.2 Endpoint hardware identity authentication

This section describes request messages and response messages associated with endpoint hardware identity authentication operations. All request messages in this section shall be supported by an endpoint that returns AUTH_CAP=1 in the CAPABILITIES response message. The high-level request-response message flow and sequence for endpoint hardware identity authentication are shown in Figure 4 for certificate retrieval and Figure 5 for runtime challenge-response.

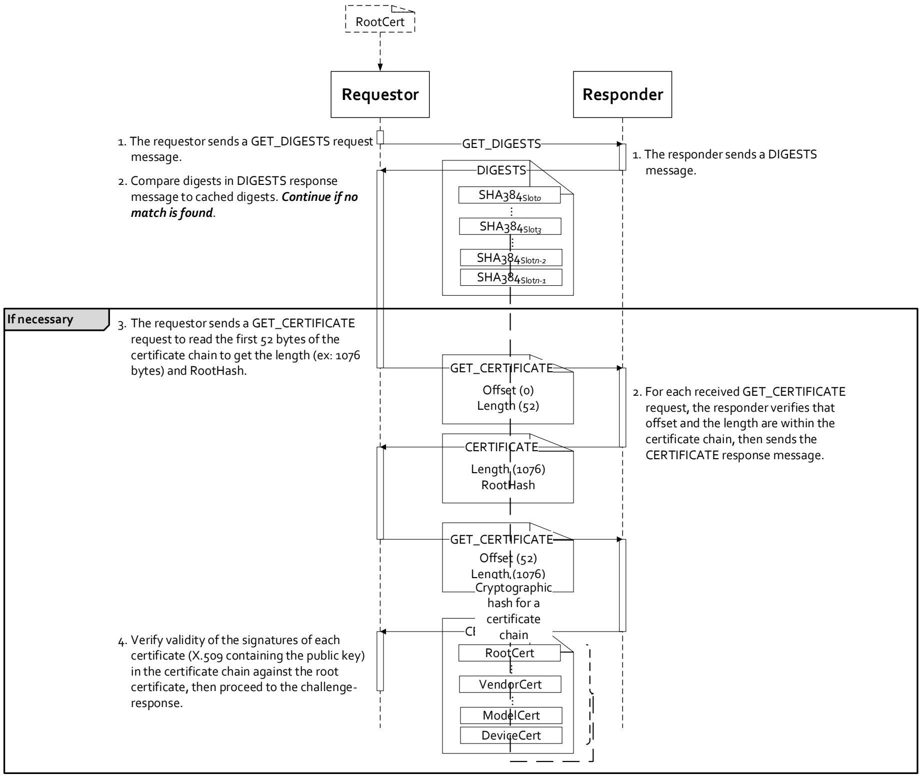


Figure 4 – Endpoint authentication: example certificate retrieval flow.

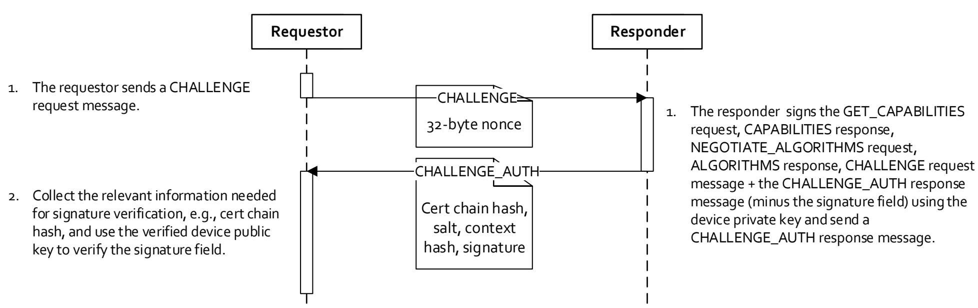


Figure 5 – Endpoint authentication: runtime challenge-response flow.

Each SPDM endpoint that supports this capability shall carry at least one certificate chain or a single certificate. The minimum number of certificates within a chain should be two and may include the device-specific certificate and the root certificate that is self-signed by the certificate authority. Each certificate shall be ASN.1 DER-encoded X509v3 format. The device shall contain only a single pair of public-private key pair for its hardware identity, regardless of how many certificate chains are stored on the device.

