Security Protocol and Data Model (SPDM) Authorization Specification

Version: 1.0.0

Document Identifier: DSP0289

Date: 2025-10-17

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

Supersedes: None

Document Class: Normative

Document Status: Published

Document Language: en-US

CONTENTS

14 1 Foreword

15 The Security Protocols and Data Models (SPDM) Working Group of DMTF prepared the *Security Protocol and Data Model (SPDM) Authorization Specification* (DSP0289).

16 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.

17 1.1 Acknowledgments

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

- Lee Ballard — Dell Technologies
- Steven Bellock — NVIDIA Corporation
- Daniil Egranov — Arm Limited
- Sakul Gupta — Micron Technology Inc.
- Brett Henning — Broadcom Inc.
- Eric Hibbard — Samsung
- Jeff Hilland — HPE Labs
- Guerney D H Hunt — IBM
- Raghu Krishnamurthy — NVIDIA Corporation
- Will Marone — AMD Inc.
- Jim Panian — Qualcomm Inc.
- Scott Phuong — Cisco Systems Inc., Axiado Corporation, Microsoft Corporation
- Xiaoyu Ruan — Intel Corporation
- Jiewen Yao — Intel Corporation
- Sungho Yoon — Samsung
- Wilson Young — Solidigm

19
2 Introduction

20 The *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.

21
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.

22
2.1.1 Reserved and unassigned values

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

24 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.

25
2.1.2 Byte ordering

26 This section describes different byte orderings.

27
2.1.2.1 Default byte order

28 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).

29
2.1.2.2 Octet string byte order

30 A string of octets is conventionally written from left to right. Also by convention, byte 0 of the octet string shall be the leftmost byte of the octet string, byte 1 of the octet string shall be the second-leftmost byte of the octet string, and this pattern shall continue. 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.

31 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.

32 **2.1.2.3 Signature byte order**

33 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.

34 **2.1.2.3.1 ECDSA signatures byte order**

35 FIPS PUB 186-5 defines `r`, `s`, and the ECDSA signature to be `(r, s)`, where `r` and `s` are 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`.

36 **2.1.2.3.2 SM2 signatures byte order**

37 GB/T 32918.2-2016 defines `r` and `s` and SM2 signatures to be `(r, s)`, where `r` and `s` are integers. The sizes 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.

38 **2.1.3 Text or string encoding**

39 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.

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

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

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

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

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

43 ### 2.1.4 Other conventions

44 Unless otherwise specified, all figures are informative.

45 3 Scope

46 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.

47 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.

48 4 Normative references

49 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 DSP0274, *Security Protocol and Data Model (SPDM) Specification*, https://www.dmtf.org/dsp/DSP0274
- DMTF DSP0277, *Secured Messages using SPDM Specification*, https://www.dmtf.org/dsp/DSP0277
- DMTF DSP0293, *Standards Body and Vendor Header Registry*, https://www.dmtf.org/dsp/DSP0293
- *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

50 5 Terms and definitions

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

52 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.

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

54 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.

55 The terms that DSP0274 defines also apply to this document.

56 This specification uses these terms:

Term	Definition
Authorization	Process of determining whether an entity has the privilege to perform an action on a protected resource.
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.
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.
Authorization session	A secure session whose privilege levels have been escalated on behalf of either a User or an SPDM endpoint.
Authorization target	A logical entity that determines if the Authorization initiator has the permission(s) and privilege level(s) to access the protected resource.
Byte	Eight-bit quantity. Also known as an *octet*.
Concurrent secure sessions	Simultaneous or parallel secure sessions between an Authorization initiator and an Authorization target.
Credential	Information used to verify the identity of an entity, such as an asymmetric public key.

Term	Definition
Endpoint	Logical entity that communicates with other endpoints over one or more transport protocols.
Message	See Authorization message.
Owner	The user or consumer of the Authorization target operating in an 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.
Protected Resource	A software or hardware resource that requires authorization before being used.
User	An Authorization initiator that is not an SPDM endpoint of the corresponding secure session. A User is identified by a Credential ID.
User-Specific Authorization Session	An Authorization session that is escalated specifically on behalf of a specific User.

57

6 Symbols and abbreviated terms

58 The following additional abbreviations are used in this document.

Abbreviation	Term
AODS	Authorization ODS
AUTH	Authorization
ODS	Opaque Data Structure
SEAP	SPDM Endpoint Authorization Process
SPDM	Security Protocol and Data Model
SVH	Standards body and Vendor-defined Header. See the Standards body or vendor-defined header in DSP0293.
USAP	User-Specific Authorization Process
USAS	User-Specific Authorization Session
VDM	Vendor-Defined Messages

59
7 Notations

60 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` (where `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 characters (`/`) 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 indicates the length of the corresponding field only for that table. The value `X` can be a number greater than 0, as in the case that multiple fields in the same table are using this notation. Note that `LenX` in one table has no connection to a `LenX` with the same value of `X` in any other table. By way of example, consider two hypothetical tables: in the first table, `Len0` has a value of 44 bytes, and in the second table `Len0` has a value of 1024 bytes. These two `Len0`s have no connection to each other. A table could have `Len1` and `Len2` (and so on), and those `Len1` and `Len2` will have no connection to the `Len1`s or `Len2`s in any other table.

61
8 Authorization architecture

62 This authorization architecture serves as a foundation for managing access to a protected resource on an endpoint. The messages and behavior defined by this specification shall apply between two SPDM endpoints within an SPDM session, except when using a trusted environment. The messages are defined in a generic fashion that allows them to be communicated across different physical mediums and over different transport protocols.

63 When messages defined by this specification are exchanged in a trusted environment, such as during initial provisioning, they may be performed outside an SPDM session. The authorization requirements for these messages may also be overridden. The security implications of such exchanges are outside the scope of this specification.

64
8.1 Architecture overview

65 The specification defines message exchanges to enable an entity to have the following capabilities:

- Discover capabilities related to authorization in an endpoint.
- Discover and securely provision credentials and their policies into an endpoint.
- Securely manage endpoint state related to authorization.
- Authorize access to protected resources in an endpoint.

66 A large part of this architecture is the use of an Authorization process to achieve many of the capabilities listed above. There are two Authorization processes: SPDM Endpoint Authorization Process (SEAP) and User-Specific Authorization Process (USAP). SEAP is the process to authorize an SPDM endpoint whereas USAP authorizes an external user.

67 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.

68
8.2 Authorization version

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

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

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

72 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".

73 The `AuthVersion` for this version of this specification shall be `0x10`. The `AuthVersionString` for this version of this specification shall be "1.0".

74 ## 8.3 Authorization flows

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

- Credential provisioning
- Authorization

76 ### 8.3.1 Credential provisioning overview

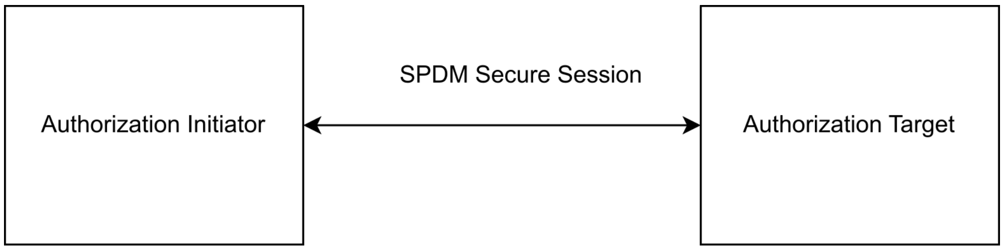
77 Credential provisioning is the process where an endpoint is securely equipped with a credential. In the context of this specification, a credential consists of an asymmetric key pair. The specifics of the key generation are outside the scope of the specification. For an asymmetric credential, the 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. This 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.

78 ### 8.3.2 Authorization overview

79 Authorization is the process by which an Authorization initiator, typically an SPDM endpoint, interacts with another endpoint to gain access to a protected resource. 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, which 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.

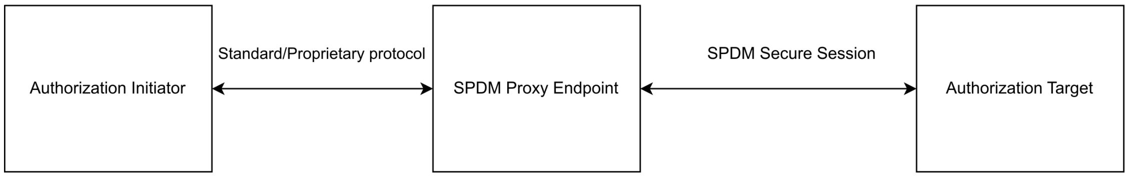
80 Note that the specification does not mandate an Authorization initiator be an SPDM endpoint, however the interactions specified are between two SPDM endpoints. In cases where an Authorization initiator is not an SPDM endpoint, it is expected that an SPDM endpoint acts as a proxy to the initiator to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

81 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.

82 

83 **Figure 1 — Model with SPDM endpoint as Authorization initiator**

84

85 

86 **Figure 2 — Model with external Authorization initiator with SPDM endpoint proxy**

87 8.4 Credentials

88 A credential is a cryptographic secret that identifies the Authorization initiator and allows messages sent by the Authorization initiator to be authenticated as Authorization flows describes.

89 8.4.1 Identifying the Authorization initiator

90 This specification supports more than one Authorization initiator. There can be multiple Authorization initiators at any given time within or across multiple secure sessions. The same Authorization initiator can be in multiple secure sessions. This raises the need to associate a credential and authorization policies with the Authorization initiator to which they belong. This specification uses a numeric identifier, called the Credential ID (`CredentialID`), to make this association. In other words, the Credential ID identifies the Authorization initiator much like a username identifies a person's online account.

91 Furthermore, for the Credential ID to identify a particular Authorization initiator, this architecture presumes the secret portion of a credential is only accessible to the associated Authorization initiator. If this presumption does not hold, then security issues can arise. This presumption allows this specification to use the Credential ID (`CredentialID`) to represent or refer to the associated Authorization initiator.

92 A single Credential ID associates exactly one credential to one Authorization initiator.

93 An Authorization target shall support a minimum of 8 Credential IDs, and these `CredentialID` s shall increase sequentially starting from `0` .

94 ## 8.4.2 Credential structure

95 A single credential structure contains all the credential information relating to a single Authorization initiator that an Authorization target needs to know in order to properly authenticate messages requiring authorization. Depending on the algorithm, credential information may contain secrets. However, for this version of the specification, only asymmetric algorithms are supported and the credential structure needs only public information, such as the public key, the exact asymmetric algorithm used, and other parameters generally deemed public information.

96 Each credential structure uses the `CredentialID` field to associate the credential information with the corresponding Authorization initiator.

97 At a minimum, the Authorization target should store credentials in integrity-protected storage. An endpoint may use the credential structure as defined in this specification or use an implementation-specific data structure to store credentials.

98 The `SET_CRED_ID_PARAMS` request can be used to provision credentials into a credential structure as Credential provisioning defines.

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

100 **Table 2 — Credential structure**

Byte offset	Field	Size (bytes)	Description
0	CredentialID	2	A unique identifier to associate the credential information in this structure with the corresponding Authorization initiator. The value of 0xFFFF shall be reserved unless other parts of this specification define the use for this value.
2	CredentialType	1	Shall be the type of the credential. • 0x01. Asymmetric Key. • All other values reserved. Shall be `0x01` for this version of the specification.
3	BaseAsymAlgo	`BaseAsymAlgoLen`	The format of this field shall be as Table 70 — Base asymmetric algorithm format defines. The value of `BaseAsymAlgoLen` shall be as Common variable names defines. If `CredentialType` is `0x01`, this field shall have exactly one bit set.

