Authorization Specification

Version: 1.0.0WIP90

Information for Work-in-Progress version:

IMPORTANT: This document is not a standard. It does not necessarily reflect the views of 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 This document is available for public review and comment until superseded.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: https://www.dmtf.org/standards/feedback

Document Identifier: DSP0289

Date: 2025-04-03

Version History: https://www.dmtf.org/dsp/DSP0289

Supersedes: None

Document Class: Normative

Document Status: Work in Progress

Document Language: en-US

12 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 for uses consistent with this purpose, 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.

13 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 products, protocols, or testing procedures. DMTF shall have no liability to any party implementing such standards, 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patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a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.

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

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

CONTENTS

16 # 1 Foreword

17 The Security Protocols and Data Models (SPDM) Working Group prepared the *Authorization Specification* (DSP0289).

18 DMTF is a not-for-profit association of industry members that promotes enterprise and systems management and interoperability. For information about DMTF, visit dmtf.org.

19 ## 1.1 Acknowledgments

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

2 Introduction

The *Security Protocol and Data Model (SPDM) Authorization Specification* defines *messages*, data objects, and sequences for performing authorized message exchanges. The description of message exchanges includes *authorization* of messages, provisioning of authorization credentials and their policies, management of authorization state and other related capabilities.

2.1 Document conventions

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

2.1.1 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 DMTF.

Unless otherwise specified, field values marked as Reserved shall be written as zero (`0`), ignored when read, not modified, and not interpreted as an error if not zero.

2.1.2 Byte ordering

This section describes different byte ordering.

2.1.2.1 Default Byte Order

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).

2.1.2.2 Octet string byte order

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

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

37 #### 2.1.2.3 Signature byte order

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

39 ##### 2.1.2.3.1 ECDSA signatures byte order

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

41 ##### 2.1.2.3.2 SM2 signatures byte order

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

43 ### 2.1.3 Text or string encoding

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

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

46 Table 1 — "spdm" encoding example shows an encoding example of the string "spdm":

47 **Table 1 — "spdm" encoding example**

Offset	Character	Value
0	s	`0x73`
1	p	`0x70`

Offset	Character	Value
2	d	`0x64`
3	m	`0x6D`

48 ### 2.1.4 Other conventions

49 Unless otherwise specified, all figures are informative.

50

3 Scope

51 This specification describes how to use messages, data objects, and sequences to exchange authorized messages between two entities 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 authorization of arbitrary messages.

52 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 Normative references

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

- DMTF DSP0004, *Common Information Model (CIM) Metamodel*, https://www.dmtf.org/dsp/DSP0004
- DMTF DSP0223, *Generic Operations*, https://www.dmtf.org/dsp/DSP0223
- DMTF DSP0274, *Security Protocol and Data Model (SPDM) Specification*, https://www.dmtf.org/dsp/DSP0274
- DMTF DSP1001, *Management Profile Usage Guide*, https://www.dmtf.org/dsp/DSP1001
- *ISO/IEC Directives, Part 2, Principles and rules for the structure and drafting of ISO and IEC documents - 2021 (9th edition)*
- IETF RFC 4716, *The Secure Shell (SSH) Public Key File Format*, November 2006
- IETF RFC 7250, *Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*, June 2014
- *TCG Algorithm Registry, Family "2.0", Level 00 Revision 01.32*, June 25, 2020
- IETF RFC 8017, *PKCS #1: RSA Cryptography Specifications Version 2.2*, November, 2016
- IETF RFC 8032, *Edwards-Curve Digital Signature Algorithm (EdDSA)*, January 2017
- IETF RFC 8998, *ShangMi (SM) Cipher Suites for TLS 1.3*, March 2021
- GB/T 32918.1-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 1: General*, August 2016
- GB/T 32918.2-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 2: Digital signature algorithm*, August 2016
- GB/T 32918.3-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 3: Key exchange protocol*, August 2016
- GB/T 32918.4-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 4: Public key encryption algorithm*, August 2016
- GB/T 32918.5-2016, *Information security technology—Public key cryptographic algorithm SM2 based on elliptic curves—Part 5: Parameter definition*, August 2016
- GB/T 32905-2016, *Information security technology—SM3 cryptographic hash algorithm*, August 2016
- GB/T 32907-2016, *Information security technology—SM4 block cipher algorithm*, August 2016
- **ECDSA**
  - Section 6, The Elliptic Curve Digital Signature Algorithm (ECDSA) in FIPS PUB 186-5 Digital Signature Standard (DSS)
  - NIST SP 800-186 Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters
  - IETF RFC 6979, Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA), August 2013

- **SHA2-256**, **SHA2-384**, and **SHA2-512**
  - FIPS PUB 180-4 Secure Hash Standard (SHS)
- **SHA3-256**, **SHA3-384**, and **SHA3-512**
  - FIPS PUB 202 SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions
- **ASCII — ISO/IEC 646:1991**, 09/1991
- 64-bit CRC
  - Annex B, Standard ECMA-182, December 1992

5 Terms and definitions

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

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

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

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

The terms that DSP0004, DSP0223, DSP0274, and DSP1001 define also apply to this document.

This specification uses these terms:

Term	Definition
Authorization	Process of determining whether an entity has the privilege to perform a protected action.
Authorization Initiator	A logical entity that triggers the process of granting permission or approval for accessing a protected resource. An Authorization initiator can have an associated Credential ID depending on the type of message it sends.
Authorization message	Unit of communication when using messages defined in this specification.
Endpoint	Logical entity that communicates with other endpoints over one or more transport protocols.
Byte	Eight-bit quantity. Also known as an *octet*.
Credential	A piece of information used to verify an entities identity, such as an asymmetric public key.
Message	See Authorization message.
Protected Resource	A software or hardware resource that requires authorization to be operated upon.
User	An Authorization Initiator that is not an SPDM endpoint of the corresponding SPDM session. Additionally, a User is identified by their Credential ID.
Authorization target	A logical entity that determines if the Authorization Initiator has the permission(s) and privilege level(s) to access the protected resource.

Term	Definition
Authorization session	An SPDM session whose privilege levels have been escalated on behalf of either a User or SPDM endpoint.
User-Specific Authorization Session	An Authorization session that is escalated specifically on behalf of a specific User.
Authorization message payload	Portion of the message body of an authorization message. This portion of the message is separate from those fields and elements that identify the authorization request and response codes and reserved fields.
Concurrent SPDM session	Simultaneous or parallel SPDM sessions between an Authorization Initiator and an Authorization Target.
Owner	The user or consumer of the Authorization target operating in a data center environment and who is either in physical possession or is a tenant of the Authorization target. Examples of an Owner are the data center administrators, cloud providers, tenants of Infrastructure-as-a-Service or equivalent services, typically, offered by cloud providers. These Owners are generally not considered part of the supply chain such as a distributor, reseller, vendor, silicon manufacturer, OEM or ODM.

91 # 6 Symbols and abbreviated terms

92 The abbreviations that DSP0004, DSP0223, and DSP1001 define apply to this document.

93 The following additional abbreviations are used in this document.

Abbreviation	Definition
AODS	Authorization ODS
AUTH	Authorization
ODS	Opaque Data Structure
SEAP	SPDM Endpoint Authorization Process
SPDM	Security Protocol and Data Model
USAP	User-Specific Authorization Process
USAS	User-Specific Authorization Session
VDM	Vendor-Defined Messages

7 Notations

The authorization specification uses the following notations:

Notation	Description
`Concatenate()`	The concatenation function `Concatenate(a, b, ..., z)`, where the first entry occupies the least-significant bits and the last entry occupies the most-significant bits.
`M:N`	In field descriptions, this notation typically represents 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. The highest offset is on the right.
`[4]`	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.
`[M:N]`	A range of bit offsets where M is greater than or equal to N. The most significant bit is on the left, and the least significant bit is on the right.
`1b`	A lowercase `b` after a number consisting of `0`s and `1`s indicates that the number is in binary format.
`0x12A`	Hexadecimal, as indicated by the leading `0x`.
`N+`	Variable-length byte range that starts at byte offset N.

`[${message_name}].${field_name}`
 	
or
 	
`[${message_name}].${field_name}`/`${field_name0}`/.../`${field_nameN}`
	Used to indicate a field in a message.
	• `${message_name}` is the name of the request or response message.
	• `${field_name}` is the name of the field in the request or response message. An asterisk (`*`) instead of a field name means all fields in that message except for any conditional fields that are empty.
	• One or more optional forward slash character (`/`) can follow to indicate hierarchy of field names similar to a directory path in many Operating Systems

Notation	Description
`LenX`	This notation is used only in tables and indicate the length of the corresponding field only for that table. The value `X` can be a number greater than 0 especially if more than one of this notation is used in the same table for multiple fields. This notation is not used outside of a table and `LenX` in one table has no relationship for the same `LenX` in a different table.

99 # 8 Authorization architecture

100 This authorization architecture serves as a foundation for managing access to a protected resource on an endpoint. The message exchanges defined by this specification can be exchanged between two SPDM endpoints within an SPDM session. The messages are defined in a generic fashion that allows them to be communicated across different physical mediums and over different transport protocols.

101 ## 8.1 Architecture overview

102 The specification-defined message exchanges enable an entity to:

103 • Discover capabilities related to authorization in an endpoint.
104 • Discover and securely provision credentials and their policies into an endpoint.
105 • Securely manage endpoint state related to authorization.
106 • Authorize access to protected resource in an endpoint.

107 These capabilities are built on top of well-known and established security practices across the computing industry. The following clauses provide further details of the message exchanges related to authorization.

108 ## 8.2 Authorization version

109 The `AuthVersion` field in the `SELECT_AUTH_VERSION` message shall indicate the version of the Authorization specification that the format of an Authorization message adheres to.

110 For example, if the version of this specification is 1.2, the value of `AuthVersion` is `0x12`, which also corresponds to an `Authorization Major Version` of 1 and an `Authorization Minor Version` of 2.

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

112 The `AuthVersion` for the version of this specification shall be `0x10`.

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

114 ## 8.3 Authorization flows

115 At a high level, the authorization flow involves these processes:

116 • Credential provisioning
117 • Runtime authorization

8.3.1 Credential provisioning overview

Credential provisioning is the process where an endpoint is securely equipped with credential. In the context of this specification, a credential consists of an asymmetric key pair. The specifics of the key generation are outside the scope of the specification. For an asymmetric credential, the public portion is provisioned into the endpoint and the private key is held securely by the Authorization Initiator. The credential is also associated with a policy that describes the privileges, scope of access, lifetime or other access related attributes, to a protected resource. The specification defines a set of messages by which credentials and their policies can be securely provisioned into an endpoint with protected resources, typically an SPDM endpoint.

8.3.2 Runtime authorization overview

Runtime authorization is the process by which an Authorization Initiator, typically an SPDM endpoint, interacts with another endpoint to gain access to a protected resource at runtime. The endpoints exchange messages defined in this specification to discover capabilities related to authorization such as supported cryptographic algorithms, number of provisioned credentials and other related information. To gain access to a protected resource, the endpoint with the protected resource challenges the Authorization Initiator, who signs the challenge along with a message to be authorized, with the private key that it holds. The signature is then verified, and the credential checked against its policy, to determine if the message has the required privileges or access to operate on the protected resource.

Note that the specification does not mandate an Authorization Initiator be an SPDM endpoint, however the interactions specified are between two SPDM endpoints. In cases where an Authorization Initiator is not an SPDM endpoint, it is expected that an SPDM endpoint acts as a proxy to the initiator to facilitate communication to the endpoint with the protected resource.

Figure 1 — Model with SPDM endpoint as Authorization Initiator shows a model where an SPDM endpoint acts as an Authorization Initiator. Figure 2 — Model with external Authorization Initiator with SPDM endpoint proxy shows a model where the Authorization Initiator is an entity that is not an SPDM endpoint, but communicates with the protected resource via a proxy SPDM endpoint.

Figure 1 — Model with SPDM endpoint as Authorization Initiator

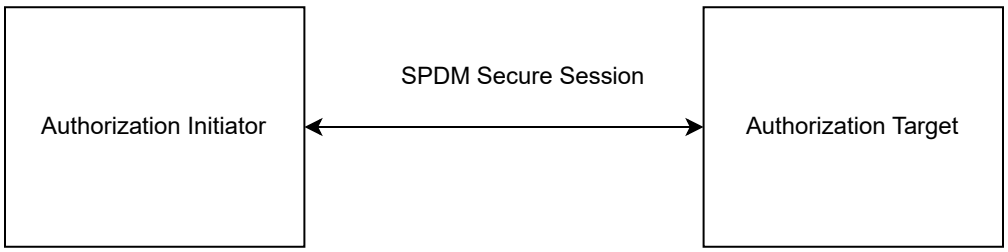
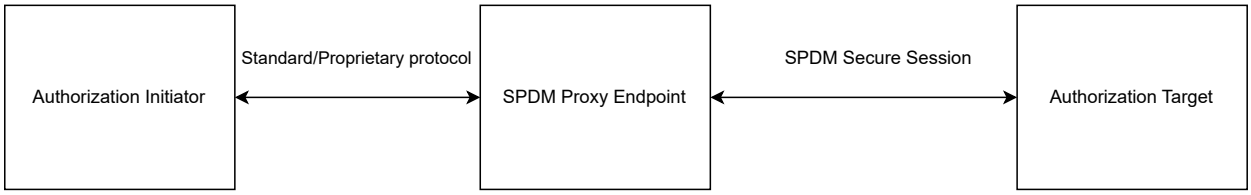


Figure 2 — Model with external Authorization Initiator with SPDM endpoint proxy



8.4 Credentials

In the context of this specification, a credential consists of an asymmetric key pair. See the Credential provisioning overview clause for how credentials are provisioned into an endpoint.

A credential is associated with an identifier, type, cryptographic algorithms, credential data and a *credential slot*. A credential slot is a logical location that holds a credential structure. An endpoint which supports provisioning credentials shall support a minimum of 8 credential slots. Credential slots are identified by the `CredentialID`. The mandatory 8 credential slots shall have `CredentialID` of 0 through 7 inclusive.

The `SET_CRED_ID_PARAMS` command shall be used to provision credentials into a credential slot and shall use the structure defined in credential structure. Credentials should be stored by the endpoint in integrity protected storage. An endpoint may use the credential structure as defined in the specification or use an implementation-specific data structure to store credentials.

Table 2 — Credential Structure describes the structure and format for a credential.

Table 2 — Credential Structure

Byte offset	Field	Size (bytes)	Description
0	CredentialID	2	A unique identifier to identify the credential and the credential slot. The value of 0xFFFF shall be reserved unless other parts of this specification defines the use for this value.
2	CredentialType	1	The type of the credential. `0x1` - Asymmetric Key. All other values - Reserved Shall be `0x1` for this version of the specification.

Byte offset	Field	Size (bytes)	Description
3	BaseAsymAlgo	`BaseAsymAlgoLen`	The format of this field shall be the format as Table 53.1.1 defines. When `CredentialType` is `0x1`, this field shall have exactly one bit set. Otherwise, no bits shall be set.
3 + `BaseAsymAlgoLen`	BaseHashAlgo	`BaseHashAlgoLen`	The format of this field shall be the format as Table 53.1.2 defines. When `CredentialType` is `0x1`, this field shall have exactly one bit set. Otherwise, no bits shall be set.
3 + `BaseAsymAlgoLen` + `BaseHashAlgoLen`	Reserved	4	Reserved.
7 + `BaseAsymAlgoLen` + `BaseHashAlgoLen`	CredentialDataSize	4	Size of Credential data in bytes.
11 + `BaseAsymAlgoLen` + `BaseHashAlgoLen`	CredentialData	`CredentialDataSize`	When `CredentialType` is `0x1`: • This field shall contain the public key in the `SubjectPublicKeyInfo` format specified by RFC 7250.

8.4.1 Credential Attributes

This section discusses various attributes that can associate with each credential ID. The Authorization initiator can use the `GET_CRED_ID_PARAMS` to see the supported attributes and their current state for the requested Credential ID. An Authorization target can support different attributes for different Credential IDs.

8.4.1.1 Locking and Unlocking Attributes

The locking attribute makes the credential and its associated policy immutable from modification from any requests for the given Credential ID regardless of authorization or the policy settings of the requesting Credential ID. Consequently, the unlocking attribute allows the credential and it associated policy modifiable according to policy of the requesting Credential ID. Furthermore, a Credential ID can only lock and unlock its own credentials and policy. In other words, no Credential ID can unlock or lock the credentials and the associated policies of other Credential IDs.

Lastly, a Credential ID can lock and unlock its own credential and policy if the `LockSelfPrivilege` bit is set in its own policy as Authorization policies section defines.

The Authorization initiator should exercise caution when deciding to lock the credential and associated policies of any Credential ID because recovery of locked credentials and their associated policies is outside the scope of this specification.

142 ### 8.4.2 Credential Changes Requirements

143 When changing the authorization policy for a given Credential ID, the new policy settings shall take effect immediately for that Credential ID. Consequently, the Authorization target shall terminate all active and saved Authorization processes using the given Credential ID.

144 ## 8.5 Authorization policies

145 All credentials shall be associated with an authorization policy. A credential shall not be usable for authorization without an associated policy. A policy shall be associated with a credential using the `SET_AUTH_POLICY` command. Policies should be stored by the endpoint in integrity protected storage. An endpoint may use the Policy List structure as defined in the specification or use an implementation-specific data structure to store authorization policies.

146 Table 3 — Policy List describes the structure and format for a list of policies.

147 **Table 3 — Policy List**

Byte Offset	Field	Size (bytes)	Description
0	CredentialID	2	A unique identifier to identify the credential and the credential slot.
2	NumPolicies	2	Shall be the number of policies listed in the `Policies` field. The value of this field shall be at least one.
4	Policies	Variable	List of policies as defined by Table 4 — Policy Structure.

148 Table 4 — Policy Structure describes the structure and format for a policy.

149 **Table 4 — Policy Structure**

Byte Offset	Field	Size (bytes)	Description
0	PolicyOwnerID	`LenSVH`	This field shall indicate the owner of the policy. The format of this field shall be the same as the `SVH` as SPDM defines.
`LenSVH`	PolicyLen	2	Shall be the length of `Policy`.

Byte Offset	Field	Size (bytes)	Description
2 + `LenSVH`	Policy	`PolicyLen`	This field indicates the policy as `PolicyOwnerID` defines. The `PolicyOwnerID` shall define the size and format of this field. If `PolicyOwnerID` is DSP0289 using DMTF-DSP as the `ID` in the `SVH`, the structure of this field is defined in Table 5 — DSP0289 Policy Structure.

