**Document Identifier: DSP0289**

**Date: 2024-10-10**

**Version: 1.0.0WIP80**

# Authorization Specification

**Information for Work-in-Progress version:**

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 This document is available for public review and comment until superseded.

**Provide any comments through the DMTF Feedback Portal:** https://www.dmtf.org/standards/feedback

**Supersedes: None**

**Document Class: Normative**

**Document Status: Work in Progress**

**Document Language: en-US**

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

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patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a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

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

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

CONTENTS

# 1 Foreword

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

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.

## 1.1 Acknowledgments

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

21 # 2 Introduction

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

23 ## 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.

24 ### 2.1.1 Reserved and unassigned values

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

26 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.

27 ### 2.1.2 Byte ordering

28 This section describes different byte ordering.

29 #### 2.1.2.1 Default Byte Order

30 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).

31 #### 2.1.2.2 Octet string byte order

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

33 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.

34 #### 2.1.2.3 Signature byte order

35 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.

36 ### 2.1.3 Text or string encoding

37 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.

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

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

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

| Offset | Character | Value |
|---|---|---|
| 0 | s | `0x73` |
| 1 | p | `0x70` |
| 2 | d | `0x64` |
| 3 | m | `0x6D` |

41 ### 2.1.4 Other conventions

42 Unless otherwise specified, all figures are informative.

43 # 3 Scope

44 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.

45 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.

46

# 4 Normative references

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

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

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

48

# 5 Terms and definitions

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

50 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.

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

52 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.

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

54 This specification uses these terms:

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

| Term | Definition |
| --- | --- |
| User-Specific Authorization Session | An Authorization session that is escalated specifically on behalf of a specific User. |
| Authorization message payload | Portion of the message body of an authorization message. This portion of the message is separate from those fields and elements that identify the authorization request and response codes and reserved fields. |
| Concurrent SPDM session | Simultaneous or parallel SPDM sessions between an Authorization Initiator and an Authorization Target. |

55 # 6 Symbols and abbreviated terms

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

57 The following additional abbreviations are used in this document.

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

58

# 7 Notations

59 The SPDM authorization specification uses the following notations:

| Notation | Description |
|---|---|
| `Concatenate()` | The concatenation function `Concatenate(a, b, ..., z)`, where the first entry occupies the least-significant bits and the last entry occupies the most-significant bits. |
| `M:N` | In field descriptions, this notation typically represents a range of byte offsets starting from byte `M` and continuing to and including byte `N` (`M ≤ N`).<br><br>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.<br><br>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.<br><br>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}` | Used to indicate a field in a message.<br>• `${message_name}` is the name of the request or response message.<br>• `${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. |
| `LenX` | This notation is used only in tables and indicate the length of the corresponding field only for that table. The value `X` can be a number greater than 0 especially if more than one of this notation is used in the same table for multiple fields.<br><br>This notation is not used outside of a table and `LenX` in one table has no relationship for the same `LenX` in a different table. |

# 8 SPDM authorization architecture

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

## 8.1 Architecture overview

The specification-defined message exchanges enable an entity to:

- 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 resource in an endpoint.

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.

## 8.2 Authorization version

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

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

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

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

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"`.

## 8.3 Authorization flows

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

- Credential provisioning
- Runtime authorization

### 8.3.1 Credential provisioning overview

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

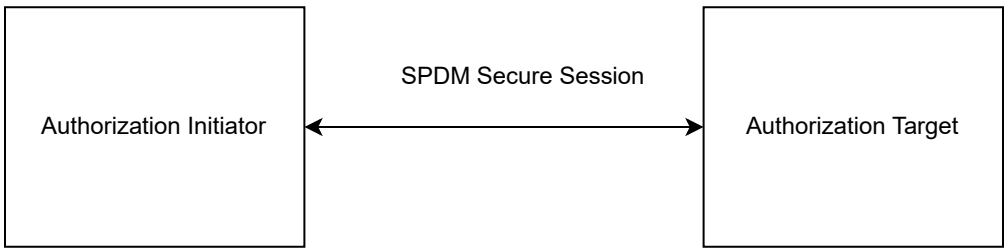
### 8.3.2 Runtime authorization overview