Byte offset	Field	Size (bytes)	Description
3 + `BaseAsymAlgoLen`	BaseHashAlgo	`BaseHashAlgoLen`	The format of this field shall be as Table 71 — Base hash algorithm format defines. The value of `BaseHashAlgoLen` shall be as Common variable names defines. If `CredentialType` is `0x01`, this field shall have exactly one bit set.
3 + `BaseAsymAlgoLen` + `BaseHashAlgoLen`	Reserved	4	Reserved
7 + `BaseAsymAlgoLen` + `BaseHashAlgoLen`	CredentialDataSize	4	Size of the `CredentialData` field in bytes.
11 + `BaseAsymAlgoLen` + `BaseHashAlgoLen`	CredentialData	`CredentialDataSize`	When `CredentialType` is `0x01`, the size and format of this field shall be the same size and format as the `SubjectPublicKeyInfo` structure encoded in DER format as specified by RFC 7250.

101 8.4.3 Credential attributes

102 This section discusses various attributes that can be associated 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.

103 8.4.3.1 Locking and unlocking attributes

104 A locked credential cannot be modified by any request for the given Credential ID and its associated policy regardless of authorization or the policy settings of the requesting Credential ID. Consequently, unlocking the credential makes the credential and its associated policy modifiable according to the policy of the requesting Credential ID. Furthermore, a Credential ID can lock and unlock only its own credentials and policy. In other words, a Credential ID cannot unlock or lock the credentials and the associated policies of other Credential IDs.

105 A Credential ID can lock and unlock its own credential and policy only if the `LockUnlockSelfPrivilege` bit is set in its own policy as Authorization policies defines.

106 The Authorization initiator should exercise caution before locking the credential and associated policies of a Credential ID, because recovery of locked credentials and their associated policies is outside the scope of this specification.

107 8.4.4 Credential change requirements

108 When the credential for a given Credential ID is changed, the new credential 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.

109 ## 8.5 Authorization policies

110 Authorization policies specify an Authorization initiator's access privileges to one or more protected resources.

111 All Credential IDs shall be associated with an Authorization policy. Similar to credentials, a single Credential ID associates a set of policies to exactly one Authorization initiator. Except for initial provisioning, a Credential ID shall not be usable for authorization without an associated policy. Each Credential ID has its own instance of policies. A policy can be provisioned for a given Credential ID 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 this specification or use an implementation-specific data structure to store authorization policies.

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

113 **Table 3 — Policy List**

Byte Offset	Field	Size (bytes)	Description
0	CredentialID	2	Shall be the Credential ID of the Authorization initiator.
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.

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

115 **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 DSP0293 defines. The value of `LenSVH` shall be set as Common variable names defines.

Byte Offset	Field	Size (bytes)	Description
`LenSVH`	PolicyVersion	4	This field shall indicate the version of the policy in the `Policy` field associated with the Policy Owner identified in the `PolicyOwnerID` field. The Policy Owner defines the format and values of this field and its association with its policy in the `Policy` field. When the `PolicyOwnerID` is DSP0289 using DMTF-DSP as the `ID` in the SVH, the format of this field shall be the same as Table 25 — VersionNumberEntry definition defines and the value of this field shall be the same as the version of this specification. Because Table 25 only defines bits [15:0], bits [31:16] shall be zero.
4 + `LenSVH`	PolicyLen	2	Shall be the length of `Policy`.
6 + `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.

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

117 **Table 5 — DSP0289 Policy structure**

Byte Offset	Field	Size (bytes)	Description
0	PolicyType	2	`Policy Type` column in Table 6 — DSP0289 Policy Types 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`.

118 ## 8.5.1 DSP0289 Authorization policy

119 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 the `SET_AUTH_POLICY` command to change the policy associated with the Credential ID provided in the request.

120 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.

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

- Privilege. A privilege field type is a bit field where setting a bit grants the ability to perform the corresponding action and clearing the bit revokes the ability to perform the corresponding action.
- Allowable. An allowable field type is a bit field where setting one or more bits allows the use of one or more characteristics (usually configuration parameters) associated with that field.

122 All Authorization initiators can modify their own credentials, limited by their associated Authorization policy. All Authorization initiators can retrieve their own Authorization policy or revoke their own privileges for all fields of Privilege field type.

123 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` field as Table 5 — DSP0289 Policy structure defines.

124 **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.
All other values	Reserved	All other values reserved

125 Table 7 — DSP0289 general policy definitions defines the credential policies for the resources (for example, commands, and actions) that this specification defines.

126 **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 as Table 70 — Base asymmetric algorithm format defines. The value of `BaseAsymAlgoLen` shall be as Common variable names defines. This field reflects the base algorithms the given Credential ID may 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 the given Credential ID cannot be used. The Authorization initiator can set any bit regardless of the supported asymmetric algorithm. The Authorization target shall accept and retain any bit that is set by the Authorization initiator.
`BaseAsymAlgoLen`	AllowedBaseHashAlgo	`BaseHashAlgoLen`	Allowable	The format of this field shall be as Table 71 — Base hash algorithm format defines. The value of `BaseHashAlgoLen` shall be as Common variable names defines. This field reflects the base hash algorithms the given Credential ID may 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. The Authorization initiator can set any bit regardless of the supported hash algorithm. The Authorization target shall accept and retain any bit that is set by the Authorization initiator.
`BaseAsymAlgoLen` + `BaseHashAlgoLen`	CredentialPrivileges	4		The format of this field shall be as Table 8 — DSP0289 Authorization policy bit definitions defines.
4 + `BaseAsymAlgoLen` + `BaseHashAlgoLen`	AuthProcessPrivileges	1		The format of this field shall be as Table 9 — DSP0289 Authorization process policy bit definitions 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.

127 Table 8 — DSP0289 Authorization policy bit definitions defines the credentials provisioning policies.

128 **Table 8 — DSP0289 Authorization policy bit definitions**

Byte Offset	Bit Offset	Field	Field Type	Description
0	0	ModifyOtherCredentialParamPrivilege	Privilege	If this bit is set, the given Credential ID shall be capable of modifying the Credential ID parameters of other Credential IDs through the `SET_CRED_ID_PARAMS` request using the `ParameterChange` operation.
0	1	QueryOtherCredentialParamPrivilege	Privilege	If this bit is set, the given Credential ID shall be capable of retrieving the Credential ID parameters of other Credential IDs through the `GET_CRED_ID_PARAMS` request.
0	2	GrantOtherPolicyPrivilege	Privilege	If this bit is set, the given Credential ID shall be capable of granting privileges for all fields of the Privilege field type for other Credential IDs through the `SET_AUTH_POLICY` request. Also, setting this bit 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 revoking privileges for all fields of the 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 the Authorization policy of other Credential IDs through the `GET_AUTH_POLICY` request.
0	5	ResetToDefaultsPrivilege	Privilege	If this bit is set, the given Credential ID shall be capable of using the `AUTH_RESET_TO_DEFAULT` request.
0	6	LockUnlockSelfPrivilege	Privilege	When this bit is set, the given Credential ID shall be capable of locking or unlocking its own Credential parameters and its policy. Note that this specification does not support a Credential ID being able to lock or unlock the Credential parameters and policies of other Credential IDs.
0	7	RetrieveAuthProcListPrivilege	Privilege	When this bit is set, the given Credential ID shall be capable of retrieving the Authorization process information of other Credential IDs using the `GET_AUTH_PROCESSES` request.
1	0	KillAuthProcPrivilege	Privilege	When this bit is set, the given Credential ID shall be capable of terminating the Authorization process of any unlocked Credential ID using the `KILL_AUTH_PROCESS` request.
1	[7:1]	Reserved	Reserved	Reserved

Byte Offset	Bit Offset	Field	Field Type	Description
2-3	All bits	Reserved	Reserved	Reserved

129 Table 9 — DSP0289 Authorization process policy bit definitions defines the Authorization process policies.

130 **Table 9 — DSP0289 Authorization process policy bit definitions**

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 USAS continuation defines. If this bit is set, the `PrivilegeUSAP` bit shall also be set.
0	[7:3]	Reserved	Reserved	Reserved

131 8.5.1.1 DSP0289 Authorization policy changes requirements

132 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 the `ModifyOtherCredentialParamPrivilege` field, the Authorization target can apply the new settings to incoming messages without ending an active Authorization process. As another example, if a bit is cleared in `AllowedBaseAlgo` and if 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, unless the affected Credential ID changes its own Credential ID parameters to comply with the new policy.

133 Here are some specific policy change requirements. If a new policy clears a bit in an Allowable field type and the current Credential ID parameters associated with that Credential ID use 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.

134 The Authorization target can return an `AUTH_ERROR` message with `ErrorCode=TermAuthProc` for the corresponding `CredentialID` to notify the Authorization initiator that the corresponding Authorization processes are terminated.

135 8.5.1.2 DSP0289 additional Authorization policy requirements

136 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.

137 8.5.2 Policy attributes

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

139 See Locking and unlocking attributes for attribute details applicable to policy.

140 8.6 Initial provisioning

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

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

143 8.6.1 Supply chain provisioning

144 As the Authorization target traverses the many entities involved in the manufacturing and distribution of the Authorization target, which in whole is called the supply chain, each entity may need to provision one or more Credential IDs 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.

145 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 modifiable only 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.

146 Supply chain entities shall not issue the `TAKE_OWNERSHIP` request because this can prevent the Owner from completing its provisioning.

147 8.6.2 Default state

148 The default state is the state of the Authorization target where ownership has not been taken, authorization is not enforced for unlocked Credential IDs, and only locked credentials remain, if any. In this state, supply chain entities are expected to have locked their provisioned credentials and associated policies. The Authorization target uses locked provisioning as an indicator of those Credential IDs provisioned by a supply chain entity that should not be modified by others.

149 8.6.3 Default state and additional supply chain requirements

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

151 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, with the following exceptions:

- The Authorization target shall verify authorization for these messages:
    - All messages requiring authorization that retrieve or modify protected resources associated with the locked Credential ID
    - A `SET_CRED_ID_PARAMS` message when locking or unlocking Credential IDs
    - A `TAKE_OWNERSHIP` request message
- The Authorization target shall bypass authorization verification for Authorization messages described in Credential provisioning and in Authorization policy provisioning and management and for `AUTH_RESET_TO_DEFAULT` messages for unlocked Credential IDs. In these cases, the Authorization target shall not require an Authorization process to occur. In other words, the Authorization target shall fulfill the request without requiring credentials if no other error occurs.

152 8.6.4 Taking ownership

153 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.

154 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 requirements section defines. Once the Owner finishes its initial provisioning, the Authorization initiator shall issue the `TAKE_OWNERSHIP` request to exit the default state and enter an operational state where authorization is fully enforced for all messages.

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

156 Lastly, an Authorization target can return to the default state using the `AUTH_RESET_TO_DEFAULT` request if the requesting Credential ID's Authorization policy permits.

157 8.6.5 Other provisioning considerations

158 This section discusses general provisioning considerations or requirements.

159 Provisioning of credentials and associated policies in the default state or throughout the supply chain should be done only in a trusted environment (such as a secure production sandbox environment or secure manufacturing). If messages used for provisioning are exchanged outside an SPDM session, there are additional factors and any

binding specification that allows such exchanges should ensure there are no gaps in functionality. 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.

160 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 assign privileges across two or more Credential IDs. Furthermore, the Owner can configure privileges in such a way that significantly restricts operation of the Authorization target; recovery from such a state is outside the scope of this specification.

161 8.7 Discovery

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

163 In the announcement phase, an Authorization target announces itself at the start of a secure session, such as the Handshake phase of an SPDM session. For an SPDM 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.