150 Table 5 — DSP0289 Policy Structure describes the structure and format for DMTF defined policy.

151 **Table 5 — DSP0289 Policy Structure**

Byte Offset	Field	Size (bytes)	Description
0	PolicyType	2	`Policy Type` column in Table 7 — DSP0289 General Policy Definitions shall define the value for this field.
2	PolicyLen	2	Table 7 — DSP0289 General Policy Definitions shall define the value of this field corresponding to `PolicyType`.
4	PolicyValue	`PolicyLen`	Table 7 — DSP0289 General Policy Definitions shall define the value of this field corresponding to `PolicyType`.

152 ### 8.5.1 DSP0289 Authorization Policy

153 This section defines the privileges for commands, actions and other resources that this specification defines. Each credential ID has an associated policy. An Authorization initiator uses `SET_AUTH_POLICY` command to change the policy associated with the Credential ID provided in the request.

154 This section uses the term, "given Credential ID" to refer to the Credential ID used in many scenarios. In general there are two types of credential IDs: the Credential ID populated in the Credential ID field, if present, of an Authorization request message and the requesting Credential ID of a message. These two credential IDs are not always the same for a message. When authorizing a message, the given Credential ID is the Credential ID of the Authorization initiator of the corresponding message. After authorization succeeds and when fulfilling the request of an Authorization request message with a Credential ID field present, the term, given Credential ID, refers to the Credential ID populated in the Credential ID field of the corresponding request message.

155 The tables in this section are structured into different field types:

156 • Privilege. A privilege field type is a single bit where setting a single bit grants the ability to perform the corresponding action and clearing the bit revokes the ability to perform the corresponding action.

157 • Allowable. An allowable is a field consisting of one or more bits where setting one or more bits allows the use of one or more characteristics (usually configuration parameters) associated with that field.

158 Lastly, all Credential IDs can modify their own credential ID parameters limited by their associated authorization policy. All Credential IDs can retrieve their own authorization policy or revoke their own privileges for all fields of Privilege field type.

159 Table 6 — DSP0289 Policy Types lists all the policies specific to this specification. The values in the **Policy Type** column shall map to the `PolicyType` as Table 5 defines.

160 **Table 6 — DSP0289 Policy Types**

Policy Type	Policy Name	Description
0	Reserved	Reserved
1	GeneralPolicy	This policy type governs the possible actions an Authorization initiator can perform that are specific to this specification. The format and size of `PolicyValue` shall be the format and size as Table 7 — DSP0289 General Policy Definitions defines.

161 Table 7 — DSP0289 General Policy Definitions defines the credentials policies for the resources (for examples, commands, actions and more) that this specification defines.

162 **Table 7 — DSP0289 General Policy Definitions**

Byte Offset	Field	Size (bytes)	Field Type	Description
0	AllowedBaseAlgo	`BaseAsymAlgoLen`	Allowable	The format of this field shall be the format as Table 53.1.2 defines. This field reflects the allowed base algorithms the given Credential ID can use. If a bit is set, the given Credential ID shall be capable of utilizing the corresponding algorithm when `CredentialType` is 1. If a bit is not set, the given Credential ID shall be prohibited from utilizing the corresponding algorithm. At least one bit should be set. If no bits are set, then `CredentialType = 1` cannot be used.
`BaseAsymAlgoLen`	AllowedBaseHashAlgo	`BaseHashAlgoLen`	Allowable	The format of this field shall be the format as Table 53.1.2 defines. This field reflects the allowed base hash algorithms the given Credential ID can use. If a bit is set, the given Credential ID shall be capable of utilizing the corresponding hash when `CredentialType` is 1. If a bit is not set, the given Credential ID shall be prohibited from utilizing the corresponding hash. At least one bit should be set. If no bits are set, then `CredentialType = 1` cannot be used.

Byte Offset	Field	Size (bytes)	Field Type	Description
`BaseAsymAlgoLen` + `BaseHashAlgoLen`	CredentialPrivileges	2		The format of this field shall be the format as Table 8 — DSP0289 Authorization policy Bits Definitions defines.
2 + `BaseAsymAlgoLen` + `BaseHashAlgoLen`	AuthProcessPrivileges	1		The format of this field shall be the format as Table 9 — Authorization Process Policy Bits Definition defines. At least one bit should be set for the Authorization target to authorize any messages for the given Credential ID. Thus, when no bits are set, the Authorization target cannot authorize any messages for the given Credential ID which effectively disables the use of the given Credential ID.

163 Table 8 — DSP0289 Authorization policy Bits Definition defines the credentials provisioning policies.

164 **Table 8 — DSP0289 Authorization policy Bits Definition**

Byte Offset	Bit Offset	Field	Field Type	Description
0	0	ModifyCredentialInfoPrivilege	Privilege	If this bit is set, the given Credential ID shall be capable of modifying credential ID parameters of other Credential IDs through `SET_CRED_ID_PARAMS` request.
0	1	QueryCredentialInfoPrivilege	Privilege	If this bit is set, the given Credential ID shall be capable of retrieving credential ID parameters of other Credential IDs through `GET_CRED_ID_PARAMS` request.
0	2	GrantOtherPolicyPrivilege	Privilege	If this bit is set, the given Credential ID shall be capable of only granting privilege for all fields of Privilege field type for other Credential IDs through the `SET_AUTH_POLICY` request. Also, setting this bit also allows the given Credential ID to modify all fields of Allowable field type in any manner. If this bit is set, the `QueryPolicyPrivilege` shall also be set.
0	3	RevokeOtherPolicyPrivilege	Privilege	If this bit is set, the given Credential ID shall be capable of only revoking privileges for all fields of Privilege field type for other Credential IDs through the `SET_AUTH_POLICY` request. If this bit is set, the `QueryPolicyPrivilege` shall also be set.
0	4	QueryPolicyPrivilege	Privilege	If this bit is set, the given Credential ID shall be capable of retrieving authorization policy of other Credential IDs through `GET_AUTH_POLICY` request.
0	5	ResetToDefaultsPrivilege	Privilege	If this bit is set, the given Credential ID shall be capable of resetting the Authorization target back to default values and behavior using the `AUTH_RESET_TO_DEFAULT` request.

Byte Offset	Bit Offset	Field	Field Type	Description
0	6	LockSelfPrivilege	Privilege	When this bit is set, the given Credential ID shall be capable of locking or unlocking its own Credential parameters and its policy.
0	7	RetrieveAuthProcListPrivilege	Privilege	When this bit is set, the given Credential ID shall be capable of retrieving authorization process information of other credential IDs using `GET_AUTH_PROCESSES` request.
1	0	KillAuthProcListPrivilege	Privilege	When this bit is set, the given Credential ID shall be capable of terminating an authorization process of another Credential ID except for locked Credential IDs using the `KILL_AUTH_PROCESS` request.
1	[7:1]	Reserved	Reserved	Reserved.

165 Table 9 — Authorization Process Policy Bits Definition defines the authorization process policies.

166 **Table 9 — DSP0289 Authorization Process Policy Bits Definition**

Byte Offset	Bit Offset	Field	Field Type	Description
0	0	PrivilegeSEAP	Privilege	If this bit is set, the given Credential ID shall be capable of invoking the SEAP process as an Authorization initiator.
0	1	PrivilegeUSAP	Privilege	If this bit is set, the given Credential ID shall be capable of being a user in the USAP Process.
0	2	PrivilegePersistUSAS	Privilege	If this bit is set, the given Credential ID shall be capable of persisting its own USAS as USAP continuation defines. If this bit is set, the `PrivilegeUSAP` bit shall also be set.
0	[7:3]	Reserved	Reserved	Reserved.

167 #### 8.5.1.1 DSP0289 Authorization Policy Changes Requirements

168 When changing the authorization policy for a given Credential ID, the new policy settings shall take effect immediately for that Credential ID. The authorization target should enforce the new policy in the least invasive manner possible. For example, if the new settings grant or revoke a privilege in `ModifyCredentialInfoPrivilege` field, the Authorization target can apply the new settings to incoming messages without ending an active Authorization process. In another example, if a bit is cleared in `AllowedBaseAlgo` and an active Authorization process is using the corresponding asymmetric algorithm, then the Authorization target will have to fail authorization for all messages requiring authorization for the affected credential ID, except when the affected Credential ID changes its own credential ID parameters to comply with the new policy.

169 For some policy changes, there are some specific requirements. If a new policy clears a bit in an Allowable field type and the current credential ID parameters associated with that Credential ID uses the corresponding bit, the Authorization target shall still allow the Authorization initiator to use the existing credential ID parameters to change

the parameters to comply with the new policy through `SET_CRED_ID_PARAMS` and `GET_CRED_ID_PARAMS` while failing authorization for all other messages requiring authorization.

8.5.1.2 DSP0289 Additional Authorization Policy Requirements

An Authorization initiator should initially configure the authorization policy for a given Credential ID using the `SET_AUTH_POLICY` request before initially setting the credential ID parameters via `SET_CRED_ID_PARAMS` request for the same Credential ID.

8.5.2 Policy Attributes

This section describes attributes associated with policies. The Authorization initiator can use the `GET_CRED_ID_PARAMS` to see the supported attributes and their current state for the requested Credential ID. An Authorization target can support different attributes for different Credential IDs.

See Locking and Unlocking Attributes for attribute details applicable to policy.

8.6 Initial Provisioning

Initial provisioning covers provisioning requirements needed by entities in the supply chain and the Owner of the Authorization target. Provisioning is the setting of persistent authorization data such as credential ID parameters and associated policies.

The Authorization initiator can discover the device provisioning state by issuing an `GET_AUTH_CAPABILITIES` request and checking the `DeviceProvisioningState` field in the response.

8.6.1 Supply Chain Provisioning

As the Authorization target traverses the many entities involved in the manufacturing and distribution of the Authorization target, wholly called the supply chain, each entity may need to provision one or more Credential ID with their credentials and associated policies for many scenarios such as in-the-field debugging or return merchandise authorization. Details of these scenarios are outside the scope of this specification.

When the supply chain entity provisions a credential ID, that entity should utilize the highest numerically available and lockable Credential ID that the Authorization target supports. When the supply chain entity completes provisioning, that entity can decide to lock the provisioning for its Credential IDs so that it is only modifiable by that supply chain entity, itself. If the supply chain entity does not lock its provisioning, the Owner can modify those credentials and policy associated with that Credential ID.

The supply chain shall not issue `TAKE_OWNERSHIP` request because this can prevent the Owner from completing their provisioning.

8.6.2 Default State

The default state is the state of all persistent authorization data, including credentials and their associated policies of

all Credential IDs, after the supply chain completes their provisioning as Supply Chain Provisioning section defines but before the Owner takes possession of the Authorization target. The default state expects supply chain entities to lock their provisioning of credentials and associated policies of their selected Credential IDs. The default state uses a locked provisioning as an indicator of those Credential IDs provisioned by the supply chain that the supply chain does not want the Owner to modify.

184 The default values of all authorization data, including credentials and their associated policies, are determined by the supply chain and thus, are outside the scope of this specification.

185 8.6.3 Default State and Additional Supply Chain Requirements

186 This section defines requirements for the Authorization target in the default state and the state of the Authorization target as it traverses the supply chain.

187 While the Authorization target is in the default state or as it traverses through the supply chain, messages, including messages from other protocols or from entities other than the Authorization initiator, can still flow to the Authorization target over multiple transports. To ensure proper setup of the Authorization target and its protected resources, the Authorization target shall fail authorization of all messages requiring authorization except for these conditions:

188 • Authorization target shall verify authorization for these messages:
189 ◦ All messages requiring authorization that retrieves or modifies protected resources associated with the locked Credential ID.
190 ◦ `SET_CRED_ID_PARAMS` when locking or unlocking Credential IDs.
191 ◦ `TAKE_OWNERSHIP` request message.
192 • Authorization target shall bypass authorization verification for Authorization messages described in Credential provisioning section, Authorization policy provisioning and management and `AUTH_RESET_TO_DEFAULT` for unlocked credential IDs. The Authorization target shall not require an Authorization process to occur. In other words, the Authorization target fulfills the request without credentials and if no other errors occurs.

193 8.6.4 Taking Ownership

194 Taking ownership is the Owner performing its initial provisioning of the Authorization target. Taking ownership is important to ensure proper operation of the Authorization target in the operational environment of the Owner.

195 While the Authorization target is in the default state, an Authorization initiator can modify both the credential ID parameters and authorization policy of all unlocked credential IDs without credentials and as many times as the Owner needs as the Default State and Additional Supply Chain Requirement section defines. Once the Owner finishes its initial provisioning, the Authorization initiator shall issue `TAKE_OWNERSHIP` request to exit the default state and enter operational state where authorization is fully enforced accordingly for all messages.

196 The Owner should check provisioning of all Credential IDs to ensure they are provisioned as expected before sending the `TAKE_OWNERSHIP` request.

197 Lastly, a Credential ID can return an Authorization target back to the default state using the `AUTH_RESET_TO_DEFAULT` request if its authorization policy permits.

198 ### 8.6.5 Other Provisioning Considerations

199 This section discusses general provisioning considerations or requirements.

200 Provisioning of credentials and its associated policies in the default state or throughout the Supply chain should only be done in a trusted environment (such as a secure production sandbox environment or secure manufacturing). After taking ownership, an Owner can provision in a trusted environment or use a credential already provisioned to authorize provisioning of other credential ID parameters or their associated policies in an untrusted environment.

201 During and after initial provisioning, the Supply chain and the Owner can configure one or more Credential IDs to have the highest privilege levels or disperse privileges across two or more Credential IDs. Furthermore, the Owner can configure privileges in such a way that significantly disables operation of the Authorization target and recovery from such a state is outside the scope of this specification.

202 ## 8.7 Discovery

203 This section describes the methodology to discover support information of an SPDM endpoint as an Authorization target. The discovery process has two phases: an announcement phase followed by the Discover-Select Flow phase.

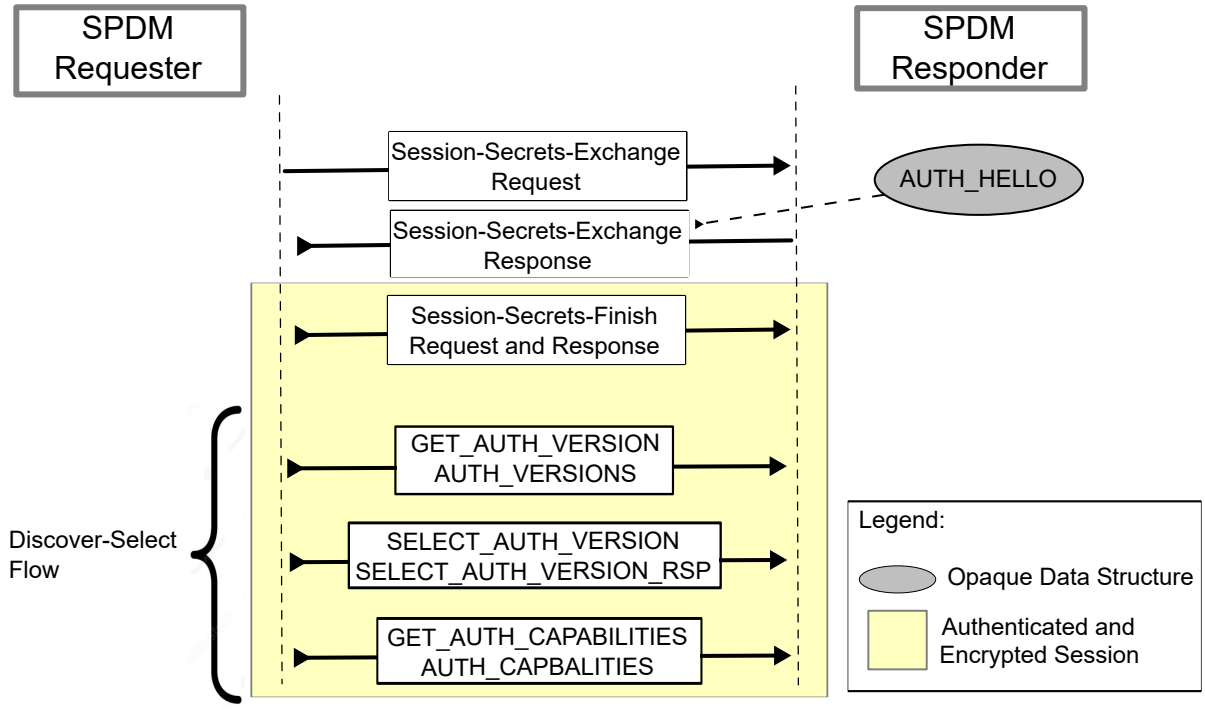
204 In the announcement phase, an Authorization target announces itself at the start of a session. If an SPDM requester is an Authorization target, the SPDM requester shall populate the `AUTH_HELLO` AODS in the Session-Secrets-Exchange request. Likewise, if an SPDM responder is an Authorization target, the SPDM responder shall populate the `AUTH_HELLO` AODS in the Session-Secrets-Exchange response.

205 The next phase is the Discover-Select Flow phase and this phase only occurs in the Application phase of an SPDM session. If the Authorization Initiator receives an `AUTH_HELLO` AODS, the Authorization Initiator can begin this phase by issuing the `GET_AUTH_VERSION` message, followed by the `SELECT_AUTH_VERSION` and ending with `GET_AUTH_CAPABILITIES`. The Authorization Initiator can issue these three requests in any order as well. The Discover-Select Flow phase does not need to occur or even complete for every session. However, the Authorization Initiator should complete this phase at least once with the corresponding Authorization target per SPDM connection.

206 Figure 3 — Most Common Discovery Phase illustrates the most common discovery methodology for an SPDM responder that is an Authorization target.

207 **Figure 3 — Most Common Discovery Phase**

208



209

8.8 Authorization Process

210 The authorization process is the process by which an Authorization target grants or denies access to a Protected resource based on policy.

211 Prior to the Authorization process, the Authorization target should have credentials and policy provisioned appropriate to its usage model. Otherwise, the Authorization target may inappropriately grant or deny access. See Credential provisioning and Authorization policy provisioning and management for details .

212 To properly setup for the execution of the authorization process, an Authorization Initiator shall successfully establish an SPDM secure session as DSP0274 defines or use an already established SPDM secure session.