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

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

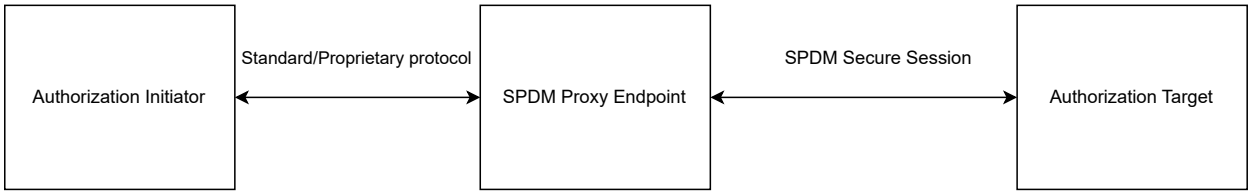
Figure 1 — Model with SPDM endpoint as Authorization Initiator shows a model where an SPDM endpoint acts as an Authorization Initiator. Figure 2 — Model with external Authorization Initiator with SPDM endpoint proxy shows a model where the Authorization Initiator is an entity that is not an SPDM endpoint,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

**Figure 1 — Model with SPDM endpoint as Authorization Initiator**



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

83




84 ## 8.4 Credentials

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

86 A credential is associated with an identifier, type, cryptographic algorithms, credential data and a *credential slot*. A credential slot is a logical location that holds a credential structure. An endpoint which supports provisioning credentials shall support a minimum of 8 credential slots. Credential slots are identified by the `CredentialID`. The mandatory 8 credential slots shall have `CredentialID` of 0 through 7 inclusive. The `SET_CREDENTIAL_INFO` command shall be used to provision credentials into a credential slot and shall use the structure defined in credential structure. Provisioning of slot 0 should only be done in a trusted environment (such as a secure manufacturing environment) as Initial Provisioning describes. All other slots may be provisioned in a trusted environment or use a credential already provisioned in the endpoint to authorize the `SET_CREDENTIAL_INFO` command in an untrusted environment. Credentials should be stored by the endpoint in integrity protected storage. An endpoint may use the credential structure as defined in the specification or use an implementation-specific data structure to store credentials.

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

88 **Table 2 — Credential Structure**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | StructVersion | 1 | Bits [7:4] - Major Version<br>Bits [3:0] - Minor Version<br>For this version of the specification, the value shall be `0x10` |
| 1 | CredentialType | 1 | The type of the credential.<br>• `0x1` - Asymmetric Key.<br>Shall be `0x1` for this version of the specification. All other values are reserved. |

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 2 | CredentialID | 2 | A unique identifier to identify the credential and the credential slot. The value of 0xFFFF shall be reserved unless other parts of this specification defines the use for this value. |
| 4 | BaseAlgo | 4 | When `CredentialType` is `0x1`:<br>• Let `SigLen` be the size of the signature in bytes. Only one of the following bits shall be set.<br>• Byte 0 Bit 0. TPM_ALG_RSASSA_2048 where `SigLen` =256.<br>• Byte 0 Bit 1. TPM_ALG_RSAPSS_2048 where `SigLen` =256.<br>• Byte 0 Bit 2. TPM_ALG_RSASSA_3072 where `SigLen` =384.<br>• Byte 0 Bit 3. TPM_ALG_RSAPSS_3072 where `SigLen` =384.<br>• Byte 0 Bit 4. TPM_ALG_ECDSA_ECC_NIST_P256 where `SigLen` =64 (32-byte r followed by 32-byte s).<br>• Byte 0 Bit 5. TPM_ALG_RSASSA_4096 where `SigLen` =512.<br>• Byte 0 Bit 6. TPM_ALG_RSAPSS_4096 where `SigLen` =512.<br>• Byte 0 Bit 7. TPM_ALG_ECDSA_ECC_NIST_P384 where `SigLen` =96 (48-byte r followed by 48-byte s).<br>• Byte 1 Bit 0. TPM_ALG_ECDSA_ECC_NIST_P521 where `SigLen` =132 (66-byte r followed by 66-byte s).<br>• Byte 1 Bit 1. TPM_ALG_SM2_ECC_SM2_P256 where `SigLen` =64 (32-byte `SM2_R` followed by 32-byte `SM2_S` ).<br>• Byte 1 Bit 2. EdDSA ed25519 where `SigLen` =64 (32-byte R followed by 32-byte S).<br>• Byte 1 Bit 3. EdDSA ed448 where `SigLen` =114 (57-byte R followed by 57-byte S).<br>• All other values reserved. |