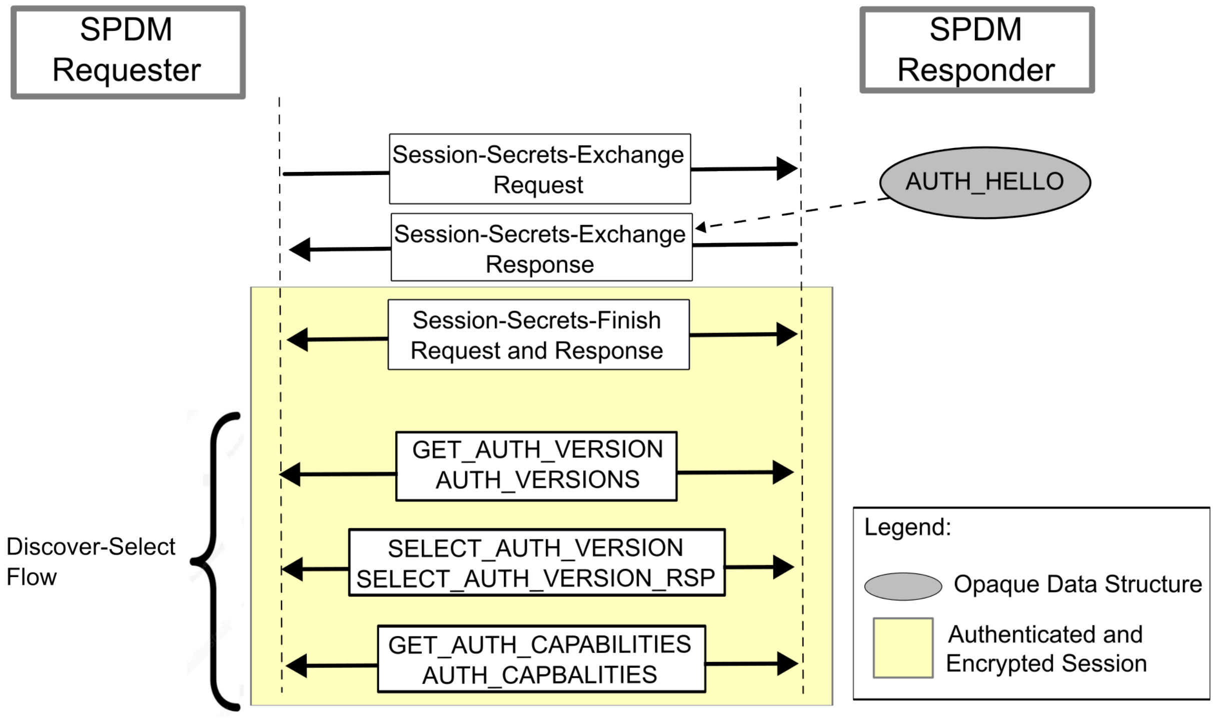
164 The next phase is the Discover-Select Flow phase and this phase only occurs after a secure session is fully established, such as in the Application phase of an SPDM session. If the Authorization initiator receives an `AUTH_HELLO` AODS in an SPDM session, the Authorization initiator can begin this phase by issuing the `GET_AUTH_VERSION` message, followed by the `SELECT_AUTH_VERSION` and ending with `GET_AUTH_CAPABILITIES`. The `GET_AUTH_VERSION` request can be issued at any time and may be skipped if the version information is already known. The `GET_AUTH_CAPABILITIES` request, if issued, shall always follow the successful completion of the `SELECT_AUTH_VERSION` request.

165 The Discover-Select Flow phase does not need to fully complete for every secure session. However, the Authorization initiator shall send a successful `SELECT_AUTH_VERSION` request in every secure session. The Discover-Select Flow phase should fully complete between the Authorization initiator and Authorization target in at least one secure session. Furthermore, for each secure session, the Authorization target shall return an `AUTH_ERROR` response with `ErrorCode=UnexpectedRequest` to all Authorization requests other than `GET_AUTH_VERSION` and `SELECT_AUTH_VERSION` requests until successfully fulfilling a `SELECT_AUTH_VERSION` request for the corresponding secure session.

166 The Authorization initiator should only send exactly one successful `SELECT_AUTH_VERSION` request. If the Authorization target receives additional `SELECT_AUTH_VERSION` requests after a version is already selected and the additional requests change the selected version, the Authorization target shall respond with an `AUTH_ERROR` of `ErrorCode=InvalidRequest`. Otherwise, it shall respond with `SELECT_AUTH_VERSION_RSP` because the subsequent requests could be a retry.

167 Figure 3 — Most common discovery phase illustrates the most common discovery methodology for an SPDM Responder that is an Authorization target.

168



169 **Figure 3 — Most common discovery phase**

170 8.8 Authorization process

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

172 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.

173 To properly prepare for the execution of the Authorization process, an Authorization initiator shall successfully establish a secure session such as an SPDM session as DSP0274 defines or use an already established secure session.

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

- User-Specific Authorization Process
- SPDM Endpoint Authorization Process

175 8.8.1 User-Specific Authorization Process (USAP)

176 The User-Specific Authorization process occurs completely within a secure session. This process establishes an

authorization session bound to the user. Thus, one or more User-specific authorization sessions can occur simultaneously within a secure session, and the Authorization session identifier shall be the Credential ID of the corresponding User.

177 The USAP starts with the Discovery-Select flow as Discovery defines. To ensure the Authorization initiator has current information, in each secure session the Authorization initiator should perform the Discovery-Select flow completely before the first User-Specific Authorization session.

178 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:

- It elevates the privilege level of the secure session for that specific User. This portion of a secure session is called an Authorization session.
- It initializes critical cryptographic parameters for the authorization session. Messages that traverse the authorization session can be messages of any protocol and are not restricted to SPDM or Authorization messages.
- It enables the message format of all messages using the authorization session to accommodate authorization data for the corresponding User. The format for such messages is defined in Authorization record.

179 The successful completion of the `START_AUTH` request and `START_AUTH_RSP` response establishes the Authorization session for the corresponding User. While the authorization session is active, messages requiring authorization shall contain authorization data, called the Authorization tag, for the corresponding User. When the Authorization target receives a message from any protocol in the corresponding secure session, the Authorization target shall determine whether the message requires authorization regardless of whether the message contains an Authorization tag. If a message requires authorization, the Authorization target shall validate the Authorization tag according to the provisioned credentials, 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 if the validation of the Authorization tag fails, the Authorization target shall take one of these actions:

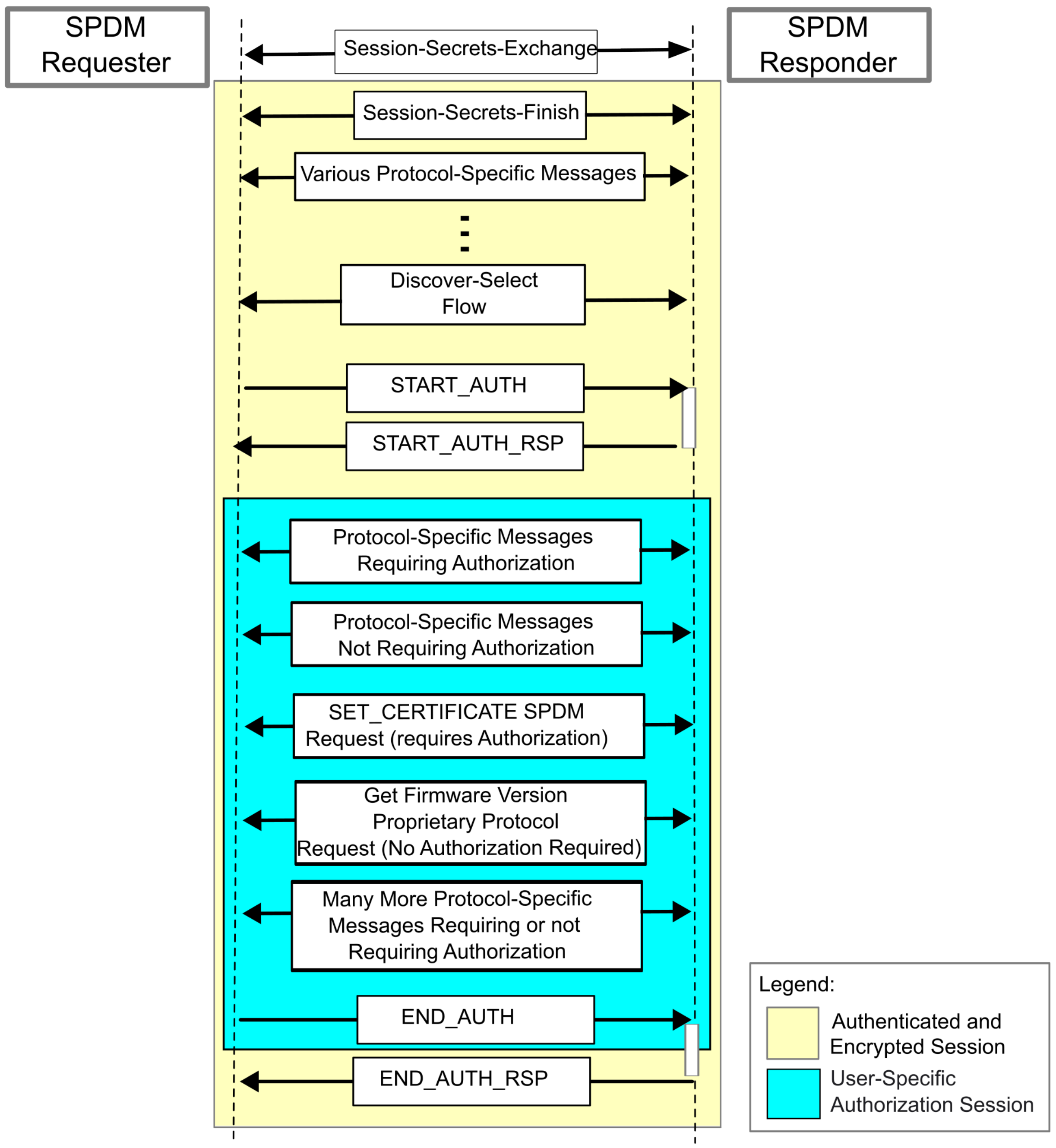
- Respond with an `AUTH_ERROR` message
- Respond with the corresponding protocol-specific error
- Silently discard the message

180 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.

181 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 secure session terminates. The termination of the Authorization session restores a secure session to its original privilege level for that User. Additionally, the termination of a User-Specific Authorization session does not end the corresponding secure session. 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.

182 Figure 4 — Authorization process illustrates an example of the User-Specific Authorization process using an SPDM session.

183


SPDM Requester
SPDM Responder
Session-Secrets-Exchange
Session-Secrets-Finish
Various Protocol-Specific Messages
Discover-Select Flow
START_AUTH
START_AUTH_RSP
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_AUTH
END_AUTH_RSP
Legend:
Authenticated and Encrypted Session
User-Specific Authorization Session

184 **Figure 4 — Authorization process**

185 **8.8.1.1 General USAP error handling, requirements, and notes**

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

187 A User shall have only one Authorization session active at a time within its corresponding secure session. 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. If a `START_AUTH` is received for a User with a corresponding active User-Specific Authorization Session, the Authorization target shall either respond with an `AUTH_ERROR` or silently discard the request.

188 A User can repeat the User-Specific Authorization Process as many times as it deems necessary as long as each iteration of the process starts and ends as User-Specific Authorization Process defines. Additionally, the Authorization target can limit the number of simultaneous active User-Specific Authorization sessions for a given secure session.

189 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.

190 8.8.1.2 USAS continuation

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

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

- Authorization initiator nonce
- Authorization target nonce
- `SavedSequenceNumber`
- Credential ID

193 The `SavedSequenceNumber` shall be calculated as: the last-used sequence number in the USAP Authorization tag 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.

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

195 If a saved USAS cannot be continued for any reason, the Authorization target shall still preserve the USAS according to its original persistence method. The preserved USAS shall be saved until the Authorization target receives a `KILL_AUTH_PROCESS` request from the Authorization initiator. To ensure maximum compatibility, the Authorization initiator should select the same Authorization version as the saved USAS for the corresponding secured session.

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

197 The `PrivilegePersistUSAS` privilege governs the ability of USAS continuation for the given Credential ID.

198 8.8.2 SPDM Endpoint Authorization Process (SEAP)

199 The SPDM Endpoint Authorization Process (SEAP) is a process that specifically authorizes an SPDM Requester or both SPDM endpoints in an SPDM 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 an Authorization target at any time within the session.

200 SEAP requires SPDM mutual authentication as SPDM defines. This version of the specification only supports asymmetric algorithms and, therefore, SEAP supports secure session establishment only through SPDM `KEY_EXCHANGE`. Additionally, SPDM mutual authentication can use certificates or a raw public key.