213 The Authorization process establishes an authorization session and allows for establishing different types of authorization sessions. The different authorization processes this specification supports are these:

214 • User-Specific Authorization Process

215 • SPDM Endpoint Authorization Process

216

8.8.1 User-Specific Authorization Process

217 The User-Specific Authorization process occurs completely within an SPDM session. This process establishes an authorization session bound to the user. Thus, one or more User-specific authorization sessions can occur simultaneously within an SPDM session and the Authorization session identifier shall be the credential ID of the corresponding User.

218 The USAP starts with the `GET_AUTH_VERSION` to query for the supported version, followed by the `SELECT_AUTH_VERSION` to select the version to be used for subsequent messages and then followed by the `GET_AUTH_CAPABILITIES` to obtain the supported capabilities of the Authorization target. The Authorization Initiator can skip `GET_AUTH_VERSION` or `GET_AUTH_CAPABILITIES` if it already knows the list of supported versions and/or Capabilities beforehand, such as from a prior SPDM session or from an earlier request in the same session. The Authorization Initiator can skip `SELECT_AUTH_VERSION` if it just wants to use the highest version supported by the Authorization Target. The Authorization Initiator should send `GET_AUTH_VERSION` and `GET_AUTH_CAPABILITIES` before the first User-Specific Authorization session in each SPDM session to ensure the Authorization Initiator has the most up to date information.

219 To establish a User-Specific Authorization session, the Authorization Initiator shall send a `START_AUTH` request to the target with the User's corresponding information and the Authorization target shall respond with `START_AUTH_RSP` for a successful response. This request and response pair is important for these reasons:

220 • It elevates the privilege level of the SPDM secure session for that specific User. This portion of an SPDM session is called an Authorization session.

221 • It initializes critical cryptographic parameters for all messages requiring authorization in the corresponding SPDM session and corresponding User. Messages that traverse the corresponding SPDM session can be messages of any protocol and not restricted to SPDM or Authorization messages.

222 • The message format of all messages requiring authorization changes to accommodate authorization data for the corresponding User. The format for such messages is defined in the Authorization Record Section.

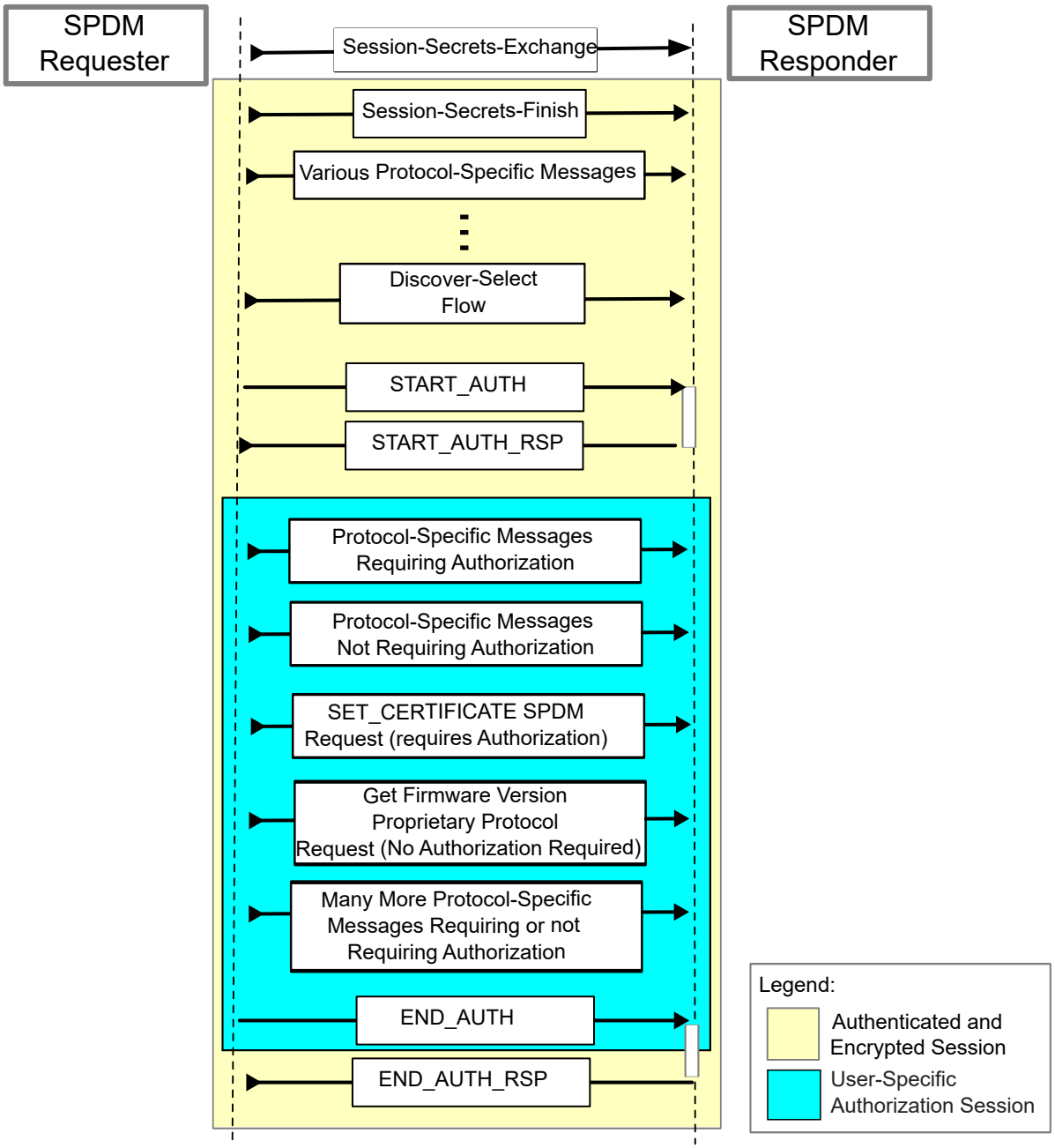
223 The successful completion of this request and response effectively establishes the Authorization session for the corresponding User. While the authorization session is active, messages requiring authorization shall contain authorization data, called Authorization tag, for the corresponding User. When the Authorization target receives a message from any protocol in the corresponding SPDM session, the Authorization target shall determine if the message requires authorization or not regardless of whether or not the message contains an Authorization tag. If a message requires authorization, the Authorization target shall validate the authorization tag according to the provisioned credentials and associated policies and the User associated with the corresponding Authorization session. Upon successful validation of the Authorization tag, the Authorization target shall process the message accordingly. If a message requiring authorization does not contain an Authorization tag or the validation of the Authorization tag fails, the Authorization target shall either respond with an `AUTH_ERROR` message, the corresponding protocol-specific error or silently discard the message. Even in error scenarios, the Authorization target still processes the Authorization tag, if present, as USAP Authorization Record details. For messages that do not require authorization, the Authorization target can process the message according to the definitions of its respective protocol.

224 The User-Specific Authorization session shall terminate for the corresponding User when the Authorization target receives an `END_AUTH` request from the Authorization Initiator or the corresponding SPDM session terminates. The termination of the Authorization session restores an SPDM session to its original privilege level for that User. Additionally, the termination of a User-Specific Authorization session does not end the corresponding SPDM session. Lastly, the termination of a USAS does not terminate the processing of received messages to completion according to the definition of their respective protocol and this specification by the Authorization target.

225 Figure 4 — Authorization Process illustrates an example of the User-Specific Authorization process.

226 **Figure 4 — Authorization Process**

227



228 **8.8.1.1 General USAP error handling, requirement and notes**

229 A User is identified by its Credential ID. The `START_AUTH`, `END_AUTH` and the Authorization record contains the Credential ID of the user.

230 A User shall have only one Authorization session active at a time. Therefore, a `START_AUTH` request shall be prohibited for the same User when the User has a corresponding active User-Specific Authorization session. The

User-Specific Authorization shall be terminated before another `START_AUTH` request can be issued. The Authorization target shall respond with an `AUTH_ERROR` or silently discard the request if a `START_AUTH` is received for a User with a corresponding active User-Specific Authorization Session.

231 A User can repeat the User-Specific Authorization process as many time as it deems necessarily. However, the Authorization target can limit the number of simultaneous active User-Specific Authorization sessions for a given SPDM session.

232 If the Authorization target receives a message with an authorization tag but the message does not require an authorization tag, the Authorization target shall still process the Authorization tag as this specification defines.

233 8.8.1.2 USAP Continuation

234 USAP continuation allows a Credential ID to continue a prior USAP session from where it ended, if supported by the Authorization target as indicated by `PermPersistCap` or `ResetPersistCap`. USAP continuation is similar to save and load operations common in numerous consumer applications. To save the USAP session, the Authorization initiator sets the `[END_AUTH].Attributes/PersistMethod` as desired. To restore the USAP session, the Authorization initiator sets the `[START_AUTH].Attributes/Continue` accordingly. See Authorization process management section for more details.

235 On a request to persist, both the Authorization target and Authorization initiator shall persist the USAP information corresponding to the `END_AUTH` request. The USAP information shall be as follows:

236 • Authorization initiator nonce

237 • Authorization target nonce

238 • `SavedSequenceNumber`

239 • Credential ID

240 The `SavedSequenceNumber` shall be the last sequence number in the USAP Authorization Tag used plus 1. To ensure the correct sequence number is saved, the User should ensure completion of all messages containing an authorization tag before issuing the `END_AUTH` request for the USAS corresponding to that User.

241 Once a saved USAP is continued, the USAP process becomes active and is no longer a saved USAP. However, the Authorization target should wait for at least one successfully authorized message before erasing the saved USAP information from its persistent storage. See `[END_AUTH].Attributes/PersistMethod` for additional requirements.

242 Additionally, the Authorization target shall persist no more than one USAP per Credential ID at a time.

243 Finally, the `PrivilegePersistUSAS` privilege governs the ability of USAP continuation for the given Credential ID.

244 8.8.2 SPDM Endpoint Authorization Process

245 The SPDM Endpoint Authorization Process (SEAP) is a process that specifically authorizes an SPDM requester only or both SPDM endpoints in an SPDM secure session. If SEAP authorizes only the SPDM requester, then the SPDM requester plays the role of the Authorization Initiator. If SEAP authorizes both endpoints, then the SPDM requester and SPDM responder can play the role of either an Authorization Initiator or Authorization target at any time within the session.

246 SEAP requires mutual authentication. Mutual authentication can use certificates or just a raw public key.

247 SEAP is broken into two parts as Figure 5 illustrates. The first part occur during the Session handshake phase as SPDM defines. The second part occurs during the SPDM Application phase.

248 The first part of SEAP begins with a Session-Secrets-Exchange request. If an SPDM Requester wants to invoke this authorization process, the SPDM Requester shall add the `INVOKE_SEAP` data structure to the `OpaqueData` field of a Session-Secrets-Exchange request. Additionally, if the SPDM responder wants to send messages requiring authorization to the SPDM requester using SEAP in the same session, the SPDM responder shall also add the `INVOKE_SEAP` data structure to the `OpaqueData` field of the Session-Secret-Exchange response. Lastly, the SPDM endpoints shall populate all fields appropriately in a Session-Secrets-Exchange request and response message to perform mutual authentication.

249 The first part of SEAP ends with the Session-Secrets-Finish message exchange. If the SPDM Requester successfully authenticates and finds a matching Credential ID for the SPDM responder, the SPDM Requester shall populate the `SEAP_SUCCESS` data structure in the `OpaqueData` field of the Session-Secrets-Finish request. Likewise, if the SPDM responder successfully authenticates and finds a matching Credential ID for the SPDM requester, the SPDM responder shall populate the `SEAP_SUCCESS` data structure in the `OpaqueData` field of the Session-Secret-Finish response. Otherwise, if there is a failure or the `OpaqueData` field does not exist, the `SEAP_SUCCESS` data structure in either the request or the response depending of which endpoint failed shall be absent. A failure to the SEAP process does not end the SPDM session.

250 Before the second part of SEAP can begin, the Authorization Initiator should send `GET_AUTH_VERSION` to query for the supported version followed by the `SELECT_AUTH_VERSION` to select the version to be used for subsequent messages and then followed the `GET_AUTH_CAPABILITIES` to obtain the supported capabilities of the Authorization target. The Authorization Initiator can skip `GET_AUTH_VERSION` or `GET_AUTH_CAPABILITIES` if it already knows the list of supported versions and/or Capabilities beforehand, such as from a prior SPDM session or from an earlier request in the same session. The Authorization Initiator can skip `SELECT_AUTH_VERSION` if it just wants to use the highest version supported by the Authorization Target. The Authorization Initiator should send `GET_AUTH_VERSION` and `GET_AUTH_CAPABILITIES` before the second part of SEAP in each SPDM session to ensure the Authorization Initiator has the most up to date information. Additionally, the SPDM requester and the SPDM responder may not support the same versions or capabilities even though they can be both Authorization Initiators in the same session.

251 The second part of SEAP can begin anytime during in the SPDM application phase. Additionally, the second part of SEAP can occur as many times as needed in the corresponding SPDM session. To initiate the second part of SEAP, the Authorization Initiator shall send a `ELEVATE_PRIVILEGE` request and the Authorization target shall respond with `PRIVILEGE_ELEVATED` for a successful response. This request and response pair elevates the privilege level of the SPDM secure session for the Authorization Initiator for all subsequent messages until the privilege level is lowered. An Authorization target shall return an `AUTH_ERROR` if there is a failure in authorization during the first part of SEAP (that is, the `SEAP_SUCCESS` was absent for the corresponding Authorization Initiator).

252 This portion of an SPDM session is called an Authorization session. In SEAP, at most two Authorization sessions can occur at any time simultaneously in the corresponding SPDM session. One Authorization session would be for the SPDM Requester who is acting as an Authorization Initiator and the other Authorization session would be for the SPDM responder who is acting as an Authorization Initiator.

253 The successful completion of this request and response effectively establishes the Authorization session for the corresponding Authorization Initiator. In an Authorization session, when the Authorization target receives a message

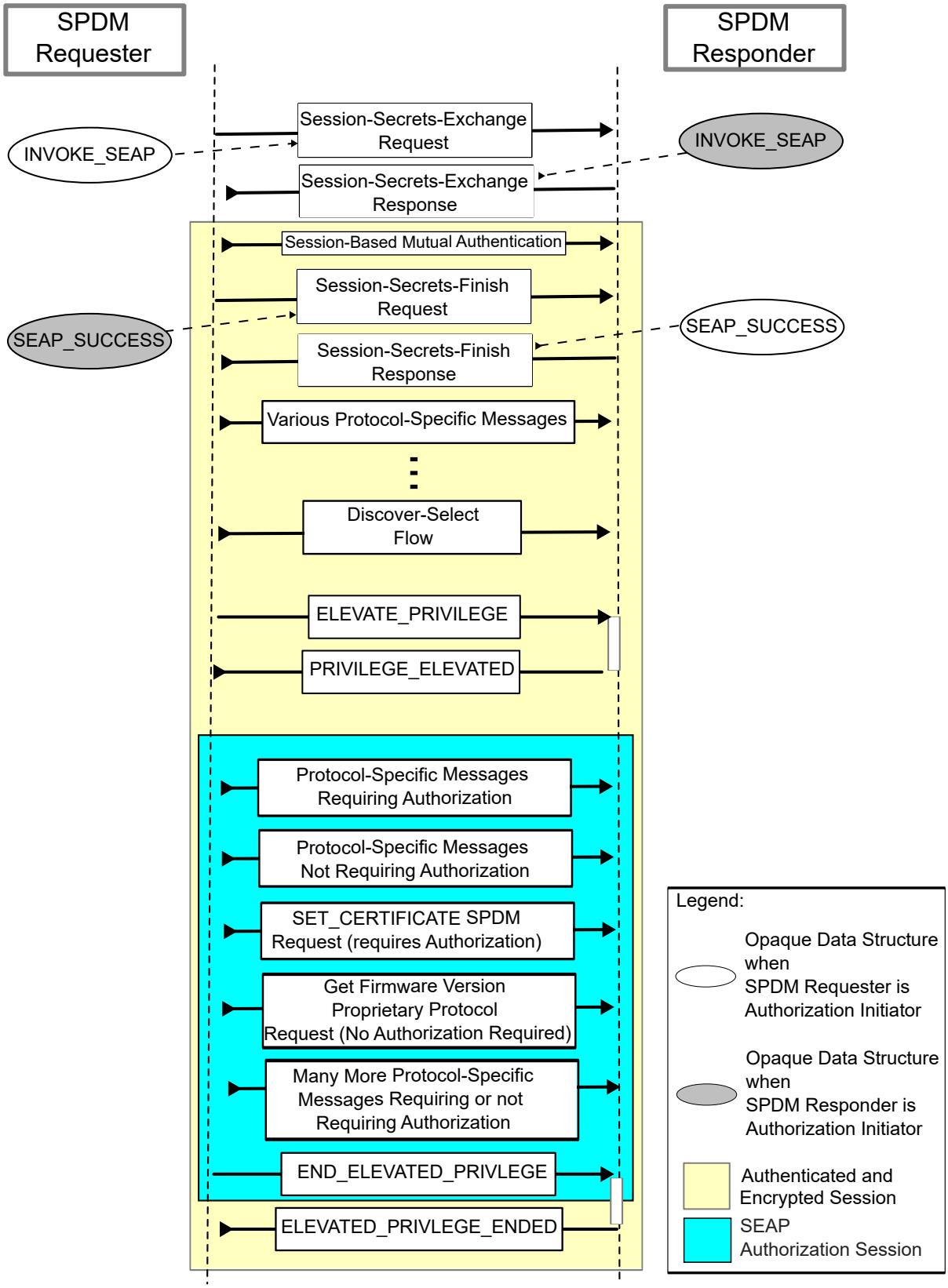
from any protocol in the corresponding SPDM session, the Authorization target shall determine if the message requires authorization or not. If a message requires authorization, the Authorization target shall validate the message according to the provisioned policies associated with the corresponding Authorization Initiator. Upon successful validation of the message, the Authorization target shall process the message accordingly. If the validation of the message fails, the Authorization target shall either respond with an `AUTH_ERROR` message, the corresponding protocol-specific error or silently discard the message. For messages that do not require authorization, the Authorization target can process the message accordingly.

254 The Authorization session shall terminate for the corresponding Authorization Initiator when the Authorization target receives an `END_ELEVATED_PRIVILEGE` request from the Authorization Initiator or the corresponding SPDM session terminates. The termination of the Authorization session restores an SPDM session to its original privilege level for that Authorization Initiator. Additionally, the termination of a SEAP Authorization session does not end the corresponding SPDM session. Lastly, the termination of a SEAP Authorization session does not terminate the processing of received messages to completion according to the definition of their respective protocol and this specification by the Authorization target.