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 8 | BaseHashAlgo | 4 | When `CredentialType` is `0x1`, only one of the following bits shall be set:<br>• Byte 0 Bit 0. TPM_ALG_SHA_256<br>• Byte 0 Bit 1. TPM_ALG_SHA_384<br>• Byte 0 Bit 2. TPM_ALG_SHA_512<br>• Byte 0 Bit 3. TPM_ALG_SHA3_256<br>• Byte 0 Bit 4. TPM_ALG_SHA3_384<br>• Byte 0 Bit 5. TPM_ALG_SHA3_512<br>• Byte 0 Bit 6. TPM_ALG_SM3_256<br>• All other values reserved. |
| 12 | Reserved | 4 | Reserved. |
| 16 | CredentialDataSize | 2 | Size of Credential data in bytes. |
| 18 | CredentialData | `CredentialDataSize` | When `CredentialType` is `0x1`:<br>• This field shall contain the public key in the `SubjectPublicKeyInfo` format specified by RFC 7250. |

89

## 8.5 Credential policies

90 All credentials shall be associated with an authorization policy. A credential shall not be usable for authorization without an associated policy. A policy shall be associated with a credential using the `SET_CREDENTIAL_POLICY` command. For the credential in slot 0, the command should only be sent in a trusted environment (such as a secure manufacturing environment) and shall not be modified outside a trusted environment without appropriate authorization. For the credential in slot 0, at least one policy shall be provisioned in a trusted environment. For credentials in other slots, `SET_CREDENTIAL_POLICY` may be used in a trusted environment (such as a secure manufacturing environment) or use a credential already provisioned in the endpoint to authorize the command. Effectively, credential policies for slots other than slot 0 may be protected resources themselves. Policies should be stored by the endpoint in integrity protected storage. An endpoint may use the Policy List structure as defined in the specification or use an implementation-specific data structure to store credential policies.

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

92 **Table 3 — Policy List**

| Byte Offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | CredentialID | 2 | A unique identifier to identify the credential and the credential slot. |
| 2 | NumPolicies | 2 | Shall be the number of policies listed in the `Policies` field. The value of this field shall be at least one. |

| Byte Offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 4 | Policies | Variable | List of policies as defined by Table 4 — Policy Structure. |

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

94 **Table 4 — Policy Structure**

| Byte Offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | PolicyOwnerID | Len0 | This field shall indicate the owner of the policy. The size and format of this field shall be the same as the `SVH` as SPDM defines. |
| `Len0` | PolicyLen | 2 | Shall be the length of `Policy`. |
| 2 + `Len0` | Policy | `PolicyLen` | This field indicates the policy as `PolicyOwnerID` defines. The `PolicyOwnerID` shall define the size and format of this field.<br><br>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. |

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

96 **Table 5 — DSP0289 Policy Structure**

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

97 ### 8.5.1 DSP0289 Credential Policy

98 This section defines the privileges for commands, actions and other resources that this specification defines. Each

credential ID has an associated policy. An Authorization initiator uses `SET_CREDENTIAL_POLICY` command to change the policy associated with the Credential ID provided in the request.

99 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.

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

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

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

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

103 **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. |

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

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

| Byte Offset | Field | Size (bytes) | Field Type | Description |
|---|---|---|---|---|
| 0 | AllowedBaseAlgo | 4 | Allowable | The format of this field shall be the same as `BaseAlgo` in Table 2 — Credential Structure. This field reflects the allowed base algorithms the given Credential ID can use.<br><br>If a bit is set, the given Credential ID shall be capable of utilizing the corresponding algorithm. If a bit is not set, the given Credential ID shall be prohibited from utilizing the corresponding algorithm.<br><br>At least one bit shall be set. |

| Byte Offset | Field | Size (bytes) | Field Type | Description |
|---|---|---|---|---|
| 4 | AllowedBaseHashAlgo | 4 | Allowable | The format of this field shall be the same as `BaseHashAlgo` in Table 2 — Credential Structure. This field reflects the allowed base hash algorithms the given Credential ID can use.<br><br>If a bit is set, the given Credential ID shall be capable of utilizing the corresponding hash. If a bit is not set, the given Credential ID shall be prohibited from utilizing the corresponding hash.<br><br>At least one bit shall be set. |
| 8 | CredentialPrivileges | 2 | | The format of this field shall be the format as Table 8 — DSP0289 Credential Policy Bits Definitions defines. |
| 10 | AuthProcessPrivileges | 1 | | The format of this field shall be the format as Table 9 — DSP0289 Authorization Policy Bits Definition defines.<br><br>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. |