201 SEAP is broken into two parts as Figure 5 — SPDM Endpoint Authorization Process (SEAP) illustrates. The first part occurs during the Session handshake phase as SPDM defines. The second part occurs during the SPDM Application phase.

202 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-Secrets-Exchange response. Lastly, the SPDM endpoints shall populate all fields appropriately in a Session-Secrets-Exchange request and response message to perform mutual authentication.

203 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-Secrets-Finish response. Otherwise, if there is a failure or if the `OpaqueData` field does not exist, the `SEAP_SUCCESS` data structure in either the request or the response depending on which endpoint failed shall be absent. A failure of the SEAP process does not end the SPDM session.

204 A matching Credential ID has three different definitions depending on ownership and lock Credential ID status. They are as follows:

- Ownership Taken: A matching Credential ID is a supported Credential ID whose credential parameters and policies are properly provisioned and whose public key matches that of the leaf certificate or raw public key of the Authorization initiator used to authenticate the Authorization initiator.
- Ownership Not Taken: A matching Credential ID is a supported Credential ID whose credential parameters and policies may not be properly or fully provisioned.
- Locked Credential IDs. A matching Credential ID is the same definition as a matching Credential ID in the Ownership Taken case regardless of ownership status.

205 Before the second part of SEAP can begin, the Authorization initiator performs the Discovery-Select flow as the Discovery section defines. The Authorization initiator should completely perform the Discovery-Select flow before the

second part of SEAP in each SPDM session to ensure the Authorization initiator has the current 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.

206 The second part of SEAP can begin at any time during 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 an `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 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, if the `SEAP_SUCCESS` was absent for the corresponding Authorization initiator).

207 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.

208 The successful completion of this request and response 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 take one of these actions:

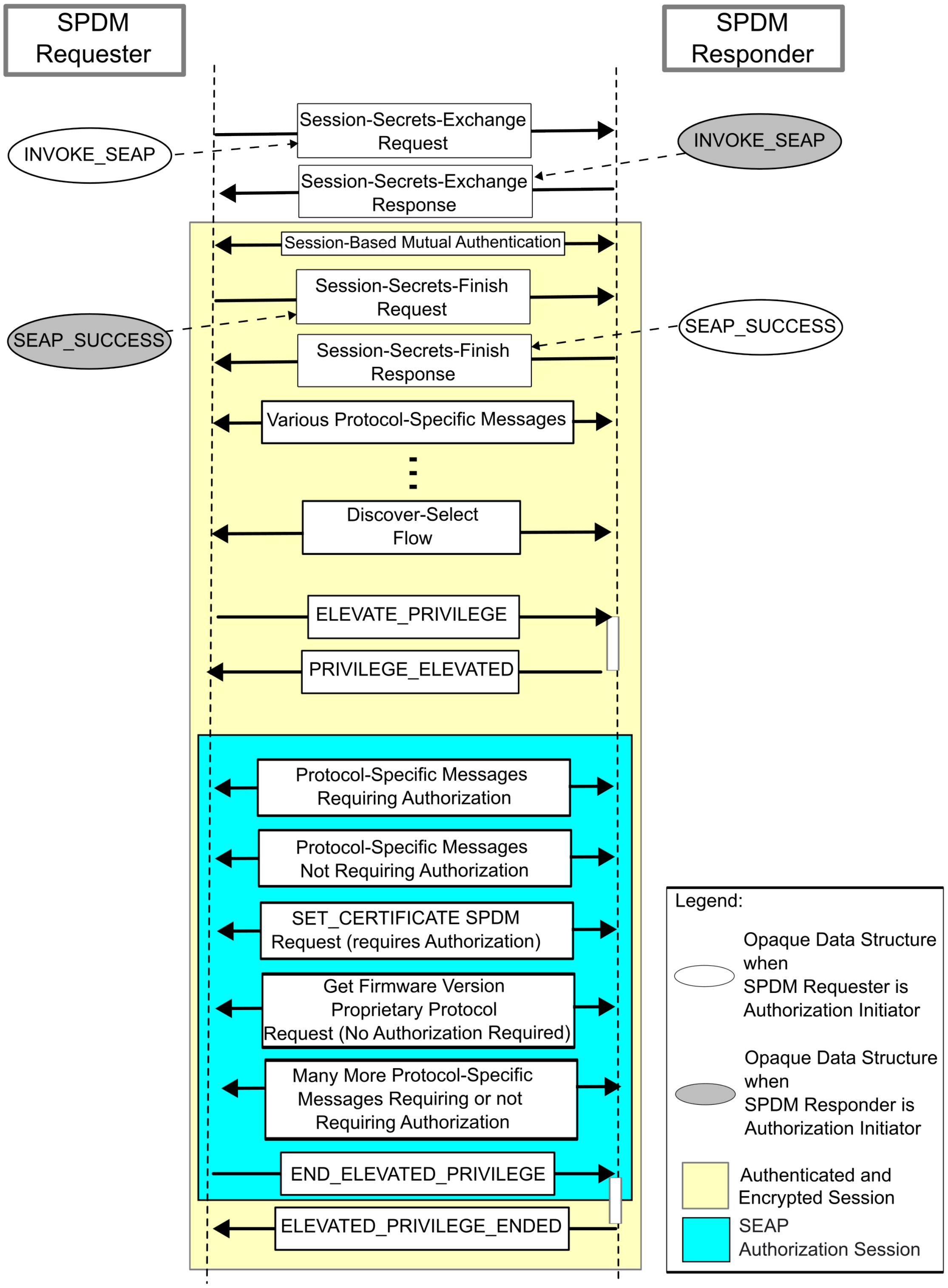
- Respond with an `AUTH_ERROR` message
- Respond with the corresponding protocol-specific error
- Silently discard the message

209 For messages that do not require authorization, the Authorization target can process the message accordingly.

210 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. The termination of a SEAP Authorization session does not end the corresponding SPDM session. 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.

211 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.

212


SPDM Requester
SPDM Responder
INVOKE_SEAP
Session-Secrets-Exchange Request
INVOKE_SEAP
Session-Secrets-Exchange Response
Session-Based Mutual Authentication
Session-Secrets-Finish Request
SEAP_SUCCESS
SEAP_SUCCESS
Session-Secrets-Finish Response
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_PRIVILEGE
ELEVATED_PRIVILEGE_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

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

214 **8.8.2.1 SEAP error handling, requirements, and notes**

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

216 If SEAP uses SPDM version 1.3 or earlier, then `SEAP_SUCCESS` cannot be supported because there is no `OpaqueData` field in the Session-Secrets-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.

217 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.

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

219 8.8.3 Terminating Authorization process

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

221 Note that other parts of this specification use forced termination.

222 In cases that do not preserve an Authorization process or that kill 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 fail authorization checks. The affected Credential ID can start a new Authorization process afterward.

223 8.8.4 Other error handling, requirements, and notes

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

- Have the Authorization target use 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

225 Authorization sessions do not limit the types of messages that can traverse a secure session, but rather they enable 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.

226 The use of the same Credential ID across multiple secure 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 secure session and Authorization session. In other words, the authorization data cannot be reused in another secure session. Here is a small example involving SPDM sessions: the sequence number, Authorization tag, or nonce that is bound to SPDM session ID 33 cannot be used again in SPDM session ID 88.

227 8.9 Authorization record

228 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 with a protocol-agnostic way to send and receive messages requiring authorization.

229 Table 10 — Authorization record format shows the Authorization record format:

230 **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 11 — 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 as specified by the `AuthRecordType` .

231 Table 11 — Authorization record types shows the supported Authorization record types.

232 **Table 11 — Authorization record types**

Value	Description
0	Authorization message. The `GenericPayload` field shall contain an Authorization message that this specification defines and which 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 Table 12 — Generic Authorization Record Type Format for Messages Requiring Authorization. When using DSP0277 as the transport, the format and size of the `MsgToAuthPayload` field shall be the same as the Application data, as DSP0277 defines. Otherwise, the format and size of the `MsgToAuthPayload` field are specific to the message protocol or the transport.

Value	Description
2	Record Error. The `GenericPayload` field shall contain data in the format specified by Table 13 — 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.
3	DSP0289-defined Authorization messages requiring authorization. The `GenericPayload` field shall contain data in the format specified by Table 12 — Generic Authorization Record Type Format for Messages Requiring Authorization. Additionally, the size and format of the `MsgToAuthPayload` field in the Message Requiring Authorization Record shall be the same format and size as an Authorization message. This Authorization record type is strictly for Authorization messages requiring authorization.
All other values	Reserved

233 8.9.1 Authorization record on the transport

234 While the Authorization record can traverse any transport, there are some requirements the transport should define. The transport should 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. For example, this can be accomplished through a single bit indicating presence or by stating that the Authorization record is always present. The transport can also choose to use the mechanism defined in Authorization record over SPDM Vendor-Defined Messages (VDM) to transmit the Authorization record since this may help prevent significant modifications to the transport.

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

236 8.9.2 Authorization types

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

238 8.9.2.1 Authorization record in Authorization process

239 This section gives additional details on the Authorization records specific to each Authorization process.

240 8.9.2.1.1 USAP Authorization record

241 This section defines requirements for all messages requiring authorization in USAP. These are the requirements for all messages requiring authorization in USAP:

- The Authorization record shall be present for all messages requiring authorization.
- The Authorization record shall be transmitted exclusively from the Authorization initiator to the Authorization

target, and transmission in the opposite direction is prohibited.

- 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 `AuthRecordType` shall be set to 0.

242 Table 12 — Generic Authorization Record Type Format for Messages Requiring Authorization shows the format for the `GenericPayload` field when `AuthRecordType` is 1:

243 **Table 12 — Generic Authorization Record Type Format for Messages Requiring Authorization**

Byte offset	Field	Size (bytes)	Description
0	AuthRecID	4	This field indicates a unique number for this Authorization record. The Authorization endpoints use this number for message tracking and error-handling purposes. The value of this field should increment by 1. 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`	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.

244 8.9.2.1.2 SEAP Authorization record

245 The transport shall specify its use of an Authorization record.

246 8.9.2.2 Authorization record Failures

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

248 **Table 13 — 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.
4	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 use only certain types of `ErrorCode` s.

249 8.10 Authorization tag

250 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 in every message. The Authorization record embeds the Authorization tag. This section details the Authorization tag for each Authorization process.

251 Furthermore, Authorization tags support only asymmetric signature algorithms.

252 8.10.1 SEAP Authorization tag

253 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.

254 8.10.2 USAP Authorization tag

255 This section provides details about the Authorization tag in a USAP.

256 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.

257 8.10.2.1 USAP Authorization tag format

258 The format and size for the `AuthTag` in the Authorization record shall be the format and size as Table 14 — Authorization tag format defines.

259 Table 14 — Authorization tag format shows the format for the USAP Authorization tag.

260 **Table 14 — Authorization tag format**

Byte offset	Field	Size (bytes)	Description
0	CredentialID	2	Shall be the Credential ID of an active User-Specific Authorization session.

Byte offset	Field	Size (bytes)	Description
2	Signature	Len0	Shall be the signature of the 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` .

261 If `CredentialID` is present in the Authorization record, the Authorization target shall use it to locate the credential in order to verify the Authorization tag.

262 8.10.2.2 USAP Authorization tag signature generation and verification

263 This section defines the operations for signature generation and verification when using asymmetric signature algorithms for USAP.

264 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.

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

1. The Credential ID of the User
2. The requester's nonce provided in the `START_AUTH` request
3. The responder's nonce provided in the `START_AUTH_RSP` response
4. The sequence number
5. The message body, which is the `MsgToAuthPayload` field of the Authorization record

266 If `[START_AUTH]` . `Attributes` / `Continue` is set, the sequence number shall start with `SavedSequenceNumber` as USAS 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 1 after each message requiring authorization and corresponding to the User. For the Authorization target, the sequence number shall increment by 1 after receiving a message containing an Authorization tag from the corresponding User regardless of whether the Authorization verification succeeds or fails.