255 Figure 5 — SPDM Endpoint Authorization Process (SEAP) illustrates the SPDM Endpoint Authorization Process (SEAP). Note, for simplicity, the figure does not illustrate all the required AODS during the SPDM handshake. See Authorization Opaque Data Structures for details on all AODS.

256 **Figure 5 — SPDM Endpoint Authorization Process (SEAP)**

257


SPDM Requester
SPDM Responder
INVOKE_SEAP
Session-Secrets-Exchange Request
Session-Secrets-Exchange Response
INVOKE_SEAP
Session-Based Mutual Authentication
SEAP_SUCCESS
Session-Secrets-Finish Request
Session-Secrets-Finish Response
SEAP_SUCCESS
Various Protocol-Specific Messages
Discover-Select Flow
ELEVATE_PRIVILEGE
PRIVILEGE_ELEVATED
Protocol-Specific Messages Requiring Authorization
Protocol-Specific Messages Not Requiring Authorization
SET_CERTIFICATE SPDM Request (requires Authorization)
Get Firmware Version Proprietary Protocol Request (No Authorization Required)
Many More Protocol-Specific Messages Requiring or not Requiring Authorization
END_ELEVATED_PRIVLEGE
ELEVATED_PRIVLEGE_ENDED
Legend:
Opaque Data Structure when SPDM Requester is Authorization Initiator
Opaque Data Structure when SPDM Responder is Authorization Initiator
Authenticated and Encrypted Session
SEAP Authorization Session

8.8.2.1 SEAP error handling, requirement and notes

If the `INVOKE_SEAP` data structure is absent in the Session-Secret-Exchange request, then the `SEAP_SUCCESS` shall be absent in the `OpaqueData` field of the corresponding Session-Secrets-Finish response. Likewise, if the `INVOKE_SEAP` data structure is absent in the Session-Secret-Exchange response, then the `SEAP_SUCCESS` shall be absent in the `OpaqueData` field of the corresponding Session-Secrets-Finish request.

If SEAP uses SPDM version 1.3 or earlier, then `SEAP_SUCCESS` cannot be supported because there is no `OpaqueData` Field in the Session-Secret-Finish message. Thus, if the first part of SEAP fails, the Authorization target shall return an `AUTH_ERROR` using `ErrorCode` = `OperationFailed` for the `ELEVATE_PRIVILEGE` request in all versions of SPDM.

If an SPDM session uses SEAP, then that session cannot use USAP because it is not possible to differentiate the Authorization Initiator of a message requiring authorization especially when an authorization tag is not present. Specifically, if an Authorization initiator invokes SEAP, then the Authorization target shall prohibit the use of USAP in the corresponding SPDM session.

If `INVOKE_SEAP` is present in a Session-Secret-Exchange message, it shall only be present exactly once.

8.8.3 Terminating Authorization Process

There are two types of Authorization process termination. The first type is natural termination where by the Authorization initiator sends an end Authorization process request such as `END_AUTH` to the Authorization target. The other type is a forced termination. Both types achieve the same effect except for the case where Authorization process is preserved. An Authorization process can only be preserved through natural termination.

Other parts of this specification use forced termination.

In cases that do not preserve an Authorization process or kills a saved Authorization process, terminating an Authorization process destroys all metadata (for example, nonce, sequence numbers) associated with that Authorization process and returns the associated Credential ID to an unprivileged state where all messages requiring authorization fails authorization checks. The affected Credential ID can start a new Authorization process thereafter.

8.8.4 Other error handling, requirements and notes

When an Authorization session is not active in an SPDM session for a given User or Authorization Initiator, the processing of messages, regardless of whether or not they require authorization, is outside the scope of this specification but likely follows the definitions of its respective protocol. From an authorization perspective, this specification, however, recommends one of these three options:

- The Authorization target uses another form of authorization, which is outside the scope of this specification.
- Respond with an `AUTH_ERROR` response for all messages requiring authorization.
- Silently discard the message.

The Authorization sessions does not limit the types of messages that can traverse an SPDM session but rather enables explicit validation of authority for all messages according to provisioned credentials and policies.

Furthermore, this specification strongly recommends that messages requiring authorization be denied access for Users or Authorization entities outside of an Authorization session.

273 The use of the same Credential ID across multiple SPDM sessions can occur at any time, including simultaneously. The Authorization target and Authorization Initiator shall ensure that authorization data associated with a given Credential ID is bound to their respective SPDM session and Authorization session. In other words, the authorization data cannot intermix with another session. For example, the sequence number, the authorization tag or nonce that is bound to session 45 cannot be used in session 99.

274

8.9 Authorization Record

275 An authorization record is a wrapper structure that carries authorization information and the message, itself, for messages requiring authorization. The authorization record provides the transport a protocol-agnostic way to send and receive messages requiring authorization.

276 Table 10 — Authorization Record format shows the Authorization Record format:

277 **Table 10 — Authorization Record format**

Byte offset	Field	Size (bytes)	Description
0	AuthRecordType	1	Specifies the record type. The values in this field shall be the values defined in Table 10.1 Authorization record types.
1	Reserved	1	Reserved.
2	GenericPayloadLen	4	Length, in bytes, of `GenericPayload`.
6	GenericPayload	`GenericPayloadLen`	The format of this field shall be the format specified by the `AuthRecordType`.

278 Table 10.1 shows the supported Authorization Record Type

279 **Table 10.1 — Authorization Record Types**

Value	Description
0	Authorization message. The `GenericPayload` field shall contain an Authorization messages that this specification defines and does not require authorization. The size and format of this field shall be the size and format of the specific Authorization message.
1	Encapsulated message requiring authorization. The `GenericPayload` field shall contain data in the format specified by Type 1 Message Requiring Authorization Record Format.
2	Record Error. The `GenericPayload` field shall contain data in the format specified by Table 10.3333 — Type 2 Authorization Record Failure. The Authorization target can use this record type to convey errors associated with the Authorization record or AUTH record over SPDM VDM. It can also silently discard the Authorization record or AUTH record over SPDM VDM.

Value	Description
All other values	Reserved

280 ## 8.9.1 Authorization Record on the Transport

281 While the Authorization Record can traverse any transport, there are some requirements the transport should define. The transport shall define at least one mechanism to indicate the presence and absence of the Authorization record, so that it can be identified and forwarded to the authorization logic for further processing. This can be done through a single bit indicating presence or simply stating the Authorization record is always present, for example. The transport can also choose to use the mechanism defined in Authorization Record over SPDM Vendor Defined Messages to transmit the Authorization record since this may help prevent significant modifications to the transport.

282 The transport can provide additional requirements, changes or constraints, if any.

283 ## 8.9.2 Authorization Types

284 This section defines the format and requirements for all Authorization record types.

285 ### 8.9.2.1 Authorization Record in Authorization Process

286 This section describes further details on the Authorization record specific to each Authorization process.

287 #### 8.9.2.1.1 USAP Authorization Record

288 This section defines requirements for all messages requiring authorization in USAP.

289 The Authorization record shall be present for all messages requiring authorization.

290 The Authorization Record shall be transmitted exclusively from the authorization initiator to the authorization target. Transmission in the opposite direction is prohibited.

291 For messages not requiring authorization in USAP, the transport can use the Authorization record. If the Authorization record is used for messages not requiring authorization, the `AuthRecordFlags[0]` bit should be set to zero.

292 Table 10.222222 — Type 1 Message Requiring Authorization Record format shows the format for the `GenericPayload` field when `AuthRecordType` is 1:

293 **Table 10.222222 — Type 1 Message Requiring Authorization Record format**

Byte offset	Field	Size (bytes)	Description
0	AuthRecID	4	This field indicates a unique number of for this Authorization record. The Authorization endpoints use this number for tracking and error handling purposes. The value of this field should increment by one. Values can repeat as long as the Authorization initiator ensures that the Authorization target finishes authorization checks on this process. The value, 0xFFFF_FFFF, shall not be used.
4	AuthTagLen	4	This field shall contain the length, in bytes, of `AuthTag`. The value of this field shall be greater than zero.
8	AuthTag	`AuthTagLen`	If present, this field shall contain the Authorization Tag for the `MsgToAuthPayload`.
8 + `AuthTagLen`	MsgToAuthPayloadLen	4	Shall be the length, in bytes, of `MsgToAuthPayload`. The value of this field shall be greater than zero.
12 + `AuthTagLen`	MsgToAuthPayload	`MsgToAuthPayloadLen`	Shall contain the message requiring authorization. The message can be a message of any protocol. The format and size of this field are specific to the message protocol. For Authorization messages, this field shall only contain Authorization requests. The size and format shall be the size and format of the respective Authorization request.

8.9.2.1.2 SEAP Authorization Record

The transport shall specify its use of an Authorization record.

8.9.2.2 Authorization Record Failures

The Authorization target can send an Authorization record with authorization tag verification failure type (Type 2) to indicate an authorization verification failure.

Table 10.3333 — Type 2 Authorization Record Failure

Byte offset	Field	Size (bytes)	Description
0	ErrorAuthRecID	4	Shall be the `AuthRecID` of the authorization record that contains the error. If the `AuthRecID` is not known, such as when a message requiring authorization does not have an Authorization tag, the value of this field shall be 0xFFFF_FFFF.
2	AuthRecErrorInfo	Len0	This field contains the error information. The format and size of this field shall be the same as `AUTH_ERROR` response. Note, Type 2 can only use certain types of `ErrorCode`s.

8.10 Authorization Tag

The authorization tag is the cryptographic data that accompanies a message that requires authorization. An authorization tag may or may not be present in every authorization process or every message. The Authorization record embeds the authorization tag. This section details the Authorization tag for each Authorization process.

Furthermore, Authorization tags only support asymmetric signature algorithms.

8.10.1 SEAP Authorization Tag

The authorization tag is not present in SEAP as SEAP Authorization Record discusses. The credential ID to use for SEAP shall be the one provided in the `INVOKE_SEAP` AODS.

8.10.2 USAP Authorization Tag

This section discusses details about the Authorization tag in a USAP.

In a User-specific authorization session, the Authorization tag identifies the user requesting authorization. Specifically, the authorization tag contains a credential ID that numerically identifies the User and verifiable cryptographic information that authenticates the user to ensure the message came from the corresponding User.

8.10.2.1 USAP Authorization Tag Format

The format and size for the `AuthTag` in the Authorization record shall be the format and size as Table 10.1111 defines.

Table 10.1111 shows the format for the USAP Authorization tag.

Table 10.1111 — Authorization Tag format

Byte offset	Field	Size (bytes)	Description
0	CredentialID	2	Shall be the credential ID of an active User-Specific Authorization session.
2	Signature	Len0	Shall be the signature of selected asymmetric algorithm associated with `CredentialID` as USAP Authorization Tag Signature Generation and Verification defines. The size of this field shall be the size of the selected signature associated with `CredentialID`.

The Authorization target shall use `CredentialID` to locate the credential to verify the authorization tag, if present in the authorization record.

8.10.2.2 USAP Authorization Tag Signature Generation and Verification

If the provisioned authorization tag cryptographic function for the correspond User is an asymmetric signature algorithm, then this section defines the operations associated with this algorithm.

314 The verifiable cryptographic information in an Authorization tag shall be a digital signature whose signature algorithm is the provisioned asymmetric signature algorithm corresponding to the User.

315 To compute the signature, first, the User shall create `AuthMsgBody` by concatenating the following fields in order:

316 1. The credential ID of the User.
317 2. The requester's nonce provided in the `START_AUTH` request.
318 3. The responder's nonce provided in the `START_AUTH_RSP` response.
319 4. The sequence number
320 5. The message body, which is the `MsgToAuthPayload` field of the Authorization Record

321 If `[START_AUTH].Attributes/Continue` is set, the sequence number shall start with `SavedSequenceNumber` as USAP continuation defines with the successful completion of `START_AUTH` request. Otherwise, the sequence number shall start at 1. Thereafter, the sequence number shall increment by one after each message requiring authorization and corresponding to the User. For the Authorization target, the sequence number shall increment by one after receiving a message containing an Authorization tag from the corresponding User regardless of whether the Authorization verification succeeds or fails.

322 The message body shall be all the bytes of the `MsgToAuthPayload` field of the Authorization Record. Because this specification regards the message body as opaque data, the message body shall have an octet string byte order.

323 The size of the sequence number shall be 32 bits. Once the sequence number exceeds the maximum value of 0xFFFF_FFFF, the User-Specific Authorization Session shall terminate.

324 Finally, the User shall compute `AuthMsgSignature` using this function and the corresponding selected asymmetric signature algorithm.

```
AuthMsgSignature = AuthSign(UserPrivKey, AuthMsgBody, context)
```

325 where:

326 - The `UserPrivKey` shall be the private key associated with the corresponding User.
327 - The `context` shall be the string "usap signing".

328 The `AuthMsgSignature` shall be the signature in an Authorization tag for the corresponding user and corresponding message.

329 Likewise, the Authorization target shall verify the message requiring the authorization through this method:

```
AuthValResult = AuthSigVerify(UserPublicKey, AuthSignature, AuthMsgBody, context)
```

330 where:

331 - The `UserPublicKey` shall be the public key associated with the User associated with the corresponding credential ID.
332 - The `AuthSignature` shall be the signature in the Authorization tag which accompanied the message.
333 - The `context` shall be the string "usap signing".

If `AuthValResult` is success, the Authorization tag validates successfully. Otherwise, it fails.

The message requiring authorization shall be successful if all the following conditions are met:

- The message contains an Authorization tag.
- The `AuthValResult` is success.
- The policy associated with the message grants the corresponding User access.

Otherwise, the message fails authorization.

9 Authorization messages

9.1 Authorization messages overview

Authorization messages are messages defined by this specification, that are sent between the Authorization Initiator and target and forms a request-response protocol. The following clauses describe the rules and requirements for the messaging protocol.

9.1.1 Bi-directional Authorization message processing

This clause describes the specifications and requirements for handling bi-directional and overlapping authorization request messages.

If an endpoint can act as both an Authorization Initiator and authorization target, it shall be able to send request messages and response messages independently.

When an SPDM endpoint acts as a proxy between an Authorization Initiator and an authorization target, how the proxy SPDM endpoint enforces the rules specified in the following clauses are outside the scope of this specification.

While the specification anticipates that, in common scenarios, an SPDM requester acts as the authorization initiator and an SPDM responder serves as the authorization target, this configuration is not mandated by the architecture. The following clause assumes that an SPDM endpoint is the Authorization Initiator.

9.1.2 Requirements for Authorization Initiators

An Authorization Initiator shall not have multiple outstanding requests to the same authorization target, within a single SPDM session. This restriction shall only apply to the messages defined by this specification. For messages defined by other protocols, the rules on multiple outstanding requests are outside the scope of this specification.

An outstanding request is a request where the request message has begun transmission, the corresponding response has not been fully received.

Within an SPDM session, if the Authorization Initiator has sent a request to an authorization target and wants to send a subsequent request to the same target, then the Authorization Initiator shall wait to send the subsequent request until after the Authorization Initiator completes one of the following actions:

- Receives the response from the authorization target for the outstanding request.
- Times out waiting for a response.
- Receives an indication from the transport layer that transmission of the request message failed.
- The Authorization Initiator encounters an internal error or Reset.

An Authorization Initiator might send simultaneous request messages to the same authorization targets across multiple SPDM sessions or to different authorization targets.

9.1.3 Requirements for Authorization Targets

An authorization target is not required to process more than one request message at a time, within a single SPDM session.

An authorization target that is not ready to accept a new request message shall either respond with an `AUTH_ERROR` message of `ErrorCode=Busy` or silently discard the request message.

If an authorization target supports authorization messages across concurrent SPDM session, a pending request in one session shall not affect pending requests in another session.

9.1.4 Authorization Messages bits-to-bytes mapping

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

Figure 6 — One-byte field bit map shows the one-byte field bit map:

Figure 6 — One-byte field bit map

Example:
A One-Byte Field Starting at Byte Offset 3

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

Figure 7 — Two-byte field bit map shows the two-byte field bit map:

Figure 7 — Two-byte field bit map

Example:
A Two-Byte Field Starting at Byte Offset 5

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

9.1.5 Version encoding

The `AuthVersion` field in the `SELECT_AUTH_VERSION` message 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 `AuthVersion` field in the `SELECT_AUTH_VERSION` message.	Protocol modification breaks backward compatibility.
Minor	Minor version field in the `AuthVersion` field in the `SELECT_AUTH_VERSION` message.	Protocol modification maintains backward compatibility.

371 EXAMPLE:

372 Version 3.7 → `0x37`

373 Version 1.0 → `0x10`

374 Version 1.2 → `0x12`

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

376 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_AUTH_VERSION`.

377 This specification considers two minor versions to be interoperable when it is possible for an implementation that is conformant to a higher minor version number to also communicate with an implementation that is conformant to a lower minor version number with minimal differences in operation. In such a case, the following rules apply:

378 • Both endpoints shall use the same lower version number in the `AuthVersion` field for all messages.

379 • Functionality shall be limited to what the lower minor version of the authorization specification defines.

380 • Computations and other operations between different minor versions of the authorization specification should remain the same, unless security issues of lower minor versions are fixed in higher minor versions and the fixes require change in computations or other operations. These differences are dependent on the value in the `AuthVersion` field in the message.

381 • In a newer minor version of the authorization specification, a given message can be longer, bit fields and enumerations can contain new values, and reserved fields can gain functionality. Existing numeric and bit fields retain their existing definitions. Also, Fields within a message may grow in length.

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

383 For details on the version agreement process, see GET_AUTH_VERSION request and AUTH_VERSION response messages and SELECT_AUTH_VERSION request and SELECT_AUTH_VERSION_RSP response message. The detailed version encoding that the `AUTH_VERSION` response message returns contains an additional byte that indicates specification bug fixes or development versions. See Table 17 — Successful AUTH_VERSION response message format.

9.1.6 Generic Authorization message format

Table 11 — Generic Authorization message field definitions defines the fields that constitute a generic authorization message, including the message header and payload:

Table 11 — Generic authorization message field definitions

Byte offset	Bit offset	Size (bits)	Field	Description
0	[7:0]	8	Request Response Code	Shall be the request message code or response code, which Table 12 — Authorization Message request codes and Table 13 — Authorization Message response codes enumerate. `0x00` through `0x7F` represent response codes and `0x80` through `0xFF` represent request codes. In request messages, this field is considered the request code. In response messages, this field is considered the response code.
1	[7:0]	8	Reserved	Reserved
2	See the description.	Variable	*Authorization message payload*	Shall be zero or more bytes that are specific to the `Request Response Code`.