The GET_DIGESTS request message and DIGESTS response message may be used to optimize the amount of data required to be transferred from the responder to the requestor, due to the potentially large size of a certificate chain. The cryptographic hash values of all of the certificate chains stored on an endpoint is returned with the DIGESTS response message, such that the requestor 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 GET_CAPABILITIES request, CAPABILITIES response, NEGOTIATE_ALGORITHMS request, ALGORITHMS response, CHALLENGE request message and the CHALLENGE_AUTH response message except for the signature field, to ensure cryptographic binding between a specific request message from a specific requestor and a specific response message from a specific responder. Inclusion of GET_CAPABILITIES request, CAPABILITIES response, NEGOTIATE_ALGORITHMS request and ALGORITHMS response allows the responder to detect the presence of an active adversary attempting to downgrade cryptographic algorithms or SPDM major versions. Furthermore, a nonce generated by the requestor protects the challenge-response from replay attacks, whereas a salt generated by the responder prevents the responder from signing over arbitrary data dictated by the requestor.

8.2.1 GET_DIGESTS request message and DIGESTS response message

This request message shall be used to retrieve the certificate chain digests. The request message format is shown in Table 12 and the response message format is shown in Table 13.

Table 12 – GET_DIGESTS request message

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

Table 13 –Successful DIGESTS response message

Offset	Field	Size in 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	H-byte digest of the first certificate chain. H is the size of the hashing algorithm output mutually agreed upon via NEGOTIATE_ALGORITHMS request message. This field is big endian.
…	…	…	…

Offset	Field	Size in bytes	Value
4 + (H * (n -1))	*Digest[n-1]*	H	H-byte digest of the last (n^{th}) certificate chain. H is the size of the hashing algorithm output mutually agreed upon via NEGOTIATE_ALGORITHMS request message. This field is big endian.

8.2.2 GET_CERTIFICATE request message and CERTIFICATE response message

This request message shall be used to retrieve the certificate chains, one chunk at a time. The request message format is shown in Table 14 and the response message format is shown in Table 15. The responder should, at a minimum save the public key of the leaf certificate and associate with each of the digests returned by DIGESTS message response.

Table 14 – GET_CERTIFICATE request message

Offset	Field	Size in 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.
6	*Length*	2	Length in bytes of the read request message. Length is an unsigned 16-bit integer. If offset=0 & length=0xFFFF, the entire chain will be returned from the device. If a device cannot return the entire chain, it shall return the ERROR response message with the *RequestedInfoTooLong* error code.

Table 15 –Successful CERTIFICATE response message

Offset	Field	Size in bytes	Value
0	*SPDMVersion*	1	V1.0 = 0x10
1	*RequestResponseCode*	1	0x02 = CERTIFICATE
2	*Param1*	1	Slot number of the certificate chain returned
3	*Param2*	1	Reserved
4	*CertChain*	Length	Data Requested contents of target certificate chain, formatted in DER. This field is big endian.

8.2.3 Leaf certificate format requirements

1. Version -Version of encoded certificate shall be present and shall be 3 (value 2).
2. Serial Number - CA assigned serial number shall be present with a positive integer value.
3. Signature Algorithm - Signature algorithm used by CA shall be present.
4. Issuer -CA distinguished name shall be specified.
5. Subject Name – Subject name shall be present and shall represent the distinguished name associated with the leaf certificate.
6. Validity - The certificate may include this attribute. If validity attribute is present, the value for *notBefore* field should be assigned the generalized time value "19700101000000Z" and *notAfter* field should be assigned the generalized time value of "99991231235959Z".
7. Subject Alternative Name- The directory name in the Subject Alternative Name should be present and populated with the following fields. If present, the following rules apply.
    a. Organization Unit –This field shall be DMTF.
    b. Common Name - The common name shall be manufacturer="manufacturer name":product="product name" pattern where "manufacturer name" is the vendor name and "product name" is the textual description of the device.
    c. Serial Number –This field shall be the textual value for device serial number.
8. Subject Public Key Info - The device public key and the algorithm shall be present.
9. Basic Constraints - Basic Constraints field shall be present with the CA value set to false.
10. Extended Key Usage - Extended Key Usage field shall be present and key usage bit for digital signature shall be set.

8.2.4 CHALLENGE request message and CHALLENGE_AUTH response message

This request message shall be used to authenticate an endpoint via challenge-response protocol. The request message format is shown in Table 16 and the response message format is shown in Table 17.