267 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.

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

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

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

270 where:

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

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

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

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

273 where:

- The `UserPublicKey` shall be the public key associated with the User that is associated with the corresponding Credential ID.
- The `AuthSignature` shall be the signature in the Authorization tag that accompanied the message.
- The `context` shall be the string "usap signing".

274 If `AuthValResult` is success, the Authorization tag has been successfully validated. Otherwise, the Authorization tag has failed validation.

275 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.

276 Otherwise, the message fails authorization.

277 9 Authorization messages

278 9.1 Authorization messages overview

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

280 9.1.1 Bi-directional Authorization message processing

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

282 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.

283 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 is outside the scope of this specification.

284 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.

285 9.1.2 Requirements for Authorization initiators

286 An Authorization initiator shall not have multiple outstanding Authorization requests to the same Authorization target, within a single secure 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.

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

288 Within a secure 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.

289 An Authorization initiator might send simultaneous request messages to the same Authorization target across multiple secure sessions or to different Authorization targets.

290 9.1.3 Requirements for Authorization targets

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

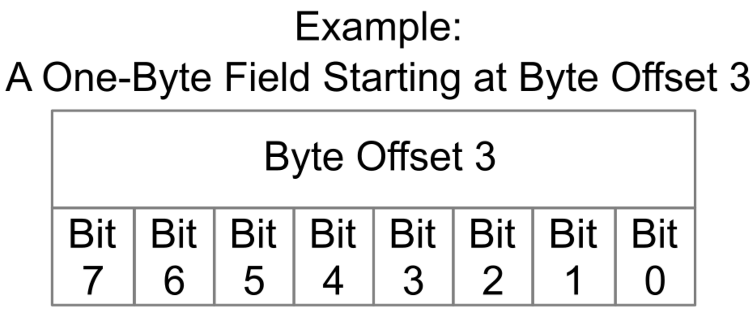
292 An Authorization target that is not ready to accept a new request message shall either respond with an `AUTH_ERROR` message of `ErrorCode=Busy` or silently discard the request message.

293 If an Authorization target supports Authorization messages across concurrent secure sessions, a pending request in one session shall not affect pending requests in another session.

294 9.1.4 Authorization messages bits-to-bytes mapping

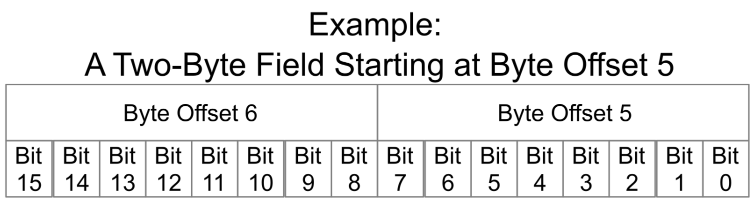
295 All fields, regardless of size or endianness, map the highest numeric bits to the highest numerically assigned byte in sequentially decreasing order down to and including the least numerically assigned byte of that field. The following two figures illustrate this mapping.

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

297


298 **Figure 6 — One-byte field bit map**

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

300


301 **Figure 7 — Two-byte field bit map**

302 9.1.5 Version encoding

303 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.

304 For example:

- Version 1.0 would be `0x10`.
- Version 1.2 would be `0x12`.
- Version 3.7 would be `0x37`.

305 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.

306 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`.

307 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:

- Both endpoints shall use the same lower version number in the `AuthVersion` field for all messages.
- Functionality shall be limited to what the lower minor version of the authorization specification defines.
- 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 changes in computations or other operations. These differences are dependent on the value in the `AuthVersion` field in the message.
- 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.
- Errata versions (indicated by a non-zero value in the `UpdateVersionNumber` field of the `AUTH_VERSION` response message after a `GET_AUTH_VERSION` request) 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.

308 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 messages. 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 24 — AUTH_VERSION response message format.

309 ### 9.1.6 Generic Authorization message format

310 Table 15 — Generic Authorization message field definitions defines the fields that constitute a generic Authorization message, including the message header and payload:

311 **Table 15 — Generic Authorization message field definitions**

Byte offset	Bit offset	Size (bits)	Field	Description
0	[7:0]	8	RequestResponseCode	Shall be the request message code or response code, which Table 16 — Authorization message request codes and Table 17 — 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 `RequestResponseCode`.

312 ## 9.2 Authorization message definitions

313 This section discusses all authorization request and response messages.

314 ### 9.2.1 Authorization message request codes

315 Table 16 — Authorization message request codes defines the Authorization message request codes. The **Implementation requirement** column indicates requirements on the Requester.

316 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 an `AUTH_ERROR` using `ErrorCode=AccessDenied`.

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

318 **Table 16 — Authorization message request codes**

Request	Code value	Implementation requirement	Authorization Requirements	Message format
GET_AUTH_VERSION	0x81	Mandatory	None	Table 23 — GET_AUTH_VERSION request message format
SELECT_AUTH_VERSION	0x82	Mandatory	None	Table 26 — SELECT_AUTH_VERSION request message format
SET_CRED_ID_PARAMS	0x83	Optional	Conditional	Table 33 — SET_CRED_ID_PARAMS request message format
GET_CRED_ID_PARAMS	0x84	Mandatory	Conditional	Table 36 — GET_CRED_ID_PARAMS request message format
SET_AUTH_POLICY	0x85	Optional	Conditional	Table 39 — SET_AUTH_POLICY request message format
GET_AUTH_POLICY	0x86	Mandatory	Conditional	Table 42 — GET_AUTH_POLICY request message format
START_AUTH	0x87	Optional	None	Table 49 — START_AUTH request message format
END_AUTH	0x88	Optional	None	Table 52 — END_AUTH request message format
ELEVATE_PRIVILEGE	0x89	Optional	None	Table 55 — ELEVATE_PRIVILEGE request message format
END_ELEVATED_PRIVILEGE	0x8A	Optional	None	Table 57 — END_ELEVATED_PRIVILEGE request message format
GET_AUTH_CAPABILITIES	0x8B	Mandatory	None	Table 28 — GET_AUTH_CAPABILITIES request message format
AUTH_RESET_TO_DEFAULT	0x8C	Optional	Conditional	Table 61 — AUTH_RESET_TO_DEFAULT request message format

Request	Code value	Implementation requirement	Authorization Requirements	Message format
TAKE_OWNERSHIP	0x8D	Mandatory	Mandatory	Table 59 — TAKE_OWNERSHIP request message format
GET_AUTH_PROCESSES	0x8E	Optional	Mandatory	Table 44 — GET_AUTH_PROCESSES request message format
KILL_AUTH_PROCESS	0x8F	Optional	Mandatory	Table 47 — 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.

319 **9.2.2 Authorization message response codes**

320 The `RequestResponseCode` field in the Authorization response message shall specify the appropriate response code for a request.

321 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.

322 Table 17 — Authorization message response codes defines the response codes for Authorization messages. The **Implementation requirement** column indicates requirements on the Responder.

323 **Table 17 — Authorization message response codes**

Response	Code value	Implementation requirement	Message format
AUTH_VERSION	0x01	Mandatory	Table 24 — AUTH_VERSION response message format
SELECT_AUTH_VERSION_RSP	0x02	Mandatory	Table 27 — SELECT_AUTH_VERSION_RSP response message format
SET_CRED_ID_PARAMS_DONE	0x03	Optional	Table 35 — SET_CRED_ID_PARAMS_DONE response message format
CRED_ID_PARAMS	0x04	Mandatory	Table 37 — CRED_ID_PARAMS response message format

Response	Code value	Implementation requirement	Message format
SET_AUTH_POLICY_DONE	0x05	Optional	Table 41 — SET_AUTH_POLICY_DONE response message format
AUTH_POLICY	0x06	Mandatory	Table 43 — AUTH_POLICY response message format
START_AUTH_RSP	0x07	Optional	Table 51 — START_AUTH_RSP response message format
END_AUTH_RSP	0x08	Optional	Table 53 — END_AUTH_RSP response message format
PRIVILEGE_ELEVATED	0x09	Optional	Table 56 — PRIVILEGE_ELEVATED response message format
ELEVATED_PRIVILEGE_ENDED	0x0A	Optional	Table 58 — ELEVATED_PRIVILEGE_ENDED response message format
AUTH_CAPABILITIES	0x0B	Mandatory	Table 29 — AUTH_CAPABILITIES response message format
AUTH_DEFAULTS_APPLIED	0x0C	Optional	Table 64 — AUTH_DEFAULTS_APPLIED response message format
OWNERSHIP_TAKEN	0x0D	Mandatory	Table 60 — OWNERSHIP_TAKEN response message format
AUTH_PROCESSES	0x0E	Optional	Table 45 — AUTH_PROCESSES response message format
PROCESS_KILLED	0x0F	Optional	Table 48 — PROCESS_KILLED response message format
AUTH_ERROR	0x7F	Mandatory	Table 20 — AUTH_ERROR response message format
Reserved	All other values	Reserved	Authorization implementations compatible with this version shall not use the reserved response codes.

324 9.2.3 Authorization Message Validity

325 Certain Authorization messages are associated with a specific Authorization process. Table 18 — Authorization

message and Authorization process association shows this high-level association. More specific information can be found in the specific message's section.

326 **Table 18 — Authorization message and Authorization process association**

Message	Authorization Process
SET_AUTH_POLICY/START_AUTH_RSP	USAP
END_AUTH/END_AUTH_RSP	USAP
ELEVATE_PRIVILEGE/PRIVILEGE_ELEVATED	SEAP
END_ELEVATED_PRIVILEGE/ELEVATED_PRIVILEGE_ENDED	SEAP
All other Authorization messages	None

327 9.2.4 Common variable names

328 This section defines some frequent variable names used in various Authorization messages. Table 19 — Common variables used in Authorization messages defines these variable names.

329 **Table 19 — 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 DSP0293 defines.

330 9.2.5 Error handling

331 This section discusses general error handling for all Authorization messages.

332 9.2.5.1 AUTH_ERROR response message

333 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.

334 Table 20 — AUTH_ERROR response message format shows the `AUTH_ERROR` response format.

335 Table 21 — 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 indicates that the `ErrorCode` shall be allowed in response to an Authorization request. A value of **R** indicates that the `ErrorCode` shall be allowed in a Type 2 Authorization record. More than one value can be present in the **Layer** column for an `ErrorCode`, in which case they are comma separated.

336 **Table 20 — AUTH_ERROR response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `AUTH_ERROR` in Table 17 — Authorization message response codes.
1	Reserved	1	Reserved
2	ErrorCode	1	Shall be the ErrorCode. See Table 21 — Error code and error data.
3	ErrorData	1	Shall be the Error data. See Table 21 — Error code and error data.
4	ExtendedErrorData	0-32	Shall be the Extended error data. See Table 21 — Error code and error data.

337 **Table 21 — 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 target 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.

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

ErrorCode	Value	Layer	Description	Error data	ExtendedErrorData
TermAuthProc	0x0B	M	The User or Authorization initiator associated with the Credential ID in Error data should terminate their active Authorization process, if any. The Authorization target has already terminated Authorization processes associated with the given Credential ID. Further use of the active Authorization processes associated with the given Credential ID can result in access denials to the protected resource. The User or Authorization initiator can start new Authorization processes.	`0x00`	The format and size of this field shall be the same as `CredentialID` field as Table 2 — Credential structure defines. A value of `0xFFFF` shall indicate all Credential IDs.
Vendor or Standards-Defined	0xFF	M, R	Vendor or standards-defined	Shall indicate the registry or standards body using one of the values in the `ID` column of Table 1 — Registry or standards body ID in DSP0293.	See Table 22 — ExtendedErrorData format for vendor or standards defined ERROR response message.
Reserved	All other values		Reserved.	Reserved	Reserved

338 **Table 22 — ExtendedErrorData format for vendor or standards defined ERROR response message**

Byte offset	Field	Size (bytes)	Description
0	VendorIDLen	1	Shall be the `VendorIDLen` as defined by DSP0293 Table 2 — Standards body or vendor-defined header (SVH).