9.2 Authorization message definitions

This section discusses all authorization request and response messages.

9.2.1 Authorization message request codes

Table 12 — Authorization message request codes defines the Authorization message request codes. The **Implementation requirement** column indicates requirements on the Requester.

The **Authorization requirements** column indicates whether or not the message requires authorization. If a value in this column is *Mandatory*, the Authorization target shall perform authorization checks for the corresponding request. If a value in this column is *None*, the Authorization target shall not perform authorization checks for the corresponding request. Finally, when the value in this column is *Conditional*, the section of this specification for the corresponding request details the requirements. If a request message fails authorization checks, the Authorization target shall respond with a `AUTH_ERROR` using `ErrorCode=AccessDenied`.

If an Authorization target receives an unsupported request, the Authorization target shall respond with an `AUTH_ERROR` using `ErrorCode = UnsupportedRequest`.

393 **Table 12 — Authorization message request codes**

Request	Code value	Implementation requirement	Authorization Requirements	Message format
GET_AUTH_VERSION	0x81	Required	None	Table 16 — GET_AUTH_VERSION request message format
SELECT_AUTH_VERSION	0x82	Required	None	Table 19 — SELECT_AUTH_VERSION request message format
SET_CRED_ID_PARAMS	0x83	Optional	Conditional	Table 25 — SET_CRED_ID_PARAMS request message format
GET_CRED_ID_PARAMS	0x84	Required	Conditional	Table 28 — GET_CRED_ID_PARAMS request message format
SET_AUTH_POLICY	0x85	Optional	Conditional	Table 30 — SET_AUTH_POLICY request message format
GET_AUTH_POLICY	0x86	Required	Conditional	Table 33 — GET_AUTH_POLICY request message format
START_AUTH	0x87	Optional	None	Table 35 — START_AUTH request message format
END_AUTH	0x88	Optional	None	Table 37 — END_AUTH request message format
ELEVATE_PRIVILEGE	0x89	Optional	None	Table 39 — ELEVATE_PRIVILEGE request message format
END_ELEVATED_PRIVILEGE	0x8A	Optional	None	Table 41 — END_ELEVATED_PRIVILEGE request message format
GET_AUTH_CAPABILITIES	0x8B	Required	None	Table 21 — GET_AUTH_CAPABILITIES request message format
AUTH_RESET_TO_DEFAULT	0x8C	Optional	Conditional	Table 43 — AUTH_RESET_TO_DEFAULT request message format
TAKE_OWNERSHIP	0x8D	Mandatory	Mandatory	Table 41.100 — TAKE_OWNERSHIP request message format

Request	Code value	Implementation requirement	Authorization Requirements	Message format
GET_AUTH_PROCESSES	0x8E	Mandatory	Mandatory	Table 1000 — GET_AUTH_PROCESSES request message format
KILL_AUTH_PROCESS	0x8F	Mandatory	Mandatory	Table 1003 — KILL_AUTH_PROCESS request message format
Reserved	All other values	Reserved	Reserved	Authorization implementations compatible with this version shall not use the reserved request codes.

394 ### 9.2.2 Authorization message response codes

395 The `Request Response Code` field in the Authorization response message shall specify the appropriate response code for a request.

396 On a successful completion of an authorization message request, the specified response message shall be returned. Upon an unsuccessful completion of an authorization command, the `AUTH_ERROR` response message should be returned.

397 Table 13 — Authorization message response codes defines the response codes for authorization messages. The **Implementation requirement** column indicates requirements on the Responder.

398 **Table 13 — Authorization message response codes**

Response	Code value	Implementation requirement	Message format
AUTH_VERSION	0x01	Required	Table 17 — Successful AUTH_VERSION response message format
SELECT_AUTH_VERSION_RSP	0x02	Required	Table 20 — Successful SELECT_AUTH_VERSION_RSP response message format
SET_CRED_ID_PARAMS_DONE	0x03	Optional	Table 27 — Successful SET_CRED_ID_PARAMS_DONE response message format
CRED_ID_PARAMS	0x04	Required	Table 29 — Successful CRED_ID_PARAMS response message format
SET_AUTH_POLICY_DONE	0x05	Optional	Table 32 — Successful SET_AUTH_POLICY_DONE response message format

Response	Code value	Implementation requirement	Message format
AUTH_POLICY	0x06	Required	Table 34 — Successful AUTH_POLICY response message format
START_AUTH_RSP	0x07	Optional	Table 36 — Successful START_AUTH_RSP response message format
END_AUTH_RSP	0x08	Optional	Table 38 — Successful END_AUTH_RSP response message format
PRIVILEGE_ELEVATED	0x09	Optional	Table 40 — Successful PRIVILEGE_ELEVATED response message format
ELEVATED_PRIVILEGE_ENDED	0x0A	Optional	Table 42 — Successful ELEVATED_PRIVILEGE_ENDED response message format
AUTH_CAPABILITIES	0x0B	Required	Table 22 — Successful AUTH_CAPABILITIES response message format
AUTH_DEFAULTS_APPLIED	0x0C	Optional	Table 46 — AUTH_DEFAULTS_APPLIED response message format
OWNERSHIP_TAKEN	0x0D	Mandatory	Table 41.101 — OWNERSHIP_TAKEN response message format
AUTH_PROCESSES	0x0E	Optional	Table 1001— AUTH_PROCESSES response message format
PROCESS_KILLED	0x0F	Optional	Table 1004— PROCESS_KILLED response message format
AUTH_ERROR	0x7F	Required	Table 14 — AUTH_ERROR response message format
Reserved	All other values	Reserved	Authorization implementations compatible with this version shall not use the reserved request codes.

9.2.3 Common Variable Names

This section defines some frequent variable names used in various Authorization messages. Table 13.100 — Common Variables used in Authorization Messages defines these variable names.

Table 13.100 — Common Variables used in Authorization Messages

Variable Names	Value
BaseAsymAlgoLen	Shall be 8.
BaseHashAlgoLen	Shall be 8.
LenSVH	Shall be the size of the `SVH` as SPDM defines.

9.2.4 Error handling

This section discusses general error handling for all authorization messages.

9.2.4.1 AUTH_ERROR response message

For an authorization request message that results in an error, the authorization target should send an `AUTH_ERROR` message to the Requester. The Authorization record also uses this response message for errors in the Authorization record, itself.

Table 14 — AUTH_ERROR response message format shows the `AUTH_ERROR` response format.

Table 15 — Error code and error data shows the detailed error code, error data, and extended error data. The **Layer** column indicates which layer can use the corresponding `ErrorCode`. A value of **M** in this column shall indicate the `ErrorCode` shall only be in a response to an Authorization request. A value of **R** shall indicate the `ErrorCode` shall only be in a Type 2 Authorization record. More than one value can be present for an `ErrorCode` in which case they are comma separated.

Table 14 — AUTH_ERROR response message format

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `AUTH_ERROR` in Table 13 — Authorization message response codes.
1	Reserved	1	Reserved.
2	ErrorCode	1	Shall be the ErrorCode. See Table 15 — Error code and error data.
3	ErrorData	0-32	Shall be the Error data. See Table 15 — Error code and error data.

Table 15 — Error code and error data

ErrorCode	Value	Layer	Description	Error data	ExtendedErrorData
Reserved	0x00	Reserved	Reserved.	Reserved	Reserved
InvalidRequest	0x01	M	One or more request fields are invalid	`0x00`	No extended error data is provided.
ResetRequired	0x02	M	The operation or request requires a reset to successfully complete.	`0x00`	No extended error data is provided.
Busy	0x03	M, R	The Authorization Initiator received the request message and the authorization target decided to ignore the request message, but might be able to process the request message if the request message is sent again in the future.	`0x00`	No extended error data is provided.
UnexpectedRequest	0x04	M	The authorization target received an unexpected request message.	`0x00`	No extended error data is provided.
Unspecified	0x05	M, R	Unspecified error occurred.	`0x00`	No extended error data is provided.
AccessDenied	0x06	R	Authorization checks failed.	`0x00`	No extended error data is provided.
OperationFailed	0x07	M	The request was valid but the requested operation failed.	`0x00`	No extended error data is provided.
VersionMismatch	0x08	M	Requested `AuthVersion` is not supported or is a different version from the selected version.	`0x00`	No extended error data is provided.
UnsupportedRequest	0x09	M	The `RequestResponseCode` in the request message is unsupported.	`RequestResponseCode` in the request message.	No extended error data is provided.

ErrorCode	Value	Layer	Description	Error data	ExtendedErrorData
InvalidRecord	0x0A	R	One or more fields in the Authorization record are invalid.	0x0	No extended error data is provided.
Reserved	All other values		Reserved.	Reserved	Reserved

9.2.5 Discovery message

Message in this section discover aspects of the Authorization target. These aspects provide basic information to understand support and establish basic communication parameters.

9.2.5.1 GET_AUTH_VERSION request and AUTH_VERSION response messages

This request message shall retrieve the authorization specification version of an endpoint. Table 16 — GET_AUTH_VERSION request message format shows the `GET_AUTH_VERSION` request message format and Table 17 — Successful AUTH_VERSION response message format shows the `AUTH_VERSION` response message format.

In all future authorization versions, the `GET_AUTH_VERSION` and `AUTH_VERSION` response messages will be backward compatible with all earlier versions.

The Authorization Initiator should begin the discovery process by sending a `GET_AUTH_VERSION` request message. It may skip this message if the information provided by the `AUTH_VERSION` response is known beforehand from a prior or concurrent SPDM session. All Authorization Targets shall always support the `GET_AUTH_VERSION` request message and provide an `AUTH_VERSION` response containing all supported versions, as Table 16 — GET_AUTH_VERSION request message format describes.

When `GET_AUTH_VERSION` is used, the Authorization Initiator should consult the `AUTH_VERSION` response to obtain information on a common supported version. The Authorization Initiator shall use one of the supported version in all future communication of other requests. The Authorization Initiator shall not issue other requests until it receives a successful `AUTH_VERSION` response and identifies a common version that both sides support. An Authorization Target shall not respond to the `GET_AUTH_VERSION` request message with an `AUTH_ERROR` message except for `ErrorCode`s specified in this clause. The selected version for communication with an authorization target shall be the version in the `AuthVersion` field of the `SELECT_AUTH_VERSION` Request message sent by the Authorization Initiator, if sent, otherwise shall be the highest version supported by the authorization target. If the Authorization Initiator uses a version other than the selected version in a Request, the Authorization Target should either return an `AUTH_ERROR` message of `ErrorCode=VersionMismatch` or silently discard the Request.

Table 16 — GET_AUTH_VERSION request message format shows the `GET_AUTH_VERSION` request message format:

Table 16 — GET_AUTH_VERSION request message format

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value in for `GET_AUTH_VERSION` in Table 12 — Authorization Message request codes.
1	Reserved	1	Reserved.

419 Table 17 — Successful AUTH_VERSION response message format shows the successful `AUTH_VERSION` response message format:

420 **Table 17 — Successful AUTH_VERSION response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `AUTH_VERSION` in Table 13 — Authorization Message response codes.
1	Reserved	1	Reserved.
2	VersionNumberEntryCount	1	Number of version entries present in this table (=n).
3	VersionNumberEntry1:n	2 * n	16-bit version entry. See Table 18 — VersionNumberEntry definition. Each entry should be unique.

421 Table 18 — VersionNumberEntry definition shows the `VersionNumberEntry` definition. See Version encoding for more details.

422 **Table 18 — VersionNumberEntry definition**

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

423 #### 9.2.5.2 SELECT_AUTH_VERSION request and SELECT_AUTH_VERSION_RSP response messages

424 The `SELECT_AUTH_VERSION` request should be used to specify the version of this specification that an authorization target shall use when interpreting request messages and providing response messages for authorization commands. The request and response parameters for this message are listed in Table 19 and Table 20. If the Authorization Initiator wants to specify a version, it shall send the `SELECT_AUTH_VERSION` request before any authorization messages other than `GET_AUTH_VERSION`. For a given SPDM session between an Authorization Initiator and authorization target, authorization target that supports multiple versions of the authorization specification but has not received a `SELECT_AUTH_VERSION` request shall interpret request messages and provide response messages according to the highest version it supports. The version selected using this request applies only to the SPDM session in which the message was sent and valid until the session terminates. If an Authorization Initiator uses concurrent SPDM sessions, this request should be sent in each SPDM session, if the highest supported version is not desired. The Authorization Initiator shall not send this request more than once within an SPDM session, and an Authorization Target shall respond with `AUTH_ERROR` and `ErrorCode=InvalidRequest` or silently discard the request, if it receives more than one `SELECT_AUTH_VERSION` in the SPDM session.

425 Table 19 — SELECT_AUTH_VERSION request message format shows the `SELECT_AUTH_VERSION` request message format:

426 **Table 19 — SELECT_AUTH_VERSION request message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `SELECT_AUTH_VERSION` in Table 12 — Authorization Message request codes.
1	Reserved	1	Reserved.
2	AuthVersion	1	The version that shall be used for all subsequent communication between the Authorization Initiator and target.

427 Table 20 — Successful SELECT_AUTH_VERSION_RSP response message format shows the successful `SELECT_AUTH_VERSION_RSP` response message format:

428 **Table 20 — Successful SELECT_AUTH_VERSION_RSP response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `SELECT_AUTH_VERSION_RSP` in Table 13 — Authorization Message response codes.
1	Reserved	1	Reserved.

429 #### 9.2.5.3 GET_AUTH_CAPABILITIES request and AUTH_CAPABILITIES response messages

430 The `GET_AUTH_CAPABILITIES` request and `AUTH_CAPABILITIES` response shall retrieve capability information from the

Authorization target. The request and response parameters for this message are listed in Table 19 and Table 20. While this request can be sent multiple times at any time, the request should be sent as the Discovery section describes. If the request is sent multiple times in the same SPDM session, the corresponding response shall be identical to the first.

431 Table 21 — GET_AUTH_CAPABILITIES request message format shows the `GET_AUTH_CAPABILITIES` request message format:

432 **Table 21 — GET_AUTH_CAPABILITIES request message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `GET_AUTH_CAPABILITIES` in Table 12 — Authorization Message request codes.
1	Reserved	1	Reserved.

433 Table 22 — Successful AUTH_CAPABILITIES response message format shows the successful `AUTH_CAPABILITIES` response message format:

434 **Table 22 — Successful AUTH_CAPABILITIES response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `AUTH_CAPABILITIES` in Table 13 — Authorization Message response codes.
1	Reserved	1	Reserved.
2	MessageCaps	2	The format of this field shall be the format as Table 23 — Message Supported Bit Definitions defines.
4	AuthProcessCaps	2	The format of this field shall be the format as Table 24 — Authorization Process Supported Bit Definitions defines.
6	DeviceProvisioningState	1	The format of this field shall be the format as Table 24444 — Device Provisioning State Values defines.

Byte offset	Field	Size (bytes)	Description
7	AuthRecordProcessTime	1	This field shall specify the additional amount of time a message of any protocol that is encapsulated in an Authorization record takes to process the Authorization record excluding the `MsgToAuthPayload` field. This time includes the time it takes to perform authorization verification. The time shall be calculated using this equation and shall be in units of milliseconds: $2^{\text{AuthRecordProcessTime}}$ milliseconds The value of this field shall not exceed 31. See Timing Requirements for additional requirements.
8	BaseAsymAlgoSupported	`BaseAsymAlgoLen`	If a bit is set, the Authorization target supports the corresponding asymmetric algorithm. Otherwise, the bit shall be clear. The format of this field shall be the format as Table 53.1.2 defines.
8 + `BaseAsymAlgoLen`	BaseHashAlgoSupported	`BaseHashAlgoLen`	If a bit is set, the Authorization target supports the corresponding hash algorithm. Otherwise, the bit shall be clear. The format of this field shall be the format as Table 53.1.2 defines.
8 + `BaseAsymAlgoLen` + `BaseHashAlgoLen`	SupportedPolicyOwnerIDCount	2	The value of this field shall be the number of policy owners in `SupportedPolicyOwnerIDList`. If the value of this field is zero, then the `SupportedPolicyOwnerIDList` field shall be absent.

Byte offset	Field	Size (bytes)	Description
10 + `BaseAsymAlgoLen` + `BaseHashAlgoLen`	SupportedPolicyOwnerIDList	Variable	This field summarizes the policies the Authorization target supports by only listing the policy owners (`PolicyOwnerID`). The format of this field shall be the concatenation of one or more `PolicyOwnerID` fields as Table 4 — Policy Structure defines for each policy the Authorization target supports. The number of `PolicyOwnerID` s in this list shall be the value in the `SupportedPolicyOwnerIDCount` field. If multiple policies share the same `PolicyOwnerID` , that `PolicyOwnerID` shall only be included once. Finally, this list shall be considered to be unordered. To retrieve more details of policy support, the Authorization initiator can use the `GET_AUTH_POLICY` and the corresponding response.

435 Table 23 — Message Supported Bit Definitions defines the messages the authorization endpoint supports.

436 **Table 23 — Message Supported Bit Definitions**

Byte Offset	Bit Offset	Field	Description
0	0	ChangeCredIDParamsCap	If the Authorization target supports `SET_CRED_ID_PARAMS_DONE` , then this bit shall be set. Otherwise, this bit shall not be set.
0	1	ChangeAuthPolicyCap	If the Authorization target supports `SET_AUTH_POLICY_DONE` , then this bit shall be set. Otherwise, this bit shall not be set.
0	2	AuthEventCap	If the Authorization target supports Authorization events as Authorization events define, then this bit shall be set.
0	3	AuthProcListCap	If the Authorization target supports `AUTH_PROCESSES` , then this bit shall be set. Otherwise, this bit shall not be set.
0	4	AuthProcKillCap	If the Authorization target supports `PROCESS_KILLED` , then this bit shall be set. Otherwise, this bit shall not be set. If this bit is set, the `AuthProcListCap` shall also be set.
0	[7:5]	Reserved	Reserved.
1	[7:0]	Reserved	Reserved

437 Table 24 — Authorization Process Supported Bit Definitions defines the messages the authorization endpoint supports.

438 **Table 24 — Authorization Process Supported Bit Definitions**

Byte Offset	Bit Offset	Field	Description
0	0	USAPcap	If the Authorization target supports USAP, then this bit shall be set. Otherwise, this bit shall not be set. If this bit is set, `START_AUTH_RSP`, `END_AUTH_RSP` response message shall be supported.
0	1	SEAPcap	If the Authorization target supports SEAP, then this bit shall be set. Otherwise, this bit shall not be set. If this bit is set, `PRIVILEGE_ELEVATED` and `ELEVATED_PRIVILEGE_ENDED` response messages shall be supported.
0	2	ResetPersistCap	If the Authorization target supports USAP continuation until device reset, this bit shall be set. Otherwise, this bit shall not be set. If `USAPcap` is not set, this bit shall not be set.
0	3	PermPersistCap	If the Authorization target supports USAP continuation across device reset, this bit shall be set. Otherwise, this bit shall not be set. If `USAPcap` is not set, this bit shall not be set.
0	[7:4]	Reserved	Reserved
1	[7:0]	Reserved	Reserved

439 **Table 24444 — Device Provisioning State Values**

Value	Name	Description
0	Unprovisioned	Device does not have any credentials provisioned.
1	DefaultState	Device has been provisioned with at least one credential in the supply chain but ownership has not been taken. See Default State for additional details.
2	Owned	Device has had ownership taken via `TAKE_OWNERSHIP`. See Taking Ownership for additional details.
3-255	Reserved	Reserved for future device states.