Table 16 – CHALLENGE request message

Offset	Field	Size in 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
3	*Param2*	1	Reserved
4	*Nonce*	H	Random H-byte nonce, a random value chosen by the authentication initiator. H is the size of the hashing algorithm output mutually agreed upon via ALGORITHMS response message BaseHashSel or ExtHashSel field.

Table 17 – Successful CHALLENGE_AUTH response message

Offset	Field	Size in 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
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	*MinSPDMVersion*	1	Minimum SPDM version supported by this endpoint
5	*MaxSPDMVersion*	1	Maximum SPDM version supported by this endpoint
6	*Capabilities*	1	Set to 01h for this specification. All other values reserved
7	*Reserved*	1	Reserved
8	*CertChainHash*	H	Hash of the certificate chain used for authentication. H is the size of the hashing algorithm output mutually agreed via NEGOTIATE_ALGORITHMS request message. This field is big endian.
8+H	*Salt*	H	Value chosen by the authentication Responder. H is the size of the hashing algorithm output mutually agreed via NEGOTIATE_ALGORITHMS request message. *Note: the Salt shall be unique per response message for the duration of a device reset cycle.*
8+2H	*ContextHash*	H	Reserved
8+3H	*Signature*	S	S is the size of the asymmetric signing algorithm output the responder selected via the last ALGORITHMS response message to the requestor. Signature generation and verification processes are defined in sections 8.2.5 and 8.2.6 respectively.

8.2.5 Signature Generation

Symbols ending with the number 1 represent the messages as observed by the responder.

Step 1: The responder shall construct M1

```
M1 = Concatenate(GET_CAPABILITIES_REQUEST1, CAPABILITIES_RESPONSE1,
NEGOTIATE_ALGORITHMS_REQUEST1, ALGORITHMS_RESPONSE1, CHALLENGE_REQUEST1,
CHALLENGE_AUTH_RESPONSE_WITHOUT_SIGNATURE1)
```

Where *Concatenate ( )* is the standard concatenation function

- *GET_CAPABILITIES_REQUEST1* is the entire contents of the last successful GET_CAPABILITIES request message processed by the responder.

CAPABILITIES_RESPONSE1 is the entire contents of the associated response message sent by the responder. Constructing M1 may require that the responder preserve the contents of these prior messages.

- *NEGOTIATE_ALGORITHMS _REQUEST1* is the entire contents of the last successful NEGOTIATE_ALGORITHMS request message processed by the responder. *ALGORITHMS_RESPONSE1* is the entire contents of the associated response message sent by the responder. Constructing M1 may require that the responder preserve the contents of these prior messages.
- *CHALLENGE_REQUEST1* is the entire contents of the CHALLENGE request message under consideration, as seen by the responder. *CHALLENGE_AUTH_RESPONSE_WITHOUT_SIGNATURE1* is the entire CHALLENGE_AUTH response message without the signature bytes, as sent by the responder.

Step 2: The responder shall generate

```
Signature = Sign(SK, Hash1(M1))
```

Where

Sign is the asymmetric signing algorithm the responder selected via the last ALGORITHMS response message sent by the responder. Refer to *BaseAsymSel* or *ExtAsymSel* fields in Table 11.

Hash1 is the hashing algorithm the responder selected via the last ALGORITHMS response message sent by the responder. Refer to *BaseHashSel* or *ExtHashSel* fields in Table 11.

SK = the private Key associated with the responder's leaf certificate in slot=Param1 of CHALLENGE request message.

8.2.6 Signature Verification

Symbols ending with the number 2 represent the messages as observed by the requestor.

Step1: The requestor shall create M2 as

```
M2 = Concatenate (GET_CAPABILITIES_REQUEST2, CAPABILITIES_RESPONSE2,
NEGOTIATE_ALGORITHMS_REQUEST2, ALGORITHMS_RESPONSE2, CHALLENGE_REQUEST2,
CHALLENGE_AUTH_RESPONSE_WITHOUT_SIGNATURE2)
```

Where *Concatenate( )* is the standard concatenation function

- *GET_CAPABILITIES_REQUEST2* is the entire contents of the last successful GET_CAPABILITIES request message sent by the requestor. *CAPABILITIES_RESPONSE2* is the entire contents of the associated response message received by the requestor. Constructing M2 may require that the requestor preserve the contents of these prior messages.
- *NEGOTIATE_ALGORITHMS _REQUEST2* is the entire contents of the last successful NEGOTIATE_ALGORITHMS request message sent by the requestor. *ALGORITHMS_RESPONSE2* is the entire contents of the associated response message received by the requestor. Constructing M2 may require that the requestor preserve the contents of these prior messages.
- CHALLENGE_REQUEST is the entire contents of the CHALLENGE request message under consideration as sent by the requestor. CHALLENGE_AUTH_RESPONSE_WITHOUT_SIGNATURE is the entire CHALLENGE_AUTH response message without the signature field, as received by the requestor.