106 Table 8 — DSP0289 Credential Policy Bits Definition defines the credentials provisioning policies.

107 **Table 8 — DSP0289 Credential Policy Bits Definition**

| Byte Offset | Bit Offset | Field | Field Type | Description |
|---|---|---|---|---|
| 0 | 0 | CredentialInfoPrivilege | Privilege | If this bit is set, the given Credential ID shall be capable of modifying or retrieving credential information of other Credential IDs through `SET_CREDENTIAL_INFO` and `GET_CREDENTIAL_INFO` requests. |
| 0 | 1 | GrantOtherPolicyPrivilege | Privilege | If this bit is set, the given Credential ID shall be capable of only granting privilege for all fields of Privilege field type for other Credential IDs through the `SET_CREDENTIAL_POLICY` request.<br><br>Also, setting this bit also allows the given Credential ID to modify all fields of Allowable field type in any manner.<br><br>If this bit is set, the `QueryPolicyPrivilege` shall also be set. |
| 0 | 2 | RevokeOtherPolicyPrivilege | Privilege | If this bit is set, the given Credential ID shall be capable of only revoking privileges for all fields of Privilege field type for other Credential IDs through the `SET_CREDENTIAL_POLICY` request.<br><br>If this bit is set, the `QueryPolicyPrivilege` shall also be set. |
| 0 | 3 | QueryPolicyPrivilege | Privilege | If this bit is set, the given Credential ID shall be capable of retrieving credential policy of other Credential IDs through `GET_CREDENTIAL_POLICY` request. |

| Byte Offset | Bit Offset | Field | Field Type | Description |
|---|---|---|---|---|
| 0 | 4 | ResetToDefaultsPrivilege | Privilege | If this bit is set, the given Credential ID shall be capable of resetting the Authorization target back to default values and behavior using the `AUTH_RESET_TO_DEFAULT` request. |
| 0 | [7:5] | Reserved | Reserved | Reserved. |
| 1 | [7:0] | Reserved | Reserved | Reserved. |

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

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

| Byte Offset | Bit Offset | Field | Field Type | Description |
|---|---|---|---|---|
| 0 | 0 | PrivilegeSEAP | Privilege | If this bit is set, the given Credential ID shall be capable of invoking the SEAP process as an Authorization initiator. |
| 0 | 1 | PrivilegeUSAP | Privilege | If this bit is set, the given Credential ID shall be capable of being a user in the USAP Process. |
| 0 | [7:2] | Reserved | Reserved | Reserved. |

110 #### 8.5.1.1 DSP0289 Additional Credential Policy Requirements

111 When changing the credential 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 `CredentialInfoPrivilege` field, the Authorization target can apply the new settings to incoming messages without ending an active Authorization process. In another example, if a bit is cleared in `AllowedBaseAlgo` and an active Authorization process is using the corresponding asymmetric algorithm, then the Authorization target will have to fail authorization for all messages requiring authorization until the Authorization initiator changes the credential ID parameters back to compliance with the new policy.

112 For some policy changes, there are some specific requirements. If a new policy clears a bit in an Allowable field type and the current credential ID parameters associated with that Credential ID uses the corresponding bit, the Authorization target shall still allow the Authorization initiator to use the existing credential ID parameters to change the parameters to comply with the new policy through `SET_CREDENTIAL_INFO` and `GET_CREDENTIAL_INFO` while failing authorization for all other messages requiring authorization.

113 An Authorization initiator should initially configure the credential policy for a given Credential ID using the `SET_CREDENTIAL_POLICY` request before initially setting the credential information via `SET_CREDENTIAL_INFO` request for the same Credential ID.

## 8.6 Initial Provisioning

Initial provisioning or provisioning from the default state is important to ensure proper operation of the Authorization target. Thus, an Authorization target shall allow an Authorization initiator to set both the credential information and credential policy of Credential ID 0 as the default. Also, in the default state, an Authorization target shall allow an Authorization initiator to modify credentials and associated policy for Credential 0 in any manner without credentials. The Authorization target shall prohibit the provisioning of information or policies associated with other Credential IDs and shall fail authorization of all messages (including messages of other protocols) requiring authorization until initial provisioning completes. The successful completion of `SET_CREDENTIAL_INFO` for Credential ID 0 shall mark the completion of initial provisioning and an exit out of the default state.