440 ## 9.2.6 Credential provisioning

441 ### 9.2.6.1 SET_CRED_ID_PARAMS request and SET_CRED_ID_PARAMS_DONE response messages

442 The `SET_CRED_ID_PARAMS` request shall be used to provision credentials into an authorization target, as described in the Credentials section. When `CredentialList` provides an invalid credential type, credential slot or algorithm, the authorization target shall respond with `AUTH_ERROR` and `ErrorCode=InvalidRequest`.

443 The Authorization Initiator shall use the `SetCredInfoOp` field to specify the operation for the request. An authorization target shall ensure that the operation is atomic, that is, the requested operation can successfully complete for all credentials in the `CredentialList`, and fail if that is not possible. When `CredentialList` provides an invalid

credential slot or policy, the authorization target shall respond with `AUTH_ERROR` and `ErrorCode=InvalidRequest`. When `SetCredInfoOp` is valid but authorization checks fails, the authorization target shall respond with `AUTH_ERROR` and `ErrorCode=AccessDenied`.

444 Table 25 — SET_CRED_ID_PARAMS request message format shows the `SET_CRED_ID_PARAMS` request message format:

445 **Table 25 — SET_CRED_ID_PARAMS request message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `SET_CRED_ID_PARAMS` in Table 12 — Authorization Message request codes.
1	Reserved	1	Reserved.
2	SetCredInfoOp	1	The field indicates the requested operation. The format of this field shall be the format as Table 26 defines.
3	CredParams	Variable	This field represents identity information associated with the given Credential ID. The format and size of this field shall be the same format and size as Table 2 — Credential Structure defines.

446 **Table 26 — Values for `SetCredInfoOp` field**

Value	Operation Name	Description
0	Reserved	Reserved
1	ParameterChange	Shall indicate an operation that modifies credential parameters associated with the given credential IDs.
2	Lock	Shall indicate an operation that locks the credential parameters and its authorization policy for the given Credential ID. The Authorization target shall only permit this operation if the `Lockable` credential attribute is set for the requested credential ID.
3	Unlock	Shall indicate an operation that unlocks the credential parameters and its authorization policy for the given Credential ID. The Authorization target shall only permit this operation if the `Unlockable` credential attribute is set for the requested credential ID.
All other values	Reserved	Reserved

447 Table 27 — Successful SET_CRED_ID_PARAMS_DONE response message format shows the successful `SET_CRED_ID_PARAMS_DONE` response message format:

448 **Table 27 — Successful SET_CRED_ID_PARAMS_DONE response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `SET_CRED_ID_PARAMS_DONE` in Table 13 — Authorization Message response codes.
1	Reserved	1	Reserved.

449 #### 9.2.6.1.1 Additional Requirements on SET_CRED_ID_PARAMS

450 When locking or unlocking, only the requested Credential ID shall be capable of locking its own credential parameters and associated policy if the `LockSelfPrivilege` policy bit permits. See Locking and Unlocking Attributes and DSP0289 Authorization Policy for additional requirements.

451 ### 9.2.6.2 GET_CRED_ID_PARAMS request and CRED_ID_PARAMS response messages

452 The `GET_CRED_ID_PARAMS` request shall be used to retrieve information about credentials provisioned in a credential slot. If an invalid credential slot or credential slot that is not provisioned is provided as input, the authorization target shall respond with `AUTH_ERROR` and `ErrorCode=InvalidRequest`. When `CredentialID` is valid but authorization checks fails, the authorization target shall respond with `AUTH_ERROR` and `ErrorCode=AccessDenied`.

453 Table 28 — GET_CRED_ID_PARAMS request message format shows the `GET_CRED_ID_PARAMS` request message format:

454 **Table 28 — GET_CRED_ID_PARAMS request message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `GET_CRED_ID_PARAMS` in Table 12 — Authorization Message request codes.
1	Reserved	1	Reserved.
2	CredentialID	2	Credential ID for which information is required.

455 Table 29 — Successful CRED_ID_PARAMS response message format shows the successful `CRED_ID_PARAMS` response message format:

456 **Table 29 — Successful CRED_ID_PARAMS response message format**

Byte offset	Field	Size (bytes)	Description
1	RequestResponseCode	1	Shall be the code value for `SET_CRED_ID_PARAMS_DONE` in Table 13 — Authorization Message response codes.
2	Reserved	1	Reserved.

Byte offset	Field	Size (bytes)	Description
3	CredAttributes	2	The field indicates credential attributes of the requested Credential ID. The format of this field shall be the format as Table 29.1 defines.
5	CredParams	Variable	This field represent identity information associated with the requested Credential ID. The size and format of this field shall be the same size and format as Table 2 — Credential Structure defines.

457 Table 29.1 — Credential Attributes defines the various credential ID attributes:

458 **Table 29.1 — Credential Attributes bit definition**

Byte Offset	Bit Offset	Field	Description
0	0	Lockable	If the Authorization target supports the ability to lock the credentials and associated policies of the requested credential ID, this bit shall be set.
0	1	Unlockable	If the Authorization target supports the ability to unlock the credentials and associated policies of the requested credential ID, this bit shall be set.
0	2	Locked	If the credentials and associated policy of the requested Credential ID is locked, this bit shall be set. This bit can be set or cleared through the `Lock` or `Unlock` operation in `SET_CRED_ID_PARAMS` request. If this bit is set, the `Lockable` bit shall also be set.
0	[7:3]	Reserved	Reserved
1	[7:0]	Reserved	Reserved

459 #### 9.2.6.3 Credential provisioning authorization requirements

460 The Authorization target shall perform authorization checks for `SET_CRED_ID_PARAMS` and `GET_CRED_ID_PARAMS` requests except for the scenarios that Initial provisioning details.

461 ## 9.2.7 Authorization policy provisioning and management

462 #### 9.2.7.1 SET_AUTH_POLICY request and SET_AUTH_POLICY_DONE response messages

463 The `SET_AUTH_POLICY` request shall be used to associate a policy with a credential as described in the Authorization policies section. When `PolicyList` provides an invalid credential slot or policy, the authorization target shall respond with `AUTH_ERROR` and `ErrorCode=InvalidRequest` respectively.

464 The Authorization Initiator shall use the `SetAuthPolicyOp` field to specify the operation for the request. An authorization target shall ensure that the operation is atomic, that is, the requested operation can successfully complete for all policies in the `PolicyList` and fail if that is not possible. When `PolicyList` provides an invalid credential slot or policy, the authorization target shall respond with `AUTH_ERROR` and `ErrorCode=InvalidRequest`.

When `SetAuthPolicyOp` is valid but authorization checks fails, the authorization target shall respond with `AUTH_ERROR` and `ErrorCode=AccessDenied`.

465 Table 30 — SET_AUTH_POLICY request message format shows the `SET_AUTH_POLICY` request message format:

466 **Table 30 — SET_AUTH_POLICY request message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `SET_AUTH_POLICY` in Table 12 — Authorization Message request codes.
1	Reserved	1	Reserved.
2	SetAuthPolicyOp	1	The field indicates the requested operation. The format of this field shall be the format as Table 31 defines.
3	PolicyList	Variable	This field represents the policy information to change that is associated with the given Credential ID. This field shall only represent the policies associated with a single Credential ID. The size and format of this field shall be the same size and format as Table 3 — Policy List defines.

467 **Table 31 — Values for `SetAuthPolicyOp` field**

Value	Operation Name	Description
0	Reserved	Reserved
1	PolicyChange	Shall indicate an operation that modifies the authorization policy associated with the given credential IDs.
2	Lock	This field shall have the same definition as the `Lock` operation as Table 26 defines.
3	Unlock	This field shall have the same definition as the `Unlock` operation as Table 26 defines.
All other values	Reserved	Reserved

468 Table 32 — Successful SET_AUTH_POLICY_DONE response message format shows the successful `SET_AUTH_POLICY_DONE` response message format:

469 **Table 32 — Successful SET_AUTH_POLICY_DONE response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `SET_AUTH_POLICY_DONE` in Table 13 — Authorization Message response codes.
1	Reserved	1	Reserved.

9.2.7.1.1 Additional requirements on SET_AUTH_POLICY

When locking or unlocking, see locking and unlocking requirements in Additional requirements on SET_CRED_ID_PARAMS.

9.2.7.2 GET_AUTH_POLICY request and AUTH_POLICY response messages

The `GET_AUTH_POLICY` request shall be used to retrieve the policy associated with a provisioned credential slot. If an invalid credential slot or credential slot that does not have a policy associated is provided as input, the authorization target shall respond with `AUTH_ERROR` and `ErrorCode=InvalidRequest`. When `CredentialID` is valid but authorization checks fails, the authorization target shall respond with `AUTH_ERROR` and `ErrorCode=AccessDenied`.

Table 33 — GET_AUTH_POLICY request message format shows the `GET_AUTH_POLICY` request message format:

Table 33 — GET_AUTH_POLICY request message format

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `GET_AUTH_POLICY` in Table 12 — Authorization Message request codes.
1	Reserved	1	Reserved.
2	CredentialID	2	Credential ID for which information is required.

Table 34 — Successful AUTH_POLICY response message format shows the successful `AUTH_POLICY` response message format:

Table 34 — Successful AUTH_POLICY response message format

Byte offset	Field	Size (bytes)	Description
1	RequestResponseCode	1	Shall be the code value for `AUTH_POLICY` in Table 13 — Authorization Message response codes.
2	Reserved	1	Reserved.
3	PolicyAttributes	2	The field indicates attributes of all policies associated with the requested Credential ID. The format of this field shall be the format as Table 29.1 defines.
5	PolicyList	Variable	This field represents all the policy information associated with the requested Credential ID. The size and format of this field shall be the same size and format as Table 3 — Policy List defines.

9.2.7.3 Authorization requirements

The Authorization target shall perform authorization checks for `SET_AUTH_POLICY` and `GET_AUTH_POLICY` requests except for the scenarios that Initial provisioning details.

9.2.8 Authorization process management

9.2.8.1 General Authorization Process Management

Authorization requests and responses in this section apply to all Authorization processes.

9.2.8.1.1 GET_AUTH_PROCESSES request and AUTH_PROCESSES response messages

The `GET_AUTH_PROCESSES` request and `AUTH_PROCESSES` response messages retrieves the list of active or saved Authentication processes associated with the requested Credential ID. A credential ID shall always be capable of retrieving its own information regardless of the value of `RetrieveAuthProcListPrivilege` bit.

Table 1000 — GET_AUTH_PROCESSES request message format shows the `GET_AUTH_PROCESSES` request message format:

Table 1000 — GET_AUTH_PROCESSES request message format

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `GET_AUTH_PROCESSES` in Table 12 — Authorization Message request codes.
1	Reserved	1	Reserved.
2	CredentialID	2	Shall be a Credential ID. A value of 0xFFFF shall indicate all credential ID.

Table 1001— AUTH_PROCESSES response message format shows the `AUTH_PROCESSES` response message format:

Table 1001— AUTH_PROCESSES response message format

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `AUTH_PROCESSES` in Table 13 — Authorization Message response codes.
1	Reserved	1	Reserved.

Byte offset	Field	Size (bytes)	Description
2	AuthProcInfoCount	2	Shall be a count of the number of Authorization processes information in `AuthProcInfoList` associated with the requested Credential ID. If there are no saved or active Authorization process for the requested Credential ID, the value of this field shall be zero.
4	AuthProcInfoList	Variable	Shall be a list of active or saved Authorization processes. The format of this field shall be the concatenation of one or more Authorization process information as Table 1002 - Authorization process information format defines. The size of this field shall be the size of an Authorization process information multiplied by `AuthProcInfoCount`.

489 Table 1002 — Authorization process info format shows the authorization process information format:

490 **Table 1002— Authorization process information format**

Byte offset	Field	Size (bytes)	Description
0	CredentialID	2	Shall be the credential ID associated with the Authorization process.
1	AuthProcessType	1	Shall indicate the type of active or saved Authorization process type associated with `CredentialID`. The values of this field shall be as follows: 0 - Shall indicate an active USAS. 1 - Shall indicate an active SEAP. 2 - Shall indicate a saved USAS. All other values - Reserved
2	AuthProcID	32	Shall be the Authorization process ID associated with the `CredentialID` and `AuthProcessType`. The value of this field shall be the Authorization process ID as Authorization Process ID Calculation defines.

491 **9.2.8.1.2 KILL_AUTH_PROCESS request and PROCESS_KILLED response messages**

492 The `KILL_AUTH_PROCESS` request and `PROCESS_KILLED` response messages terminates an authorization process.

493 If the requested Authorization process to terminate is an active USAS, the USAS session shall end immediately and incoming messages requiring authorization shall fail authorization checks for the given credential ID. If the requested Authorization process is a saved USAS, the saved USAP information shall no longer persist and consequently, the User shall not be able to continue the requested USAS.

494 If the requested Authorization process to terminate is an active SEAP, all messages requiring authorization shall fail authorization checks but the SPDM session shall remain unaffected. The Authorization target can consequently end the SPDM session.

495 A credential ID shall only be capable of killing its own Authorization process regardless of the value of `KillAuthProcListPrivilege` bit.

496 Table 1003 — KILL_AUTH_PROCESS request message format shows the `KILL_AUTH_PROCESS` request message format:

497 **Table 1003 — KILL_AUTH_PROCESS request message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `KILL_AUTH_PROCESS` in Table 12 — Authorization Message request codes.
1	Reserved	1	Reserved.
2	CredentialID	2	Shall be the Credential ID of the desired Authorization process to terminate.
4	AuthProcID	32	Shall be the desired Authorization process ID associated with the `CredentialID` to terminate. The value of this field shall be the authorization process ID as Authorization Process ID Calculation defines.

498 Table 1004— PROCESS_KILLED response message format shows the `PROCESS_KILLED` response message format:

499 **Table 1004— PROCESS_KILLED response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `PROCESS_KILLED` in Table 13 — Authorization Message response codes.
1	Reserved	1	Reserved.
2	AuthProcID	32	Shall be the requested Authorization process ID.

500 #### 9.2.8.1.2.1 Additional requirements for KILL_AUTH_PROCESS

501 If the Authorization target fails to kill a process after passing authorization checks, the Authorization target shall respond with an `AUTH_ERROR` message using the `ErrorCode = OperationFailed`.

502 #### 9.2.8.1.3 Authorization Process ID Calculation

503 The Authorization Process ID shall be the TPM_ALG_SHA_256.

504 To calculate the SHA2-256 hash, the Authorization endpoint shall first form `auth_proc_id_octet_string` by concatenate the following in order for a given Authorization process:

505 - String Prefix
506 - For USAP, the prefix shall not be present.
507 - For SEAP, the prefix shall be one of the following:
508 - If the SPDM Responder is an Authorization target, the prefix shall be "Responder".
509 - If the SPDM Requester is an Authorization target, the prefix shall be "Requester".
510 - The Authorization initiator's nonce
511 - For USAP, this shall be the `[START_AUTH].Nonce`.

512 ◦ For SEAP, this shall be the SPDM requester's nonce provided in the Session-Secret-Exchange Request.
513 • Authorization target's nonce
514 ◦ For USAP, this shall be the `[START_AUTH_RSP].Nonce`.
515 ◦ For SEAP, this shall be the SPDM responder's nonce provided in the Session-Secret-Exchange Response.
516 • `SavedSequenceNumber` if the Authorization process is a saved USAS.

517 The `auth_proc_id_octet_string` shall be the message to hash.

518 9.2.8.2 USAP Management

519 9.2.8.2.1 START_AUTH request and START_AUTH_RSP response messages

520 The `START_AUTH` request and `START_AUTH_RSP` messages are used to establish a User-specific authorization session as described in USAP. The Authorization target shall respond with an `AUTH_ERROR` with `ErrorCode=UnexpectedRequest` or silently discard the request if a `START_AUTH` is received for a User with a corresponding active USAS. See General USAP Error Handling for more information.

521 Table 35 — START_AUTH request message format shows the `START_AUTH` request message format:

522 **Table 35 — START_AUTH request message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `START_AUTH` in Table 12 — Authorization Message request codes.
1	Reserved	1	Reserved.
2	CredentialID	2	A unique identifier to identify the credential and the credential slot. This also identifies the user for whom a USAS is started.
4	Attributes	1	Shall be the same format as Table 35-1 - START_AUTH Request Attributes definition defines.
5	NonceLen	1	Length of the `Nonce` field. Shall be 32 bytes for this version of the specification
6	Nonce	`NonceLen`	Random sequence of bytes chosen by the user identified by `CredentialID`.

523 Table 35-1 — START_AUTH Request Attributes definition shows the field definition for `[START_AUTH].Attributes` field:

524 **Table 38.1 — START_AUTH Request Attributes definition**

Bit offset	Field	Description
0	Continue	If set, the Authorization target shall continue a prior USAP associated with the requested `CredentialID`. The Authorization target shall use the requested `CredentialID` and `Nonce` to ensure the correct USAP information is loaded. See more details in USAP continuation section.
[7:1]	Reserved	Reserved.

525 Table 36 — Successful START_AUTH_RSP response message format shows the `START_AUTH_RSP` response message format:

526 **Table 36 — Successful START_AUTH_RSP response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `START_AUTH_RSP` in Table 13 — Authorization Message response codes.
1	Reserved	1	Reserved.
2	CredentialID	2	Shall be the `CredentialID` from the corresponding `START_AUTH` request.
4	NonceLen	1	Length of the `Nonce` field. Shall be 32 bytes for this version of the specification
5	Nonce	`NonceLen`	Random sequence of bytes chosen by the authorization target. If the `Continue` bit in the `Attributes` field of the corresponding request is set, the Authorization target shall populate this field with the saved `Nonce` corresponding to the `Nonce` in the corresponding request.