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

Step 2: The requestor shall perform

```
Verify(PK, Hash2(M2), Signature)
```

Where PK is the Public key associated with the leaf certificate of the responder with slot=Param1 of CHALLENGE request message.

Verify is the asymmetric verification algorithm the responder selected via the last ALGORITHMS response message as received by the requestor. Refer to *BaseAsymSel* or *ExtAsymSel* fields in Table 11.

Hash2 is the hashing algorithm the responder selected via the last ALGORITHMS response message sent as received by the requestor. Refer to *BaseHashSel* or *ExtHashSel* fields in Table 11.

8.3 Firmware measurement

This section describes request messages and response messages associated with endpoint firmware measurement. All request messages in this section shall be supported by an endpoint that returns MEAS_CAP=1 in CAPABILITIES Response. The high-level request-response flow and sequence for endpoint firmware measurement is shown in Figure 6.

If MEAS_FRESH_CAP bit in the CAPABILITIES response message returns 0, and the requestor requires fresh measurements, the responder must be reset prior to GET_MEASUREMENTS. The mechanisms employed for resetting the responder are outside the scope of this specification.

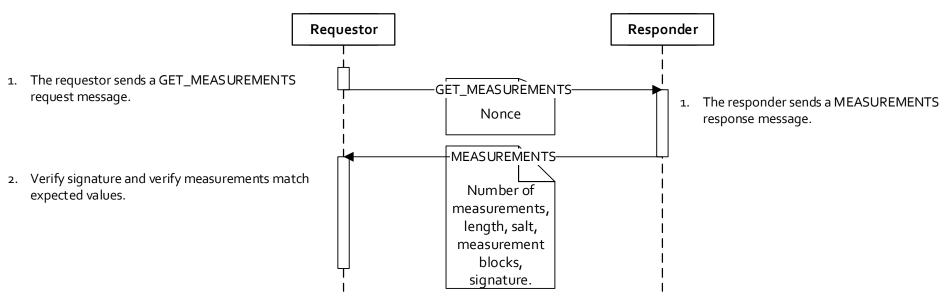


Figure 6 – Firmware measurement retrieval flow

8.3.1 GET_MEASUREMENTS request message and MEASUREMENTS response message

This request message shall be used to retrieve firmware measurements. The request message format is shown in Table 18 and the response message format is shown in Table 19.

Table 18 – GET_MEASUREMENTS request message

Offset	Field	Size in bytes	Value
0	*SPDMVersion*	1	V1.0 = 0x10
1	*RequestResponseCode*	1	0xE0 = GET_MEASUREMENTS
2	*Param1*	1	Measurement Request type 0: Single or All Measurements All other bits are reserved.
3	*Param2*	1	Measurement index Value of 0xFF return all Measurements.
4	*Nonce*	H	Random H-byte nonce chosen by the authentication initiator. H is the size of the hashing algorithm that the responder selected via ALGORITHMS response message.

Table 19 – Successful MEASUREMENTS response message

Offset	Field	Size in bytes	Value
0	*SPDMVersion*	1	V1.0 = 0x10
1	*RequestResponseCode*	1	0x60 = MEASUREMENTS
2	*Param1*	1	When the requested Measurement index is 0, this parameter returns the total number of Measurement indices on the device; otherwise reserved.
3	*Param2*	1	Reserved
4	*NumberOfBlocks*	1	N
8	*MeasurementRecord*	L=N*(H+4)	Concatenation of all Measurement Blocks that correspond to Measurement Request type and Measurement Index input values. Measurement Block structure is defined in section 8.3.2.
8+L	*Salt*	H	H bytes of arbitrary salt chosen by the Responder. H is the size of the hashing algorithm output the responder selected via ALGORITHMS response message.
8+L+H	*Signature*	S	Signature of the GET_MEASUREMENTS Request and MEASUREMENTS 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 ALGORITHMS response message to the requestor and S is the size of that asymmetric signing algorithm output.