Byte offset	Field	Size (bytes)	Description
1	VendorID	`VendorIDLen`	Shall be the `VendorID` as defined by DSP0293 Table 2 — Standards body or vendor-defined header (SVH).
1 + `VendorIDLen`	OpaqueErrorData	Variable	The vendor or standards body defines this value.

339 ## 9.2.6 Discovery message

340 Messages in this section allow an Authorization initiator to discover aspects of the Authorization target. These aspects provide basic information to understand support and establish basic communication parameters.

341 ### 9.2.6.1 GET_AUTH_VERSION request and AUTH_VERSION response messages

342 This request message shall retrieve the authorization specification version of an endpoint. Table 23 — GET_AUTH_VERSION request message format shows the `GET_AUTH_VERSION` request message format and Table 24 — AUTH_VERSION response message format shows the `AUTH_VERSION` response message format.

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

344 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 secure 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 23 — GET_AUTH_VERSION request message format describes.

345 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 versions in all future communication of other requests. 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.

346 Table 23 — GET_AUTH_VERSION request message format shows the `GET_AUTH_VERSION` request message format:

347 **Table 23 — GET_AUTH_VERSION request message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `GET_AUTH_VERSION` in Table 16 — Authorization message request codes.
1	Reserved	1	Reserved

348 Table 24 — AUTH_VERSION response message format shows the successful `AUTH_VERSION` response message format:

349 **Table 24 — AUTH_VERSION response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `AUTH_VERSION` in Table 17 — Authorization message response codes.
1	Reserved	1	Reserved
2	VersionNumberEntryCount	1	Number of version entries in `VersionNumberEntries` .
3	VersionNumberEntries	2 * `VersionNumberEntryCount`	16-bit version entry. See Table 25 — VersionNumberEntry definition. Each entry should be unique. The number of entries in this field shall be the same value as `VersionNumberEntryCount` . The versions in this field shall be in ascending order sorted by `MajorVersion` , `MinorVersion` , `UpdateVersionNumber` , and `Alpha` .

350 Table 25 — VersionNumberEntry definition shows the `VersionNumberEntry` definition. See Version encoding for more details.

351 **Table 25 — 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`).

352 9.2.6.2 SELECT_AUTH_VERSION request and SELECT_AUTH_VERSION_RSP response messages

353 The `SELECT_AUTH_VERSION` request shall 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 26 — SELECT_AUTH_VERSION request message format and Table 27 — SELECT_AUTH_VERSION_RSP response message format. The version selected

using this request applies only to the secure session in which the message was sent and is valid until the session terminates. See Discovery section for additional requirements.

354 The selected version for communication with an Authorization target shall be the version in the `AuthVersion` field of the `SELECT_AUTH_VERSION`. The `AuthVersion` shall be one of the supported versions of an Authorization target. Otherwise, the Authorization target shall either return an `AUTH_ERROR` message of `ErrorCode=VersionMismatch` or silently discard the request.

355 In all future authorization versions, the `SELECT_AUTH_VERSION` and `SELECT_AUTH_VERSION_RSP` response messages will be backward compatible with all earlier versions.

356 Table 26 — SELECT_AUTH_VERSION request message format shows the `SELECT_AUTH_VERSION` request message format:

357 **Table 26 — 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 16 — 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, as Version encoding describes.

358 Table 27 — SELECT_AUTH_VERSION_RSP response message format shows the successful `SELECT_AUTH_VERSION_RSP` response message format:

359 **Table 27 — 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 17 — Authorization message response codes.
1	Reserved	1	Reserved

360 9.2.6.3 GET_AUTH_CAPABILITIES request and AUTH_CAPABILITIES response messages

361 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 28 — GET_AUTH_CAPABILITIES request message format and Table 29 — AUTH_CAPABILITIES response message format respectively. This request can be sent multiple times and should be sent as the Discovery section describes. If the request is sent multiple times in the same secure session, the corresponding responses shall be identical to the first.

362 Table 28 — GET_AUTH_CAPABILITIES request message format shows the `GET_AUTH_CAPABILITIES` request message format:

363 **Table 28 — 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 16 — Authorization message request codes.
1	Reserved	1	Reserved

364 Table 29 — AUTH_CAPABILITIES response message format shows the successful `AUTH_CAPABILITIES` response message format:

365 **Table 29 — AUTH_CAPABILITIES response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `AUTH_CAPABILITIES` in Table 17 — Authorization message response codes.
1	Reserved	1	Reserved
2	MessageCaps	2	The format of this field shall be as Table 30 — Message supported bit definitions defines.
4	AuthProcessCaps	2	The format of this field shall be as Table 31 — Authorization process supported bit definitions defines.
6	DeviceProvisioningState	1	The format of this field shall be as Table 32 — Device provisioning state values defines.
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.

Byte offset	Field	Size (bytes)	Description
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 as Table 70 — Base asymmetric algorithm format defines. The value of `BaseAsymAlgoLen` shall be as Common variable names 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 as Table 71 — Base hash algorithm format defines. The value of `BaseHashAlgoLen` shall be as Common variable names 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.
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. 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.

366 Table 30 — Message supported bit definitions defines the messages the Authorization endpoint supports.

367 **Table 30 — 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.

Byte Offset	Bit Offset	Field	Description
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	5	ResetToDefaultCap	If the Authorization target supports `AUTH_RESET_TO_DEFAULT` , then this bit shall be set. Otherwise, this bit shall not be set.
0	[7:6]	Reserved	Reserved
1	[7:0]	Reserved	Reserved

368 Table 31 — Authorization process supported bit definitions defines the Authorization processes the Authorization endpoint supports.

369 **Table 31 — 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, the `START_AUTH_RSP` and `END_AUTH_RSP` response messages 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, the `PRIVILEGE_ELEVATED` and `ELEVATED_PRIVILEGE_ENDED` response messages shall be supported.
0	2	ResetPersistCap	If the Authorization target supports USAS 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 USAS 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

Byte Offset	Bit Offset	Field	Description
1	[7:0]	Reserved	Reserved

370 **Table 32 — Device provisioning state values**

Value	Name	Description
0	Unprovisioned	Device does not have any credentials provisioned.
1	DefaultState	Device is in the default state. See Default state for details.
2	Owned	Device has had ownership taken via `TAKE_OWNERSHIP` . See Taking ownership for additional details.
All other values	Reserved	Reserved

371 9.2.7 Credential provisioning

372 9.2.7.1 SET_CRED_ID_PARAMS request and SET_CRED_ID_PARAMS_DONE response messages

373 The `SET_CRED_ID_PARAMS` request shall be used to provision credentials into an Authorization target, as described in the Credentials section. When `CredParams` provides an invalid credential type, Credential ID or algorithm, the Authorization target shall respond with `AUTH_ERROR` and `ErrorCode=InvalidRequest` .

374 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 `CredParams` , and fail if that is not possible. When `CredParams` provides an invalid Credential ID or other invalid values, the Authorization target shall respond with `AUTH_ERROR` and `ErrorCode=InvalidRequest` . When `SetCredInfoOp` is valid but authorization checks fail, the Authorization target shall respond with `AUTH_ERROR` and `ErrorCode=AccessDenied` .

375 Table 33 — SET_CRED_ID_PARAMS request message format shows the `SET_CRED_ID_PARAMS` request message format:

376 **Table 33 — 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 16 — Authorization message request codes.
1	Reserved	1	Reserved
2	SetCredInfoOp	1	The field indicates the requested operation. The format of this field shall be as Table 34 — Values for `SetCredInfoOp` field defines.

Byte offset	Field	Size (bytes)	Description
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. If `SetCredInfoOp` field indicates a lock or unlock operation, the format and size of this field shall be the same format and size as the `CredentialID` field defined in Table 2 — Credential structure. Additionally, the value of the `CredentialID` shall be the same as the Credential ID of the Requester.

377 **Table 34 — 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

378 Table 35 — SET_CRED_ID_PARAMS_DONE response message format shows the successful `SET_CRED_ID_PARAMS_DONE` response message format:

379 **Table 35 — 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 17 — Authorization message response codes.
1	Reserved	1	Reserved

380 **9.2.7.1.1 Additional requirements on SET_CRED_ID_PARAMS**

381 When locking or unlocking, the requested Credential ID shall only be capable of locking its own credential parameters and associated policy if the `LockUnlockSelfPrivilege` policy bit is set. See Locking and unlocking attributes and DSP0289 Authorization policy for additional requirements.

382 **9.2.7.2 GET_CRED_ID_PARAMS request and CRED_ID_PARAMS response messages**

383 The `GET_CRED_ID_PARAMS` request shall be used to retrieve information about credentials provisioned in a credential structure. If the request contains an invalid Credential ID or the corresponding credential structure is not provisioned, the Authorization target shall respond with `AUTH_ERROR` and `ErrorCode=InvalidRequest`. When `CredentialID` is valid but authorization checks fail, the Authorization target shall respond with `AUTH_ERROR` and `ErrorCode=AccessDenied`.

384 Table 36 — GET_CRED_ID_PARAMS request message format shows the `GET_CRED_ID_PARAMS` request message format:

385 **Table 36 — 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 16 — Authorization message request codes.
1	Reserved	1	Reserved
2	CredentialID	2	Shall be the Credential ID that identifies the requested credential.

386 Table 37 — CRED_ID_PARAMS response message format shows the successful `CRED_ID_PARAMS` response message format:

387 **Table 37 — CRED_ID_PARAMS response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `CRED_ID_PARAMS` in Table 17 — Authorization message response codes.
1	Reserved	1	Reserved
2	CredAttributes	2	The field indicates credential attributes of the requested Credential ID. The format of this field shall be as Table 38 — Credential attributes bit definitions defines.
4	CredParams	Variable	This field represents 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.

388 Table 38 — Credential attributes bit definitions defines the various Credential ID attributes:

389 **Table 38 — Credential attributes bit definitions**

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. If this bit is set, the `Lockable` bit shall also be set.
0	2	Locked	If the credentials and associated policy of the requested Credential ID are locked, this bit shall be set. This bit can be set or cleared through the `Lock` or `Unlock` operation in either `SET_CRED_ID_PARAMS` or `SET_AUTH_POLICY` request. If this bit is set, the `Lockable` bit shall also be set.
0	[7:3]	Reserved	Reserved
1	[7:0]	Reserved	Reserved

390 9.2.7.3 Credential provisioning authorization requirements

391 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.

392 9.2.8 Authorization policy provisioning and management

393 9.2.8.1 SET_AUTH_POLICY request and SET_AUTH_POLICY_DONE response messages

394 The `SET_AUTH_POLICY` request shall be used to modify a policy associated with a Credential ID as Authorization policies discusses. When `PolicyList` provides an invalid Credential ID, the Authorization target shall respond with `AUTH_ERROR` and `ErrorCode=InvalidRequest`.

395 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 ID or invalid values, the Authorization target shall respond with `AUTH_ERROR` and `ErrorCode=InvalidRequest`. When `SetAuthPolicyOp` is valid but authorization checks fail, the Authorization target shall respond with `AUTH_ERROR` and `ErrorCode=AccessDenied`.

396 Table 39 — SET_AUTH_POLICY request message format shows the `SET_AUTH_POLICY` request message format:

397 **Table 39 — 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 16 — Authorization message request codes.
1	Reserved	1	Reserved
2	SetAuthPolicyOp	1	The field indicates the requested operation. The format of this field shall be as Table 40 — Values for `SetAuthPolicyOp` field defines.
3	PolicyList	Variable	If `SetAuthPolicyOp` field indicates a PolicyChange operation, 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. If `SetAuthPolicyOp` field indicates a lock or unlock operation, the format and size of this field shall be the same format and size as the `CredentialID` field defined in Table 3 — Policy List.