527 #### 9.2.8.2.1.1 START_AUTH Additional Errors

528 If an Authorization target cannot find a preserved USAP associated with the requested `CredentialID` and `Nonce`, the Authorization target shall return an `AUTH_ERROR` response using an `ErrorCode = InvalidRequest`.

529 #### 9.2.8.2.2 END_AUTH request and END_AUTH_RSP response messages

530 The `END_AUTH` request and `END_AUTH_RSP` messages are used to terminate a USAS established using the `START_AUTH` command. The termination of the Authorization session restores an SPDM session to its original privilege level for that User. Additionally, the termination of a USAS does not end the corresponding SPDM session. If a session for the corresponding user does not exist, the authorization target shall return `AUTH_ERROR` with `ErrorCode=InvalidRequest`.

531 Table 37 — END_AUTH request message format shows the `END_AUTH` request message format:

532 **Table 37 — END_AUTH request message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `END_AUTH` in Table 12 — Authorization Message request codes.
1	Reserved	1	Reserved.
2	CredentialID	2	A unique identifier to identify the credential and the credential slot. This also identifies the user for which a user-specific authorization session is started.
3	Attributes	1	Shall be the format as Table 38-1 - END_AUTH Request Attributes definition.

533 Table 38 — Successful END_AUTH_RSP response message format shows the `END_AUTH_RSP` response message format:

534 **Table 38 — Successful END_AUTH_RSP response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `END_AUTH_RSP` in Table 13 — Authorization Message response codes.
1	Reserved	1	Reserved.
2	CredentialID	2	Shall be the `CredentialID` from the corresponding `END_AUTH` request.

535 Table 38-1 — END_AUTH Request Attributes definition shows the field definition for `[END_AUTH].Attributes` field:

536 **Table 38.1 — END_AUTH Request Attributes definition**

Bit offset	Field	Description
[1:0]	PersistMethod	Shall indicate the persistence type for the USAP associated with the requested `CredentialID`. This field shall have the following definition: 0 - The Authorization target shall erase the USAP information immediately upon the successful completion of this request. This also means if the USAP information was previous persisted, the USAP information will no longer persist. 1 - The Authorization target shall persist or continue to persist the USAP information until the next device reset. 2 - The Authorization target shall persist or continue to persist the USAP information across a reset until credential information associated with the requested Credential ID changes. 3 - Reserved USAP continuation defines the USAP information associated with `CredentialID` to persist or erase. An Authorization initiator can change the value of this field the next time it continues and ends the same USAS. However, if a User continues a saved USAP and ends the USAP without issuing a successfully authorized message, then the value of this field shall remain the same persist method as before the continuation. All Authorization processes can terminate by the `KILL_AUTH_PROCESS` request regardless of the value of this field.
[7:1]	Reserved	Reserved.

9.2.8.3 SEAP Management

9.2.8.3.1 ELEVATE_PRIVILEGE request and PRIVILEGE_ELEVATED response messages

`ELEVATE_PRIVILEGE` request and `PRIVILEGE_ELEVATED` response are used to start the authorization session when the SPDM Endpoint Authorization Process is used. These messages shall be used only during the application phased of the SPDM session. To initiate the authorization session, the Authorization Initiator shall send a `ELEVATE_PRIVILEGE` request and the Authorization target shall respond with `PRIVILEGE_ELEVATED` for a successful response. This request and response pair elevates the privilege level of the SPDM secure session for the Authorization Initiator for all subsequent messages until the privilege level is lowered. An Authorization target shall return an `AUTH_ERROR` with `ErrorCode=InvalidRequest` if there is a failure during the first part of SEAP (that is, the `SEAP_SUCCESS` was absent for the corresponding Authorization Initiator). An Authorization target shall return an `AUTH_ERROR` with `ErrorCode=InvalidRequest` or silently discard the `ELEVATE_PRIVILEGE` request if the session's privilege level is already elevated.

Table 39 — ELEVATE_PRIVILEGE request message format shows the `ELEVATE_PRIVILEGE` request message format:

Table 39 — ELEVATE_PRIVILEGE request message format

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `ELEVATE_PRIVILEGE` in Table 12 — Authorization Message request codes.

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

542 Table 40 — Successful PRIVILEGE_ELEVATED response message format shows the `PRIVILEGE_ELEVATED` response message format:

543 **Table 40 — Successful PRIVILEGE_ELEVATED response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `PRIVILEGE_ELEVATED in Table 13 — Authorization Message response codes.
1	Reserved	1	Reserved.

544 **9.2.8.3.2 END_ELEVATED_PRIVILEGE request and ELEVATED_PRIVILEGE_ENDED response message**

545 `END_ELEVATED_PRIVILEGE` request and `ELEVATED_PRIVILEGE_ENDED` response are used to terminate the authorization session when SEAP is used. An Authorization target shall return an `AUTH_ERROR` with `ErrorCode=InvalidRequest` if there is no SEAP in progress.

546 Table 41 — END_ELEVATED_PRIVILEGE request message format shows the `END_ELEVATED_PRIVILEGE` request message format: **Table 41 — END_ELEVATED_PRIVILEGE request message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `END_ELEVATED_PRIVILEGE` in Table 12 — Authorization Message request codes.
1	Reserved	1	Reserved.

547 Table 42 — Successful ELEVATED_PRIVILEGE_ENDED response message format shows the `ELEVATED_PRIVILEGE_ENDED` response message format:

548 **Table 42 — Successful ELEVATED_PRIVILEGE_ENDED response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `ELEVATED_PRIVILEGE_ENDED` in Table 13 — Authorization Message response codes.
1	Reserved	1	Reserved.

549 ## 9.2.9 Basic Management

550 Messages in this section provide general management of the Authorization target.

9.2.9.1 TAKE_OWNERSHIP request and OWNERSHIP_TAKEN response

The `TAKE_OWNERSHIP` request and its successful `OWNERSHIP_TAKEN` response shall cause the Authorization target to exit the default state and fully enforce authorization for all messages requiring authorization. This request and response has no associated policy bit and thus any Credential ID has the authority to issue this request. However, the Authorization target still performs authorization checks.

If Ownership is already taken, the Authorization target shall respond with an `AUTH_ERROR` message using `ErrorCode = UnexpectedRequest` error code.

Table 41.100 — TAKE_OWNERSHIP request message format shows the `TAKE_OWNERSHIP` request message format:

Table 41.100 — TAKE_OWNERSHIP request message format

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `TAKE_OWNERSHIP` in Table 12 — Authorization Message request codes.
1	Reserved	1	Reserved.

Table 41.101 — Successful OWNERSHIP_TAKEN response message format shows the `OWNERSHIP_TAKEN` response message format:

Table 41.101 — Successful OWNERSHIP_TAKEN response message format

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `OWNERSHIP_TAKEN` in Table 13 — Authorization Message response codes.
1	Reserved	1	Reserved.

9.2.9.2 AUTH_RESET_TO_DEFAULT request and AUTH_DEFAULTS_APPLIED response

The `AUTH_RESET_TO_DEFAULT` request and its successful `AUTH_DEFAULTS_APPLIED` response shall cause the authorization target to restore all data associated with the requested parameters back to factory defaults. Depending on the requested parameters, an Authorization target may require a reset for defaults to become effective.

The Authorization target shall restore data to defaults only for unlocked credential IDs and their associated policies.

Table 43 — AUTH_RESET_TO_DEFAULT request message format shows the `AUTH_RESET_TO_DEFAULT` request message format:

Table 43 — AUTH_RESET_TO_DEFAULT request message format

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `AUTH_RESET_TO_DEFAULT` in Table 12 — Authorization Message request codes.

Byte offset	Field	Size (bytes)	Description
1	Reserved	1	Reserved.
2	DataType	2	This field indicates the type of data to reset back to default. The format of this field shall be the format as Table 44 — Data Type Bit Definition defines.
4	CredentialID	2	The value of this field shall indicate the unlocked credential ID(s) to reset back to default. The value of 0xFFFF shall indicate all unlocked credential IDs.
6	SVResetDataTypeCount	2	This field shall be the count of Standard or Vendor Reset Data Type Elements in `SVResetDataTypeList`. A value of zero shall indicate the absence of `SVResetDataTypeList`.
8	SVResetDataTypeList	Variable	This field shall cause data types defined by a standard body or vendor to restore back to their factory defaults. The format of this field shall be the concatenation of Standard or Vendor Reset Data Type Element as Table 45 — Standard or Vendor Reset Data Type Element Format defines. If a standard or vendor is present in this list, then the list can contain more than one instance of that standard or vendor because a standard body may have multiple standards with their corresponding data types. This specification recommends that the standard or vendor prevent duplicate instances to minimize payload.

562 Table 44 — Data Type Bit Definition shows the `DataType` bit definition:

563 **Table 44 — Data Type Bit Definition**

Byte Offset	Bit Offset	Field	Description
0	0	CredIDParams	If this bit is set, Credential ID parameters shall be reset to their default values.
0	1	AuthPolicy	If this bit is set, the authorization policy shall be reset to their default values.
0	[7:2]	Reserved	Reserved
1	[7:0]	Reserved	Reserved.

564 Table 45 — Standard or Vendor Reset Data Type Element Format shows the definition for the standard or vendor data type to restore back to factory defaults:

565 **Table 45 — Standard or Vendor Reset Data Type Element Format**

Byte offset	Field	Size (bytes)	Description
0	SVResetDataTypeOwner	`LenSVH`	This field shall specify the owner of the `SVResetDataType` field. The format and size of this field shall be the format and size of the SVH as SPDM defines. If other DMTF DSP uses the format as this table defines, then the other DMTF DSP specifications shall use the value associated with DMTF-DSP for the `ID` field as SPDM defines.

Byte offset	Field	Size (bytes)	Description
`LenSVH`	SVResetDataTypeLen	1	The value of this field plus 1 shall specify the length of `SVResetDataType`. The value of this field shall not exceed 31, indicating a maximum of 32 bytes.
1 + `LenSVH`	SVResetDataType	Variable	This field shall indicate the standard or vendor specific data types to restore back to factory defaults. The `SVResetDataTypeOwner` defines the format and size for this field.

566 The Authorization target shall reset all data associated with the requested `DataType` and requested `CredentialID`.

567 Table 46 — AUTH_DEFAULTS_APPLIED response message format shows the `AUTH_DEFAULTS_APPLIED` response message format:

568 **Table 46 — AUTH_DEFAULTS_APPLIED response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `AUTH_DEFAULTS_APPLIED` in Table 13 — Authorization Message response codes.
1	Reserved	1	Reserved.

569 If the Authorization requires a reset to successfully complete the request and there are no other errors, the Authorization target shall reply with `AUTH_ERROR` with a `ErrorCode=ResetRequired`. Otherwise, a successful response shall indicate all requested data types for the requested Credential ID(s) have been restored to their default values and the default values immediately applied. The behavior of the authorization target for the requested Credential ID(s) and the requested data type also restores back to default behavior. The default values and behavior of the Authorization target is outside the scope of this specification.

570 When a value of 0xFFFF is used in the `CredentialID` field and all non-reserved bits set in `DataType` field of the request, the Authorization target shall return to the default state as Initial provisioning describes.

571 Lastly, `AUTH_RESET_TO_DEFAULT` request is an invasive operation. Thus, an Authorization target shall immediately terminate all active and saved Authorization processes associated with the requested Credential IDs after the `AUTH_DEFAULTS_APPLIED` response has been sent.

572 ## 9.3 Timing Requirements

573 This section discusses timing requirements for Authorization messages and all messages requiring authorization.

574 ### 9.3.1 Message Transmission Time

575 The message transmission time is the worst-case transmission time it takes the Authorization initiator to completely transmit a message to the Authorization target plus the worst-case transmission time for the Authorization target to completely send a message to the Authorization initiator.

9.3.2 Authorization Messages Timing

For messages not requiring authorization, the Authorization target shall respond within `AuthResponseTime` measured from the reception of the Authorization request to the transmission of the corresponding response. The value of `AuthResponseTime` shall be 100 ms.

If an Authorization initiator wants to retry a request, the Authorization initiator shall wait at least `AuthResponseTime` plus message transmission time. The actual value and method of measurement of the message transmission time is outside the scope of this specification.

9.3.3 All Messages requiring Authorization

Because this specification provides a mechanism for authorizing messages for any protocol, the Authorization target can consume additional processing time to process the messages. Protocols that adopt this specification should consider the additional process time needed and adjust existing timing requirements accordingly.

The Authorization target provides this additional processing time in `[AUTH_CAPABILITIES].AuthRecordProcessTime` field to process the authorization record. The transport can use this value if it uses the Authorization record.

If an Authorization initiator wants to retry an Authorization request, the Authorization initiator shall wait at least the sum of these timing parameters:

- `AuthResponseTime`
- `[AUTH_CAPABILITIES].AuthRecordProcessTime`
- The message transmission time.

Unless otherwise specified by the transport, the Authorization initiator should wait at least the sum of these timing parameters before performing any error handling for messages of other protocols encapsulated in an Authorization record:

- `AuthRecordProcessTime`
- The process time of `MsgToAuthPayload` in the Authorization record as specified by the transport
- The message transmission time

10 Authorization Opaque Data Structures

Authorization Opaque Data Structures (AODS) are data structures that are populated into the `OpaqueData` field of various SPDM messages. Other parts of this specification define which AODS populates into which SPDM message. This section defines the format for each AODS.

10.1 General Authorization Opaque Data Structure

All AODS format shall follow the General opaque data format as SPDM defines. This section binds the AODS to the General opaque data format.

Table 47 — AODS General Format defines the general format of all AODS.

Table 47 — AODS General Format

Byte Offset	Field	Size (bytes)	Description
0	ID	1	The value of this field shall be 0xB to identify DMTF-DSP as the standards body.
1	VendorIDLen	1	The value of this field shall be 2 to identify DMTF-DSP as the owner of the definition of all AODS.
2	DMTFspecID	2	The value of this field shall be 289. This field indicates that the definition of the `OpaqueElementData` belongs to this DMTF specification.
4	OpaqueElementDataLen	2	The value of this field shall be the total size of these fields: `AODSid` and `AODSbody` field.
6	AODSid	1	This field identifies the AODS and its format in `AODSbody`. The value of this field shall be one of the values in the **AODS ID** column of Table 48 — AODS IDs.
7	AODSbody	AODSbodyLen	This field shall contain the actual AODS content according to the value in `AODSid`. See the respective AODS section for the actual definition. The size of this field shall be the size of `AODSbody` corresponding to the value in `AODSid` field.
7 + `AODSbodyLen`	AlignPadding	Variable	See field of the same name in SPDM for definition and requirements. The `OpaqueElementData` are the fields following `DMTFspecID` inclusively but not including this field.

SPDM 1.2 or later defines the General opaque data format for all opaque data populated in all `OpaqueData` fields of SPDM messages when `OpaqueDataFmt1` is selected as the Opaque data format for the SPDM connection. Prior to SPDM 1.2 or when `OpaqueDataFmt1` is not the selected Opaque data format for the SPDM connection, the format of the `OpaqueData` field is out of scope of this specification.

10.2 AODS Error Handling

This specification defines which SPDM message an AODS can be present in and other AODS requirements. An error arises when an Authorization endpoint does not meet these AODS requirements, such as an unexpected presence. When an error occurs, an Authorization endpoint can terminate the session, prevent Authorization processes in the corresponding session or other error handling mechanisms that are outside the scope of this specification.

10.3 AODS IDs

Table 48 — AODS IDs lists out all AODS in this specification with a short description.

Table 48 — AODS IDs

AODS ID	AODS Name	Description
0	INVOKE_SEAP	Shall invokes the SEAP process for an SPDM endpoint. The format of the `AODSbody` shall be the INVOKE_SEAP AODS.
1	SEAP_SUCCESS	Shall indicate the SPDM secure session handshake phase of the SEAP process has successfully passed for the corresponding SPDM endpoint. The format of the `AODSbody` shall be the SEAP_SUCCESS AODS.
2	AUTH_HELLO	Shall indicate the SPDM endpoint supports being an Authorization target. The format of the `AODSbody` shall be the AUTH_HELLO AODS.
All other values	Reserved	Reserved

10.4 INVOKE_SEAP AODS

The INVOKE_SEAP AODS shall request the other SPDM endpoint to invoke the SEAP process for the requesting SPDM endpoint. Table 49 — INVOKE_SEAP body definition defines the format for the `AODSbody` in the AODS general format when AODS ID is zero.

Table 49 — INVOKE_SEAP Body Definition

Byte Offset	Field	Size (bytes)	Description
0	PresenceExtension	1	This field shall indicate the presence of extra fields. The value of this field shall be reserved.
1	CredentialID	2	The field shall contain the credential ID of the requesting SPDM endpoint.

Because the INVOKE_SEAP AODS occurs before the SPDM endpoint knows the supported Authorization versions of the other SPDM endpoints, the `PresenceExtension` field helps maintain future compatibility. Future versions of this specification could define the next lowest unused bit. If a bit is set, the corresponding field shall be present.

606 This allows current implementation to skip the remaining fields and only process fields it knows about. An implementation can skip remaining fields it doesn't know about by taking into account the `OpaqueElementDataLen` in the General AODS format.

607 10.5 SEAP_SUCCESS AODS

608 The SEAP_SUCCESS AODS shall indicate the SEAP process during the SPDM session handshake phase for the requesting SPDM endpoint is successful. Table 51 — SEAP_SUCCESS body definition defines the format for the `AODSbody` in the AODS general format when AODS ID is two.

609 **Table 51 — SEAP_SUCCESS Body Definition**

Byte Offset	Field	Size (bytes)	Description
0	PresenceExtension	1	This field shall indicate the presence of extra fields. The value of this field shall be reserved.

610 Because the SEAP_SUCCESS AODS occurs before the SPDM endpoint knows the supported Authorization versions of the other SPDM endpoints, the `PresenceExtension` field helps maintain future compatibility. Future versions of this specification could define the next lowest unused bit. If a bit is set, the corresponding field shall be present.

611 This allows current implementation to skip the remaining fields and only process fields it knows about. An implementation can skip remaining fields it doesn't know about by taking into account the `OpaqueElementDataLen` in the General AODS format.

612 10.6 AUTH_HELLO AODS

613 The AUTH_HELLO AODS shall indicate the SPDM endpoint providing this AODS is an Authorization target. Table 52 — AUTH_HELLO body definition defines the format for the `AODSbody` in the AODS general format when AODS ID is 3.