8.3.2 Measurement block

Each Measurement block defined in the MEASUREMENTS response message shall contain a 4-byte descriptor (offsets 0-3), followed by the Measurement Data corresponding to a particular *Measurement Index* and *Measurement Type*. The blocks will be ordered by *Index*.

The format for a measurement block is shown in Table 20.

Table 20 – Measurement block definition

Offset	Field	Size in bytes	Value
0	*Index*	1	For *MeasurementType*=0-3, Index represents the Firmware stage and incrementing of Index represents bootstrapping of firmware stages. For example, index 0 firmware measures index 1 firmware, and so on.
1	*MeasurementType*	1	0: immutable ROM 1: mutable firmware 2: hardware configuration, e.g., straps, debug modes 3: firmware configuration, e.g., configurable firmware policy All other values reserved
2	*MeasurementSpecification*	1	0: DMTF All other bits reserved
3	*Reserved*	1	Reserved.
8	*Measurement*	M	This field contains M bytes of cryptographic hash measurement value. The length M is derived from the measurement hash algorithm returned in ALGORITHMS response message.

8.3.3 Signature Generation

Symbols ending with the number 1 represent the messages as observed by the responder.

Step 1: The responder shall construct L1

```
L1 = Concatenate(GET_MEASUREMENTS_REQUEST1, MEASUREMENTS_RESPONSE_WITHOUT_SIGNATURE1)
```

Where *Concatenate ( )* is the standard concatenation function

- *GET_ MEASUREMENTS _REQUEST1* is the entire MEASUREMENTS request message under consideration, as seen by the responder.
- *MEASUREMENTS _RESPONSE_WITHOUT_SIGNATURE1* is the entire *MEASUREMENTS* response message without the signature bytes, as sent by the responder.

Step 2: The responder shall generate

```
Signature = Sign(SK, Hash1(L1))
```

Where

Sign is the asymmetric signing algorithm the responder selected via the last ALGORITHMS response message sent by the responder. Refer to *BaseAsymSel* or *ExtAsymSel* fields in Table 11.

Hash1 is the hashing algorithm the responder selected via the last ALGORITHMS response message sent by the responder. Refer to *BaseHashSel* or *ExtHashSel* fields in Table 11.

SK = the private Key associated with the responder's Slot 0 leaf certificate.

8.3.4 Signature Verification

Symbols ending with the number 2 represent the messages as observed by the requestor.

Step1: The requestor shall create L2 as

```
L2= Concatenate(GET_MEASUREMENTS_REQUEST2, MEASUREMENTS_RESPONSE_WITHOUT_SIGNATURE2)
```

Where *Concatenate ( )* is the standard concatenation function

- *GET_ MEASUREMENTS _REQUEST2* is the entire contents of the MEASUREMENTS request message under consideration, as sent by the requestor.
- *MEASUREMENTS _RESPONSE_WITHOUT_SIGNATURE2* is the entire contents of the *MEASUREMENTS* response message without the signature bytes, as received by the requestor.

Step 2: The requestor shall perform

```
Verify(PK, Hash2(L2), Signature)
```

Where PK is the Public key associated with the slot 0 certificate of the responder. PK is extracted from the CERTIFIATES response.

Verify is the asymmetric verification algorithm the responder selected via the last ALGORITHMS response message as received by the requestor. Refer to *BaseAsymSel* or *ExtAsymSel* fields in Table 11.

Hash2 is the hashing algorithm the responder selected via the last ALGORITHMS response message sent as received by the requestor. Refer to *BaseHashSel* or *ExtHashSel* fields in Table 11.

8.4 ERROR response message

For a SPDM operation resulting in an error, the endpoint responding to the request message shall use the ERROR response message. The ERROR Response format is shown in Table 21 and the detailed error code, error data and extended error data are shown in Table 22.