398 **Table 40 — 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 ID.
2	Lock	This field shall have the same definition as the `Lock` operation as Table 34 — Values for `SetCredInfoOp` field defines.
3	Unlock	This field shall have the same definition as the `Unlock` operation as Table 34 — Values for `SetCredInfoOp` field defines.
All other values	Reserved	Reserved

399 Table 41 — SET_AUTH_POLICY_DONE response message format shows the successful `SET_AUTH_POLICY_DONE` response message format:

400 **Table 41 — 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 17 — Authorization message response codes.

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

401 **9.2.8.1.1 Additional requirements on SET_AUTH_POLICY**

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

403 **9.2.8.2 GET_AUTH_POLICY request and AUTH_POLICY response messages**

404 The `GET_AUTH_POLICY` request shall be used to retrieve the policy associated with a provisioned Credential ID. If an invalid Credential ID is requested, the Authorization target shall respond with `AUTH_ERROR` and `ErrorCode=InvalidRequest`. When `CredentialID` is valid but authorization checks fail, the Authorization target shall respond with `AUTH_ERROR` and `ErrorCode=AccessDenied`.

405 Table 42 — GET_AUTH_POLICY request message format shows the `GET_AUTH_POLICY` request message format:

406 **Table 42 — 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 16 — Authorization message request codes.
1	Reserved	1	Reserved
2	CredentialID	2	Shall be the Credential ID that identifies the requested policy.

407 Table 43 — AUTH_POLICY response message format shows the successful `AUTH_POLICY` response message format:

408 **Table 43 — AUTH_POLICY response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `AUTH_POLICY` in Table 17 — Authorization message response codes.
1	Reserved	1	Reserved
2	PolicyAttributes	2	The field indicates attributes of all policies associated with the requested Credential ID. The format of this field shall be as Table 38 — Credential attributes bit definitions defines.

Byte offset	Field	Size (bytes)	Description
4	PolicyList	Variable	This field contains 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.

409 **9.2.8.3 Authorization requirements**

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

411 9.2.9 Authorization process management

412 **9.2.9.1 General Authorization process management**

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

414 **9.2.9.1.1 GET_AUTH_PROCESSES request and AUTH_PROCESSES response messages**

415 The `GET_AUTH_PROCESSES` request and `AUTH_PROCESSES` response messages retrieve the list of active or saved Authorization 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.

416 Table 44 — GET_AUTH_PROCESSES request message format shows the `GET_AUTH_PROCESSES` request message format:

417 **Table 44 — 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 16 — Authorization message request codes.
1	Reserved	1	Reserved
2	CredentialID	2	Shall be a Credential ID. A value of 0xFFFF shall indicate all Credential IDs.

418 Table 45 — AUTH_PROCESSES response message format shows the `AUTH_PROCESSES` response message format:

419 **Table 45 — AUTH_PROCESSES response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `AUTH_PROCESSES` in Table 17 — Authorization message response codes.

Byte offset	Field	Size (bytes)	Description
1	Reserved	1	Reserved
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 processes 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 46 — Authorization process information format defines. The size of this field shall be the size of an Authorization process information multiplied by `AuthProcInfoCount`.

420 Table 46 — Authorization process information format shows the Authorization process information format:

421 **Table 46 — Authorization process information format**

Byte offset	Field	Size (bytes)	Description
0	CredentialID	2	Shall be the Credential ID associated with the Authorization process.
2	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.
3	AuthProcID	48	Shall be the Authorization Process ID associated with the `CredentialID` and `AuthProcessType`, as Authorization Process ID calculation defines.

422 **9.2.9.1.2 KILL_AUTH_PROCESS request and PROCESS_KILLED response messages**

423 The `KILL_AUTH_PROCESS` request and `PROCESS_KILLED` response messages terminate an Authorization process.

424 If the requested Authorization process to terminate is an active USAS, the USAS 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 USAS information shall no longer persist and consequently, the User shall not be able to continue the requested USAS.

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

426 An Authorization initiator shall be capable of killing only its own Authorization process, regardless of the value of `KillAuthProcPrivilege` bit.

427 Table 47 — KILL_AUTH_PROCESS request message format shows the `KILL_AUTH_PROCESS` request message format:

428 **Table 47 — 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 16 — 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	48	Shall be the Authorization Process ID associated with the `CredentialID` to terminate, as Authorization Process ID calculation defines.

429 Table 48 — PROCESS_KILLED response message format shows the `PROCESS_KILLED` response message format:

430 **Table 48 — PROCESS_KILLED response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `PROCESS_KILLED` in Table 17 — Authorization message response codes.
1	Reserved	1	Reserved
2	AuthProcID	48	Shall be the requested Authorization process ID.

431 ###### 9.2.9.1.2.1 Additional requirements for KILL_AUTH_PROCESS

432 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`.

433 ##### 9.2.9.1.3 Authorization Process ID calculation

434 The Authorization Process ID shall use the TPM_ALG_SHA_384 hash algorithm.

435 To calculate the SHA2-384 hash, the Authorization endpoint shall form `auth_proc_id_octet_string` for a given Authorization process by concatenating the following four elements in the order shown. (Note that some elements will be omitted depending on conditions.)

- 1. String prefix
  - For USAP, this element shall be omitted.
  - For SEAP, the prefix shall be one of the following:
    - If the SPDM Responder is an Authorization target, the prefix shall be "Responder".

        - If the SPDM Requester is an Authorization target, the prefix shall be "Requester".
- 2. The Authorization initiator's nonce
    - For USAP, this shall be the `[START_AUTH]` . `Nonce` .
    - For SEAP, this shall be the SPDM Requester's nonce provided in the Session-Secrets-Exchange Request.
- 3. Authorization target's nonce
    - For USAP, this shall be the `[START_AUTH_RSP]` . `Nonce` .
    - For SEAP, this shall be the SPDM Responder's nonce provided in the Session-Secrets-Exchange Response.
- 4. Saved sequence number
    - If the Authorization process is a saved USAS, this shall be the `SavedSequenceNumber` . Otherwise this element shall be omitted.

436 The `auth_proc_id_octet_string` shall be the message to hash, resulting in the Authorization Process ID.

437 9.2.9.2 USAP Management

438 9.2.9.2.1 START_AUTH request and START_AUTH_RSP response messages

439 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` of `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.

440 Table 49 — START_AUTH request message format shows the `START_AUTH` request message format:

441 **Table 49 — START_AUTH request message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `START_AUTH` in Table 16 — Authorization message request codes.
1	Reserved	1	Reserved
2	CredentialID	2	The value of this field shall be the Credential ID of the User making this request.
4	Attributes	1	Shall be the same format as Table 50 — 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` .

442 Table 50 — START_AUTH Request Attributes definition shows the field definition for `[START_AUTH]` . `Attributes` field:

443 **Table 50 — START_AUTH Request Attributes definition**

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

444 Table 51 — START_AUTH_RSP response message format shows the `START_AUTH_RSP` response message format:

445 **Table 51 — 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 17 — 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.

446 9.2.9.2.1.1 START_AUTH Additional Errors

447 If the `Continue` bit is set and an Authorization target cannot find a preserved USAS associated with the requested `CredentialID` and `Nonce`, the Authorization target shall return an `AUTH_ERROR` with `ErrorCode=InvalidRequest`.

448 9.2.9.2.2 END_AUTH request and END_AUTH_RSP response messages

449 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 a secure session to its original privilege level for that User. Additionally, the termination of a USAS does not end the corresponding secure session. If a session for the corresponding user does not exist, the Authorization target shall return an `AUTH_ERROR` with `ErrorCode=InvalidRequest`.

450 Table 52 — END_AUTH request message format shows the `END_AUTH` request message format:

451 **Table 52 — END_AUTH request message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `END_AUTH` in Table 16 — Authorization message request codes.
1	Reserved	1	Reserved
2	CredentialID	2	The value of this field shall be the Credential ID of the User making this request.
4	Attributes	1	Shall be the format as Table 54 — END_AUTH Request Attributes definition.

452 Table 53 — END_AUTH_RSP response message format shows the `END_AUTH_RSP` response message format:

453 **Table 53 — 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 17 — Authorization message response codes.
1	Reserved	1	Reserved
2	CredentialID	2	Shall be the `CredentialID` from the corresponding `END_AUTH` request.

454 Table 54 — END_AUTH Request Attributes definition shows the field definition for `[END_AUTH]` . `Attributes` field:

455 **Table 54 — 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 USAS information immediately upon the successful completion of this request. If the USAS information was previously persisted, the USAS information shall no longer be persisted. • `1`. The Authorization target shall persist or continue to persist the USAS information until the next device reset. • `2`. The Authorization target shall persist or continue to persist the USAS information across resets until credential information associated with the requested Credential ID changes. • `3`. Reserved USAS continuation defines the USAS 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 USAS and ends the USAS without issuing a successfully authorized message, then the value of this field shall remain the same persist method as before the continuation. The `KILL_AUTH_PROCESS` request can terminate all Authorization processes, regardless of the value of this field.
[7:2]	Reserved	Reserved

456 9.2.9.3 SEAP Management

457 9.2.9.3.1 ELEVATE_PRIVILEGE request and PRIVILEGE_ELEVATED response messages

458 `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 phase of the secure session. To initiate the authorization session, the Authorization initiator shall send an `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 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.

459 Table 55 — ELEVATE_PRIVILEGE request message format shows the `ELEVATE_PRIVILEGE` request message format:

460 **Table 55 — ELEVATE_PRIVILEGE request message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `ELEVATE_PRIVILEGE` in Table 16 — Authorization message request codes.
1	Reserved	1	Reserved

461 Table 56 — PRIVILEGE_ELEVATED response message format shows the `PRIVILEGE_ELEVATED` response message format:

462 **Table 56 — PRIVILEGE_ELEVATED response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `PRIVILEGE_ELEVATED` in Table 17 — Authorization message response codes.
1	Reserved	1	Reserved

463 9.2.9.3.2 END_ELEVATED_PRIVILEGE request and ELEVATED_PRIVILEGE_ENDED response message

464 The `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.

465 Table 57 — END_ELEVATED_PRIVILEGE request message format shows the `END_ELEVATED_PRIVILEGE` request message format:

466 **Table 57 — 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 16 — Authorization message request codes.
1	Reserved	1	Reserved

467 Table 58 — ELEVATED_PRIVILEGE_ENDED response message format shows the `ELEVATED_PRIVILEGE_ENDED` response message format:

468 **Table 58 — 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 17 — Authorization message response codes.

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

469 ## 9.2.10 Basic management

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

471 ### 9.2.10.1 TAKE_OWNERSHIP request and OWNERSHIP_TAKEN response

472 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.

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

474 Table 59 — TAKE_OWNERSHIP request message format shows the `TAKE_OWNERSHIP` request message format:

475 **Table 59 — TAKE_OWNERSHIP request message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `TAKE_OWNERSHIP` in Table 16 — Authorization message request codes.
1	Reserved	1	Reserved

476 Table 60 — OWNERSHIP_TAKEN response message format shows the `OWNERSHIP_TAKEN` response message format:

477 **Table 60 — OWNERSHIP_TAKEN response message format**

Byte offset	Field	Size (bytes)	Description
0	RequestResponseCode	1	Shall be the code value for `OWNERSHIP_TAKEN` in Table 17 — Authorization message response codes.
1	Reserved	1	Reserved

478 ### 9.2.10.2 AUTH_RESET_TO_DEFAULT request and AUTH_DEFAULTS_APPLIED response

479 The `AUTH_RESET_TO_DEFAULT` request and its successful `AUTH_DEFAULTS_APPLIED` response shall cause the Authorization target to reset the requested data types (such as credentials and policies) to initial or default values for unlocked credentials. This request and response shall not affect locked credentials and their associated policies and data types.

480 The Authorization target shall reset all data associated with the requested `DataType` and/or `SVResetDataType` of the SVH owner for the requested `CredentialID` to initial or default values.

481 Table 61 — AUTH_RESET_TO_DEFAULT request message format shows the `AUTH_RESET_TO_DEFAULT` request message format:

482 **Table 61 — 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 16 — Authorization message request codes.
1	Reserved	1	Reserved
2	DataType	2	This field indicates the type of data to reset to initial or default values. The format of this field shall be as Table 62 — `DataType` bit definitions defines. Zero or more bits can be set.
4	CredentialID	2	The value of this field shall indicate the unlocked Credential ID(s) of the data type(s) to reset to initial or default values. 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 reset to their initial or default values. The format of this field shall be the concatenation of Standard or Vendor Reset Data Type Element as Table 63 — 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.