614 **Table 52 — AUTH_HELLO Body Definition**

Byte Offset	Field	Size (bytes)	Description
0	PresenceExtension	1	This field shall indicate the presence of extra fields. The value of this field shall be reserved.

615 Because the AUTH_HELLO AODS occurs before the SPDM endpoint knows the supported Authorization versions of the other SPDM endpoints, the `PresenceExtension` field helps maintain future compatibility. Future versions of this specification could define the next lowest unused bit. If a bit is set, the corresponding field shall be present.

616 This allows current implementation to skip the remaining fields and only process fields it knows about. An implementation can skip remaining fields it doesn't know about by taking into account the `OpaqueElementDataLen` in the General AODS format.

11 Other Transport Requirements

This section describes other or additional requirements that are not discussed elsewhere in this specification.

11.1 Authorization Record over SPDM Vendor Defined Messages

This clause defines the Authorization record over SPDM Vendor-defined messages to enable transmission of Authorization messages, Authorization records and messages of any protocol requiring authorization over existing transports. By leveraging SPDM's Vendor-defined messages, existing transports can utilize their current SPDM bindings without requiring significant modifications. These requests and responses are intended for use between SPDM endpoints acting as Authorization initiator and Authorization target.

AUTH record over SPDM VDM messages shall not affect the SPDM transcript defined in the SPDM specification. Additionally, depending on the type of Authorization record and its content, one or more SPDM requests can be outstanding at any time. Furthermore, an Authorization Record over SPDM VDM request can have a response that is not encapsulated in an Authorization record over SPDM VDM response. In a way, AUTH record over SPDM VDM behaves more like a transport than a request and response model.

All Authorization record over SPDM VDM shall use the SPDM `VENDOR_DEFINED_REQUEST` and `VENDOR_DEFINED_RESPONSE` request and response with these requirements:

- The `StandardID` shall be `0xB` to indicate DMTF-DSP.
- The `VendorID` shall be `289` (`0x121`) to indicate this specification.

The `VendorDefinedReqPayload` field of the `VENDOR_DEFINED_REQUEST` and `VendorDefinedRespPayload` field of the `VENDOR_DEFINED_RESPONSE` shall be the same format and size as Table 10 — Authorization Record format. If `LargeVendorDefinedReqPayload` is present in the `VENDOR_DEFINED_REQUEST` or `LargeVendorDefinedRespPayload` is present in the `VENDOR_DEFINED_RESPONSE` , then the format of these fields shall be the same format and size as Table 10 — Authorization Record format.

11.1.1 Additional AUTH over SPDM VDM requirements

The timing requirements for the AUTH Record over SPDM VDM requirements shall be the same as defined in the Timing Requirements clause.

12 Cryptographic Operations

This section describes or defines cryptographic functions specific to Authorization

12.1 Asymmetric Algorithms

This section defines the supported asymmetric algorithms.

Table 53.1.1 — Base Asymmetric Algorithm Format lists and defines the bit definition and other parameters associated with the respective asymmetric algorithms.

Table 53.1.1— Base Asymmetric Algorithm Format

Byte Offset	Bit Offset	Algorithm	Signature Length (bytes)	Description
0	0	TPM_ALG_RSASSA_2048	256	
0	1	TPM_ALG_RSAPSS_2048	256	
0	2	TPM_ALG_RSASSA_3072	384	
0	3	TPM_ALG_RSAPSS_3072	384	
0	4	TPM_ALG_ECDSA_ECC_NIST_P256	64	The signature format shall be 32-byte `r` followed by 32-byte `s`.
0	5	TPM_ALG_RSASSA_4096	512	
0	6	TPM_ALG_RSAPSS_4096	512	
0	7	TPM_ALG_ECDSA_ECC_NIST_P384	96	The signature format shall be 48-byte `r` followed by 48-byte `s`.
1	0	TPM_ALG_ECDSA_ECC_NIST_P521	132	The signature format shall be 66-byte `r` followed by 66-byte `s`
1	1	TPM_ALG_SM2_ECC_SM2_P256	64	The signature format shall be 32-byte `SM2_R` followed by 32-byte `SM2_S`.
1	2	EdDSA ed25519	64	The signature format shall be 32-byte `R` followed by 32-byte `S`.
1	3	EdDSA ed448	114	The signature format shall be 57-byte `R` followed by 57-byte `S`.
1	[7:4]	Reserved	Reserved	
2:7	All bits	Reserved	Reserved	

12.2 Hash Algorithms

This section defines the supported hash algorithms

Table 53.1.2 — Base Hash Algorithm Format lists and defines the bit definition of all supported base hash algorithms.

Table 53.1.2— Base Hash Algorithm Format

Byte Offset	Bit Offset	Algorithm
0	0	TPM_ALG_SHA_256
0	1	TPM_ALG_SHA_384
0	2	TPM_ALG_SHA_512
0	3	TPM_ALG_SHA3_256
0	4	TPM_ALG_SHA3_384
0	5	TPM_ALG_SHA3_512
0	6	TPM_ALG_SM3_256
0	7	Reserved
1:7	All bits	Reserved

12.3 Signature Generation and Validation

This sections describes the `AuthSign` and `AuthSigVerify` functions.

12.3.1 Signature algorithm references

Refer to the Signature algorithm references section in the SPDM specification (DSP0274) for details on signature algorithms.

12.3.2 Signature generation

The `AuthSign` function used in various part of this specification defines the signature generation algorithm while accounting for the differences in the various supported cryptographic signing algorithms.

The signature generation function takes this form:

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

The `AuthSign` function shall take these input parameters:

- `PrivKey` : a secret key associated with the given Credential ID
- `data_to_be_signed` : a bit stream of the data that will be signed
- `context` : a string

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

The signing function shall follow these steps to create `auth_prefix` and `auth_context` (See Text or string encoding for encoding rules):

1. Create `auth_prefix`. The `auth_prefix` shall be the repetition, four times, of the concatenation of "dmtf-auth-v", `AuthVersionString` and ".*". This will form a 64-character string.
2. Create `auth_context`. If the User is generating the signature, `auth_context` shall be the concatenation of "user-" and `context`.

Now follows an example, designated Example 1, of creating a `combined_auth_prefix`.

The version of this specification for this example is 1.4.3, the User is generating a signature, and the `context` is "my example context". Thus, the `auth_prefix` is "dmtf-auth-v1.4.*dmtf-auth-v1.4.*dmtf-auth-v1.4.*dmtf-auth-v1.4.*". The `auth_context` is "user-my example context".

Next, the `combined_auth_prefix` is formed. The `combined_auth_prefix` shall be the concatenation of four elements: `auth_prefix`, a byte with a value of zero, `zero_pad`, and `auth_context`. The size of `zero_pad` shall be the number of bytes needed to ensure that the length of `combined_auth_prefix` is 100 bytes. The size of `zero_pad` can be zero. The value of `zero_pad` shall be zero.

Continuing Example 1, Table 53 — Combined SPDM prefix shows the `combined_auth_prefix` with offsets. Offsets increase from left to right and top to bottom. As shown, the length of `combined_auth_prefix` is 100 bytes. Furthermore, a number surrounded by double quotation marks indicates that the ASCII value of that number is used. See Text or string encoding for encoding rules. Table 53 concludes Example 1.

Table 53 — Combined SPDM prefix

Offset	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0	d	m	t	f	-	a	u	t	h	-	v	"1"	.	"4"	.	*
0x10	d	m	t	f	-	a	u	t	h	-	v	"1"	.	"4"	.	*
0x20	d	m	t	f	-	a	u	t	h	-	v	"1"	.	"4"	.	*
0x30	d	m	t	f	-	a	u	t	h	-	v	"1"	.	"4"	.	*
0x40	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	u	s	e
0x50	r	-	m	y	space (0x20)	e	x	a	m	p	l	e	space (0x20)	c	o	n
0x60	t	e	x	t												

The next step is to form the `message_hash`. The `message_hash` shall be the hash of `data_to_be_signed` using the selected hash function associated with the given Credential ID. Many hash algorithms allow implementations to

compute an intermediate hash, sometimes called a running hash. An intermediate hash allows for the updating of the hash as each byte of the ordered data of the message becomes known. Consequently, the ability to compute an intermediate hash allows for memory utilization optimizations where an Authorization endpoint can discard bytes of the message that are already covered by the intermediate hash while waiting for more bytes of the message to be received.

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

660 12.3.2.1 RSA and ECDSA signing algorithms

661 All RSA and ECDSA specifications do not define a specific hash function. Thus, the hash function to use shall be the selected hash function associated with the given Credential ID.

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

663 In the specification for these algorithms, the letter `M` denotes the message to be signed. `M` shall be the concatenation of `combined_auth_prefix` and `message_hash`.

664 RSA and ECDSA algorithms are described in Signature algorithm references.

665 The FIPS PUB 186-5 supports deterministic ECDSA as a variant of ECDSA. RFC 6979 describes this deterministic digital signature generation procedure. This variant does not impact the signature verification process. How an implementation chooses to support ECDSA or deterministic ECDSA is outside the scope of this specification.

666 12.3.2.2 EdDSA signing algorithms

667 These algorithms are described in RFC 8032.

668 The private key, defined by RFC 8032, shall be `PrivKey`.

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

670 12.3.2.2.1 Ed25519 sign

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

672 `M` shall be the concatenation of `combined_auth_prefix` and `message_hash`.

673 12.3.2.2.2 Ed448 sign

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

675 `M` shall be the concatenation of `combined_auth_prefix` and `message_hash`.

676 Ed448 defines a context string, `C`. `C` shall be the `auth_context`.

677 12.3.2.3 SM2 signing algorithm

678 This algorithm is described in GB/T 32918.2-2016. GB/T 32918.2-2016 also defines the variable `M` and ID_A.

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

680 In the specification for SM2, the letter `M` denotes the message to be signed. `M` shall be the concatenation of `combined_auth_prefix` and `message_hash`.

681 The SM2 specification does not define a specific hash function. Thus, the hash function to use shall be the selected hash function associated with the given Credential ID.

682 Lastly, SM2 expects a distinguishing identifier, which identifies the signer and is indicated by the variable ID_A. If this algorithm is selected, the ID shall be an empty string of size 0.

683 12.3.3 Signature verification

684 The `AuthSigVerify` function, used in various part of this specification, defines the signature verification algorithm while accounting for the differences in the various supported cryptographic signing algorithms.

685 The signature verification function takes this form:

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

686 The `AuthSigVerify` function shall take these input parameters:

687 • `PubKey` : the public key associated with the given Credential ID

688 • `signature` : a digital signature

689 • `unverified_data` : a bit stream of data that needs to be verified

690 • `context` : a string

691 The function shall verify the `unverified_data` using `signature`, `PubKey`, and a selected cryptographic signing algorithm. `AuthSigVerify` shall return success if the signature verifies correctly and failure otherwise. Each cryptographic signing algorithm states the verification steps or criteria for successful verification.

692 The verifier of the signature shall create `auth_prefix`, `auth_context`, and `combined_auth_context` as described in Signature generation.

693 The next step is to form the `unverified_message_hash`. The `unverified_message_hash` shall be the hash of the `unverified_data` using the selected hash function associated with the given Credential ID.

694 The selected cryptographic signature verification algorithm is the one associated with the given Credential ID.

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

696 12.3.3.1 RSA and ECDSA signature verification algorithms

697 All RSA and ECDSA specifications do not define a specific hash function. Thus, the hash function to use shall be the selected hash function associated with the given Credential ID.

698 The public key, defined in the specification for these algorithms, shall be `PubKey`.

In the specification for these algorithms, the letter `M` denotes the message that is signed. `M` shall be concatenation of the `combined_auth_prefix` and `unverified_message_hash`.

For RSA algorithms, `AuthSigVerify` shall return success when the output of the signature verification operation, as defined in the RSA specification, is "valid signature". Otherwise, `AuthSigVerify` shall return a failure.

For ECDSA algorithms, `AuthSigVerify` shall return success when the output of "ECDSA Signature Verification Algorithm" as defined in FIPS PUB 186-5 is `"accept"`. Otherwise, `AuthSigVerify` shall return failure.

RSA and ECDSA algorithms are described in Signature algorithm references.

12.3.3.2 EdDSA signature verification algorithms

RFC 8032 describes these algorithms. RFC 8032, also, defines the `M`, `PH`, and `C` variables.

The public key, also defined in RFC 8032, shall be `PubKey`.

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

12.3.3.2.1 Ed25519 verify

`M` shall be the concatenation of `combined_auth_prefix` and `unverified_message_hash`.

`AuthSigVerify` shall return success when step 1 does not result in an invalid signature and when the constraints of the group equation in step 3 are met as described in RFC 8032 section 5.1.7. Otherwise, `AuthSigVerify` shall return failure.

12.3.3.2.2 Ed448 verify

`M` shall be the concatenation of `combined_auth_prefix` and `unverified_message_hash`.

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

`AuthSigVerify` shall return success when step 1 does not result in an invalid signature and when the constraints of the group equation in step 3 are met as described in RFC 8032 section 5.2.7. Otherwise, `AuthSigVerify` shall return failure.

12.3.3.3 SM2 signature verification algorithm

This algorithm is described in GB/T 32918.2-2016, which also defines the variable `M` and ID_A.

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

In the specification for SM2, the variable `M'` is used to denote the message that is signed. `M'` shall be the concatenation of `combined_auth_prefix` and `unverified_message_hash`.

The SM2 specification does not define a specific hash function. Thus, the hash function to use shall be the selected hash function associated with the given Credential ID.

719 Lastly, SM2 expects a distinguishing identifier, which identifies the signer, and is indicated by the variable ID_A. See SM2 signing algorithm to create the value for ID_A.

720 `AuthSigVerify` shall return success when the Digital signature verification algorithm, as described in GB/T 32918.2-2016, outputs an "accept". Otherwise, `AuthSigVerify` shall return failure.

721 # 13 Authorization events

722 The Authorization events are sent using SPDM Event mechanism. This section uses many variable names that SPDM defines. See DSP0274 for details, especially the eventing mechanism sections.

723 Authorization event requirements only apply when `AuthEventCap` is set. Otherwise, an Authorization target does not support Authorization events. The **Requirement** column indicates whether or not the event is mandatory or conditional. If a value in this column is *Mandatory*, the event shall be supported. If a value in this column is *conditional*, the section for the corresponding request details the requirements.

724 The `EventGroupId` in SPDM events identifies the owner of the event. For Authorization, the `EventGroupId` shall indicate DMTF-DSP with a Vendor ID value of 289.

725 The Authorization event types table shows the supported Authorization event types for the Authorization event group. The values in the **Event Type ID** column shall be the same values for `EventTypeId` field in the SPDM Event data table for the Authorization event group for the corresponding event in the **Event Name** column. The version (`EventGroupVer`) of the Authorization Event Group shall be `1` .

726 **Table 54 — Authorization event types table**

Event Type ID	Event Name	Requirement	Description
0	Reserved	Reserved	Reserved.
1	CredIDparamsChanged	Conditional	A change to one or more parameters via the `SET_CRED_ID_PARAMS` has occurred for a Credential ID.
2	AuthPolicyChanged	Conditional	One or more parameters associated with `SET_AUTH_POLICY` has changed for a Credential ID.
All others	Reserved	Reserved	Reserved.

727 ## 13.1 Event type details

728 Each Authorization event type has its own event-specific information, referred to as `EventDetail` , to describe the event. These clauses describe the format for each Authorization event type. The event types are listed in the Authorization event types table.

729 ### 13.1.1 Credential ID Parameters Changed event

730 An Authorization target shall use this event (`EventTypeId=CredIDparamsChanged`) to notify the Event Recipient as SPDM defines that the Authorization target made a change to one or more parameters by the `SET_CRED_ID_PARAMS` request. The event shall apply to all operations indicated by the `SetCredInfoOp` field in the `SET_CRED_ID_PARAMS` request.

731 The event shall be supported if the Authorization target supports the `SET_CRED_ID_PARAMS` request.

732 The Credential ID Parameters Changed format table describes the format for `EventDetail` as SPDM defines.

733 **Table 55 — Credential ID Parameters Changed format**

Offset	Field	Size (bytes)	Description
0	CredentialIdCount	2	Shall be the number of Credential IDs in `CredentialIdList`
2	CredentialIdList	Variable	Shall be a list of Credential IDs whose credential ID parameters changed through the `SET_CRED_ID_PARAMS` request. The format of this field shall be the concatenation of `CredentialID` s as Table 2 — Credential Structure defines. Thus, the size of this field shall be `CredentialIdCount` * the size of `CredentialID` .

734 The Authorization initiator can issue `GET_CRED_ID_PARAMS` to obtain details of this change.

735 ## 13.1.2 Authorization policy changed event

736 An Authorization target shall use the authorization policy changed event (`EventTypeId=AuthPolicyChanged`) to notify the Event Recipient as SPDM defines when one or more authorization policies have changed through the `SET_AUTH_POLICY` request. The event shall apply to all operations indicated by the `SetAuthPolicyOp` field in the `SET_AUTH_POLICY` request. The `EventDetail` format for this event type shall be as the Authorization policy changed event details format defines. This event only indicates a single policy change. If more than one policy changes, then each change will have their own event.

737 The event shall be supported if the Authorization target supports the `SET_AUTH_POLICY` request.

738 Table 56 — Authorization policy changed event details format describes the format for `EventDetail` for the `AuthPolicyChanged` event.

739 **Table 56 — Authorization policy changed event details format**

Offset	Field	Size (bytes)	Description
0	CredentialID	2	Shall be the credential ID associated with the authorization policy that changed.
2	PolicyOwnerID	PolicyOwnerIdLen	Shall identify the owner of the definition of the policy that changed. The format of this field shall be the SVH as SPDM defines. The length of this field shall be the length of the SVH.
2 + `PolicyOwnerIdLen`	PolicyIdLen	2	Shall be the length of `PolicyID` field.
4 + `PolicyOwnerIdLen`	PolicyID	`PolicyIdLen`	Shall identify the actual policy, defined by `PolicyOwnerID` , that changed. If the `PolicyOwnerID` indicates DSP0289 using DMTF-DSP as standards body registry, then the format and size of this field is the `PolicyType` field as Table 7 — DSP0289 General Policy Definitions defines.

740 The Authorization initiator can issue `GET_AUTH_POLICY` to obtain further details on the change.

14 ANNEX A (informative) change log

14.1 Version 1.0.0 (in progress)

- Initial release

15 Bibliography

DMTF DSP4014, *DMTF Process for Working Bodies*, https://www.dmtf.org/dsp/DSP4014