Table 21 – ERROR response message

Offset	Field	Size in bytes	Value
0	*SPDMVersion*	1	V1.0 = 0x10
1	*RequestResponseCode*	1	0x7F = ERROR
2	*Param1*	1	Error Code. See Table 22
3	*Param2*	1	Error Data. See Table 22
4	*ExtendedErrorData*	0-32	Optional Extended Data. See Table 22

853 **Table 22 – Error Code and Error Data**

Error Code	Value	Description	Error Data	Extended Error Data
Reserved	0x00	Reserved	Reserved	Reserved
InvalidRequest	0x01	One or more Request fields are invalid	0x00	No extended error data is provided.
Reserved	0x02	Reserved.	Reserved	Reserved
Busy	0x03	The endpoint cannot respond now, but may be able to respond in the future	0x00	No extended error data is provided.
UnexpectedRequest	0x04	The endpoint received an unexpected request message. For example, CHALLENGE prior to NEGOTIATE_ALGORITHMS.	0x00	No extended error data is provided.
Unspecified	04h	Unspecified error occurred.	00h	No extended error data is provided.
Uninitialized	05h	Command received without session initialization.	00h	No extended error data is provided.
Reserved	05-0x3F	Reserved	Reserved	No extended error data is provided.
RequestedInfoTooLong	40h	The requested data cannot be sent in one response message	Returns length of the extended data field=4.	Maximum size supported (4 bytes)
MajorVersionMismatch	41h	Requested SPDM major Version is not supported.	Returns length of the extended data field=2.	16 bit bitmap representing all the SPDM major versions supported by the responder.
ResponseNotReady	42h	The response message is not ready. Requestor may ask for the response by sending RESPOND_IF_READY request message until the timeout CT is reached.	Returns length of the extended data field=4.	See Table 23
Reserved	43h-CFh	Reserved	Reserved.	Reserved
Reserved for other standards	D0h-EFh	Reserved for other standards	Reserved for other standard.	Reserved for other standards
Vendor Defined	F0h-FFh	Vendor defined	Vendor defined	Vendor defined

Table 23 –ResponseNotReady Extended Error Data

Offset	Field	Size in bytes	Value
0	*RecommendedDelay*	1	Time duration in uS for which the responder should wait before issuing RESPOND_IF_READY. It is expressed in logarithmic (base 2) scale. E.g. the raw value 8 indicates requestor should wait for 2^8=512 uS.
1	*Request Code*	1	The request code that triggered this response.
2	Token	1	The opaque handle that the requestor shall pass in with the RESPOND_IF_READY request message.
3	Reserved	1	Reserved

8.5 RESPOND_IF_READY request message

This request message shall be used to 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 with ERROR response, set ErrorCode=*ResponseNotReady* and return the same Token as the previous *ResponseNotReady* response.

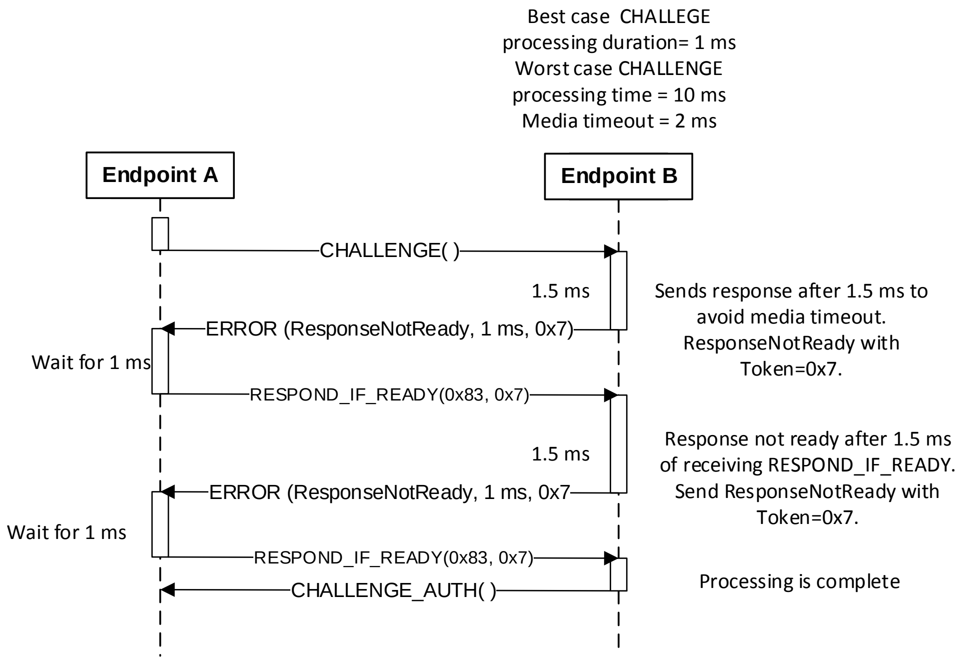


Figure 7 – RESPOND_IF_READY flow leading to completion

The request message format is shown in Table 18.