483 Table 62 — `DataType` bit definitions shows the `DataType` bit definitions:

484 **Table 62 — `DataType` bit definitions**

Byte Offset	Bit Offset	Field	Description
0	0	CredIDParams	If this bit is set, Credential ID parameters shall be reset to initial or default values for the specified Credential IDs.
0	1	AuthPolicy	If this bit is set, the Authorization policy associated with the SVH in `SVResetDataTypeList` shall reset to initial or default values for the specified Credential IDs.
0	[7:2]	Reserved	Reserved

Byte Offset	Bit Offset	Field	Description
1	[7:0]	Reserved	Reserved

485 Table 63 — Standard or Vendor Reset Data Type Element format shows the definition for the standard or vendor data type to reset to initial or default values:

486 **Table 63 — 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 DSP0293 defines. The value of `LenSVH` shall be set as Common variable names 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 DSP0293 defines.
`LenSVH`	SVResetDataTypeLen	1	The value of this field shall specify the length of `SVResetDataType`, in bytes. The value of this field shall not exceed 32.
1 + `LenSVH`	SVResetDataType	`SVResetDataTypeLen`	This field shall indicate the standard or vendor specific data types to reset to initial or default values. The `SVResetDataTypeOwner` defines the format and size for this field. For this specification, the `SVResetDataType` is not present and thus the `SVResetDataTypeLen` shall have a value of zero.

487 Table 64 — AUTH_DEFAULTS_APPLIED response message format shows the `AUTH_DEFAULTS_APPLIED` response message format:

488 **Table 64 — 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 17 — Authorization message response codes.
1	Reserved	1	Reserved

489 9.2.10.2.1 AUTH_RESET_TO_DEFAULT additional requirements

490 The `DataType` field indicates either a global data type or a data type specific to this specification. For global data types, note that SVH owners or any specifications that implement this specification do not need to define an equivalent bit in their `SVResetDataType`.

491 To restore the Authorization target back to default state as Initial provisioning describes, the request shall have these values:

- `CredentialID` field shall have a value of 0xFFFF.
- All non-reserved bits shall be set in the `DataType` field of the request.
- `SVResetDataTypeCount` shall be zero.

492 Upon receiving the above request, the Authorization target shall return to the default state upon successful completion. For all other parameter combinations, the Authorization target shall remain in the Owned state and shall reset to initial or default values for the requested data types.

493 If the Authorization requires a reset to successfully complete the request and there are no other errors, the Authorization target shall reply with an `AUTH_ERROR` of `ErrorCode=ResetRequired`. Otherwise, a successful response shall indicate the Authorization target has successfully completed the requested operation.

494 `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.

495 9.3 Timing requirements

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

497 9.3.1 Message transmission time

498 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. The actual value and method of measurement of the message transmission time is outside the scope of this specification.

499 9.3.2 Authorization messages timing

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

501 If an Authorization initiator wants to retry a request, the Authorization initiator shall wait at least `AuthResponseTime` plus Message transmission time.

502 9.3.3 All messages requiring Authorization

503 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 implement this specification should consider the additional processing time needed and adjust existing timing requirements accordingly.

504 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.

505 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.

506 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.

507 10 Authorization Opaque Data Structures (AODS)

508 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 populate into which SPDM messages. This section defines the format for each AODS.

509 AODS requirements shall apply only when the secure session is an SPDM session. AODS can be used for other types of secure sessions. However, the use of AODS to fulfill the requirements in this specification while outside of an SPDM session is outside the scope of this specification.

510 10.1 General Authorization Opaque Data Structure

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

512 Table 65 — AODS general format defines the general format of all AODS.

513 **Table 65 — 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 66 — 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 the field of the same name in SPDM for definition and requirements. The `OpaqueElementData` are the fields following `DMTFspecID` (without including the `DMTFspecID` field itself).

514 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.

515 10.2 AODS error handling

516 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 use other error-handling mechanisms that are outside the scope of this specification.

517 10.3 AODS IDs

518 Table 66 — AODS IDs lists out all AODS in this specification with a short description.

519 **Table 66 — AODS IDs**

AODS ID	AODS Name	Description
0	INVOKE_SEAP	Shall invoke 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 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

520 10.4 INVOKE_SEAP AODS

521 The INVOKE_SEAP AODS shall request the other SPDM endpoint to invoke the SEAP process for the requesting SPDM endpoint. Table 67 — INVOKE_SEAP Body definition defines the format for the `AODSbody` in the AODS general format when AODS ID is `INVOKE_SEAP`.

522 **Table 67 — 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.

523 Because the INVOKE_SEAP AODS occurs before the SPDM endpoint knows the supported Authorization versions of the other SPDM endpoint, 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.

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

525 10.5 SEAP_SUCCESS AODS

526 The SEAP_SUCCESS AODS shall indicate the SEAP process during the SPDM session handshake phase for the requesting SPDM endpoint is successful. Table 68 — SEAP_SUCCESS Body definition defines the format for the `AODSbody` in the AODS general format when AODS ID is `SEAP_SUCCESS`.

527 **Table 68 — 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.

528 Because the SEAP_SUCCESS AODS occurs before the SPDM endpoint knows the supported Authorization versions of the other SPDM endpoint, 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.

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

530 10.6 AUTH_HELLO AODS

531 The AUTH_HELLO AODS shall indicate the SPDM endpoint providing this AODS is an Authorization target. Table 69 — AUTH_HELLO Body definition defines the format for the `AODSbody` in the AODS general format when AODS ID is `AUTH_HELLO`.

532 **Table 69 — 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.

533 Because the AUTH_HELLO AODS occurs before the SPDM endpoint knows the supported Authorization versions of the other SPDM endpoint, 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.

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

535 # 11 Other transport requirements

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

537 ## 11.1 Authorization record over SPDM Vendor-Defined Messages (VDM)

538 This clause defines the Authorization record over SPDM Vendor-Defined Messages (VDM) 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 an Authorization initiator and an Authorization target.

539 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.

540 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.

541 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.

542 ### 11.1.1 Additional AUTH over SPDM VDM requirements

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

544 The Authorization target should size `MaxSPDMmsgSize` in the `GET_CAPABILITIES` request and `CAPABILITIES` response messages in SPDM appropriately to receive all supported Authorization record types especially when the Authorization record carries a message requiring authorization.

545
12 Cryptographic operations

546 This section describes or defines cryptographic functions specific to Authorization.

547
12.1 Asymmetric algorithms

548 This section defines the supported asymmetric algorithms.

549 Table 70 — Base asymmetric algorithm format lists the bit definitions and other parameters associated with the respective asymmetric algorithms. BaseAsymAlgoLen is defined in Common variable names.

550 **Table 70 — 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	

551 12.2 Hash algorithms

552 This section defines the supported hash algorithms.

553 Table 71 — Base hash algorithm format lists the bit definitions of all supported base hash algorithms. BaseHashAlgoLen is defined in Common variable names.

554 **Table 71 — 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

555 12.3 Signature generation and validation

556 This section describes the `AuthSign` and `AuthSigVerify` functions.

557 12.3.1 Signature algorithm references

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

559 12.3.2 Signature generation

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

561 The signature generation function takes this form:

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

562 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

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

564 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` .

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

566 In this example, the version of this specification is 1.9.3, the User is generating a signature, and the `context` is "my example context". Thus, the `auth_prefix` is "dmtf-auth-v1.9.*dmtf-auth-v1.9.*dmtf-auth-v1.9.*dmtf-auth-v1.9.*". The `auth_context` is "user-my example context".

567 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.

568 Continuing Example 1, Table 72 — Example `combined_auth_prefix` structure 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 72 — Example `combined_auth_prefix` structure concludes Example 1.

569 **Table 72 — Example `combined_auth_prefix` structure**

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"	.	"9"	.	*
0x10	d	m	t	f	-	a	u	t	h	-	v	"1"	.	"9"	.	*
0x20	d	m	t	f	-	a	u	t	h	-	v	"1"	.	"9"	.	*
0x30	d	m	t	f	-	a	u	t	h	-	v	"1"	.	"9"	.	*
0x40	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	u	s	e

Offset	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0x50	r	-	m	y	space (0x20)	e	x	a	m	p	l	e	space (0x20)	c	o	n
0x60	t	e	x	t												

570 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.

571 Because cryptographic signing algorithms can vary widely, the following clauses define the binding of `SPDMsign` to these algorithms.

572 12.3.2.1 RSA and ECDSA signing algorithms

573 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.

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

575 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`.

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

577 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.

578 12.3.2.2 EdDSA signing algorithms

579 These algorithms are described in RFC 8032.

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

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

582 12.3.2.2.1 Ed25519 sign

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

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

585 12.3.2.2.2 Ed448 sign

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

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

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

589 12.3.2.3 SM2 signing algorithm

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

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

592 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`.

593 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.

594 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.

595 12.3.3 Signature verification

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

597 The signature verification function takes this form:

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

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

- `PubKey` : the public key associated with the given Credential ID
- `signature` : a digital signature
- `unverified_data` : a bit stream of data that needs to be verified
- `context` : a string

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

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

601 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.

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

603 Because cryptographic signature verification algorithms can vary widely, the following clauses define the binding of `AuthSigVerify` to these algorithms.

604 12.3.3.1 RSA and ECDSA signature verification algorithms

605 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.

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

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

608 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 failure.

609 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.

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

611 12.3.3.2 EdDSA signature verification algorithms

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

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

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

615 12.3.3.2.1 Ed25519 verify

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

617 `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.

618 12.3.3.2.2 Ed448 verify

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

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

621 `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.

622 **12.3.3.3 SM2 signature verification algorithm**

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

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

625 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`.

626 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.

627 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$_A$.

628 `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.

629 13 Authorization events

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

631 Authorization event requirements apply only 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.

632 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.

633 Table 73 — Authorization event types 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` .

634 **Table 73 — Authorization event types**

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

635 13.1 Event type details

636 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 Table 73 — Authorization event types.

637 13.1.1 Credential ID Parameters Changed Event

638 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.

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

640 Table 74 — Credential ID Parameters Changed Event format describes the format for `EventDetail` as SPDM defines.

641 **Table 74 — Credential ID Parameters Changed Event 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` .

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

643 13.2 Protecting the Authorization record

644 The Authorization record carries both messages requiring authorization and Authorization messages. To protect Authorization records, Authorization records should traverse a secured transport that provides, minimally, the ability to authenticate the message. The secured transport should also provide the ability to obfuscate messages.

645 If an SPDM session is used, then both the SPDM Requester and SPDM Responder shall set the `MAC_CAP` bit in their corresponding `GET_CAPABILITIES` or `CAPABILITIES` message. The SPDM endpoints can set their corresponding `ENCRYPT_CAP` bit as well.

646 13.2.1 Authorization Policy Changed Event

647 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 Table 75 — Authorization Policy Changed Event format defines. This event only indicates a single policy change. If more than one policy changes, then each change will have its own event.

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

649 Table 75 — Authorization Policy Changed Event format describes the format for `EventDetail` for the `AuthPolicyChanged` event.

650 **Table 75 — Authorization Policy Changed Event format**

Offset	Field	Size (bytes)	Description
0	CredentialID	2	Shall be the Credential ID associated with the Authorization policy that changed.

Offset	Field	Size (bytes)	Description
2	PolicyOwnerID	LenSVH	Shall identify the owner of the definition of the policy that changed. The format of this field shall be the SVH as DSP0293 defines.
2 + `LenSVH`	PolicyIdLen	2	Shall be the length of `PolicyID` field.
4 + `LenSVH`	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.

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

652 # 14 ANNEX A (informative) change log

653 ## 14.1 Version 1.0.0 (2025-10-17)

- Initial release

654 15 Bibliography

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