Table 24 – RESPOND_IF_READY request message

Offset	Field	Size in bytes	Value
0	*SPDMVersion*	1	V1.0 = 0x10
1	*RequestResponse Code*	1	0xFF = RESPOND_IF_READY
2	*Param1*	1	The original request code that triggered the *ResponseNotReady* error code response. Shall match the *Request Code* returned as part of the *ResponseNotReady* extended error data.
3	*Param2*	1	The token that was returned as part of the *ResponseNotReady* extended error data.

9 SPDM messaging control and discovery examples

9.1 Negotiating base hashing algorithms

This section illustrates how two endpoints negotiate base hashing algorithm under different scenarios.

In Example 1, endpoint A issues NEGOTIATE_ALGORITHMS request message and endpoint B selects an algorithm that both endpoints are capable of.

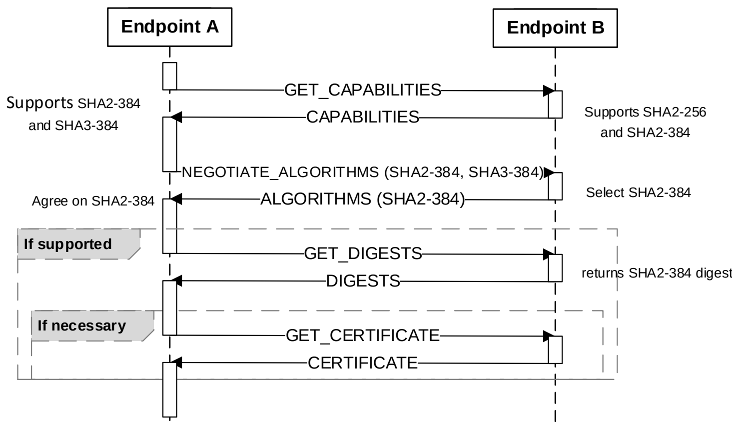


Figure 8 – Hashing Algorithm Selection: Example 1

In example 2, both endpoints issue NEGOTIATE_ALGORITHMS request message at about the same time. NEGOTIATE_ALGORITHMS request message from endpoint A is processed by endpoint B after endpoint B has sent out NEGOTIATE_ALGORITHMS request message. Both endpoints independently process the NEGOTIATE_ALGORITHMS request message and generate ALGORITHMS response message. Both endpoints are capable of SHA2-256 and SHA3-384, but both independently select SHA3-384.

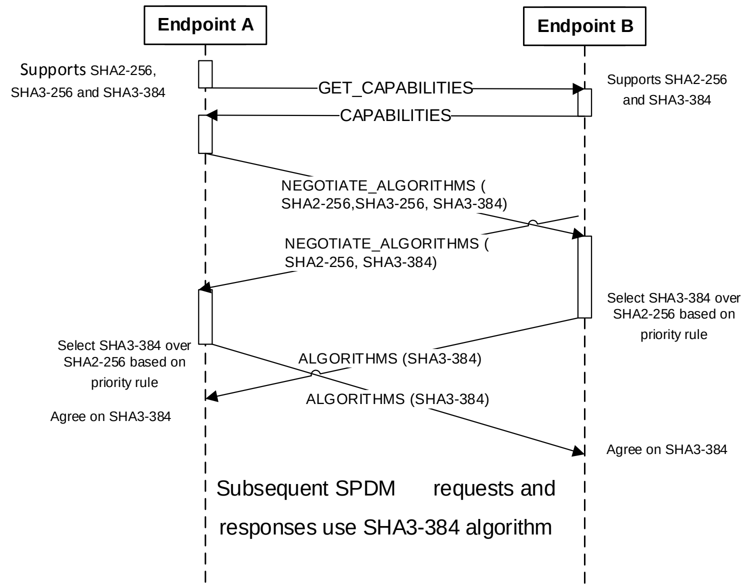


Figure 9 – Hashing Algorithm Selection: Example 2

In example 3, endpoint B does not support SPDM base algorithms and therefore is free to select extended algorithms. Both endpoints issue NEGOTIATE_ALGORITHMS request message at about the same time. NEGOTIATE_ALGORITHMS request message from endpoint A is processed by endpoint B after endpoint B has sent out NEGOTIATE_ALGORITHMS request message. Both endpoints independently process the NEGOTIATE_ALGORITHMS request message and generate ALGORITHMS response message. Both endpoints are capable of algorithms X and Y, but both independently select X based on the negotiation rules.

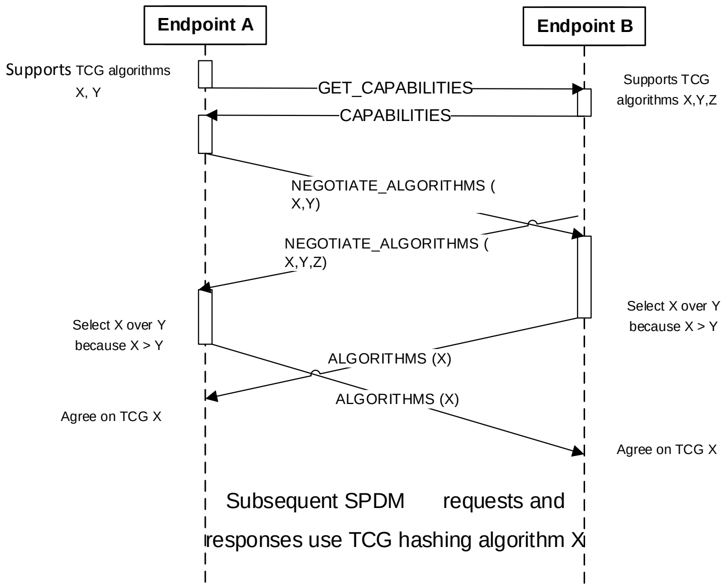


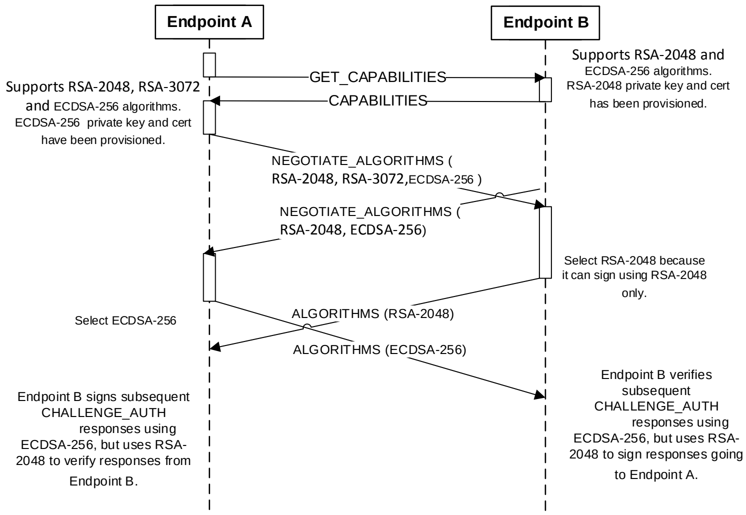
Figure 10 – Hashing Algorithm Selection: Example 3

9.2 Negotiating base asymmetric signature algorithms

This section illustrates how two endpoints negotiate asymmetric signature algorithms.

Endpoint A supports three algorithms, which means it can verify a message that is signed with either of the three algorithms. However, it holds private key and certificates corresponding to only one of these algorithms. That restricts endpoint A's signing capability to that single algorithm.

Unlike the hashing algorithm negotiation, the asymmetric signature algorithm selected in ALGORITHMS response message of endpoint A does not have to match the one selected by endpoint B. Endpoint A offers algorithms based on its verification capabilities and endpoint B selects an algorithm from that offer based on its signing capabilities. Similarly, endpoint B proposes algorithms based on its verification capabilities and endpoint A selects an algorithm from that proposal based on B's signing capabilities. The results of the negotiation are predictable even under scenarios where both endpoints issue NEGOTIATE_ALGORITHMS request message at about the same time.



911

912 **Figure 11 – Asymmetric Signature Algorithm Selection**

913

ANNEX A
(informative)

Change log

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

Bibliography

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