After initial provisioning, the Authorization target shall then enforce authorization for all messages requiring authorization.

The above method gives Credential ID 0 the highest privileges in the Authorization target by default. Depending on the final policy settings of the Authorization target after initial provisioning, the Authorization initiator can configure one or more Credential IDs to have the highest privilege levels or disperse privileges across two or more Credential IDs. Furthermore, the authorization initiator can configure privileges in such a way that significantly disables operation of the Authorization target and recovery from such a state is outside the scope of this specification.

## 8.7 Discovery

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

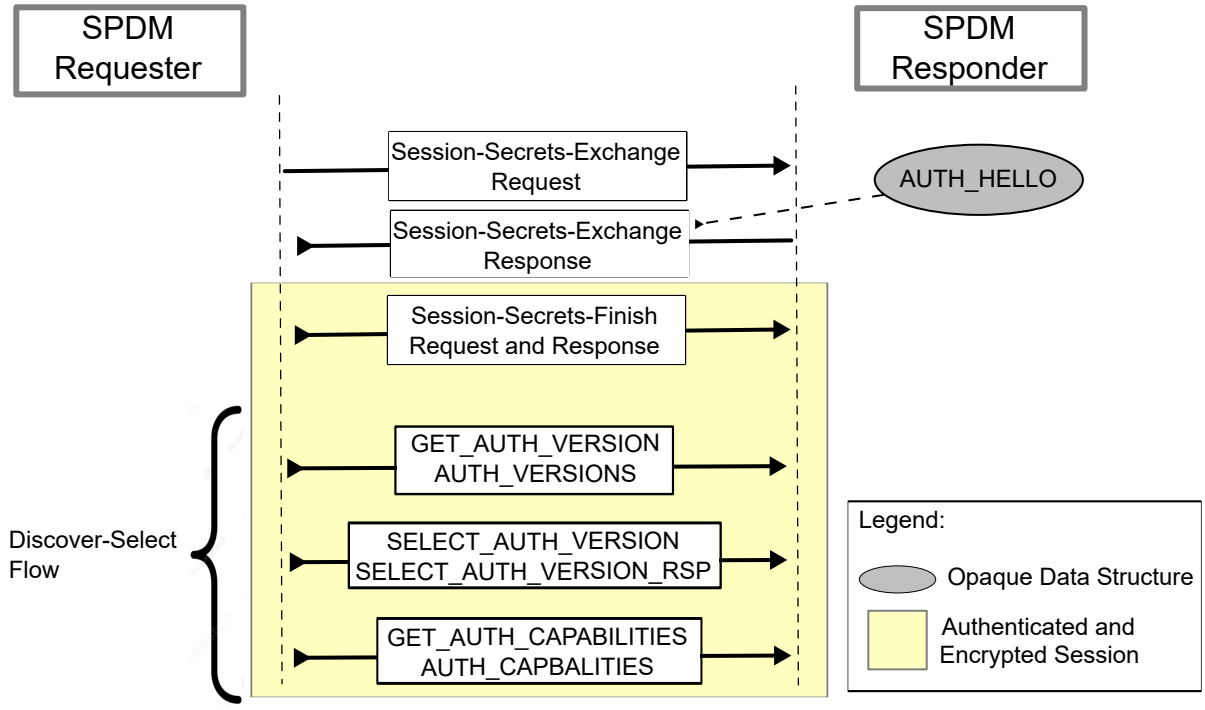
In the announcement phase, an Authorization target announces itself at the start of a session. If an SPDM requester is an Authorization target, the SPDM requester shall populate the `AUTH_HELLO` AODS in the Session-Secrets-Exchange request. Likewise, if an SPDM responder is an Authorization target, the SPDM responder shall populate either the `AUTH_HELLO` AODS or the `SEAP_INVOKED` AODS in the Session-Secrets-Exchange response. If an SPDM responder can populate the `SEAP_INVOKED` AODS according to the SEAP process, an SPDM responder shall use the `SEAP_INVOKED` AODS instead of the `AUTH_HELLO` AODS.

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

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

**Figure 3 — Common Discovery Phase**

124




125

## 8.8 Authorization Process

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

127 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 Credential policy provisioning and management for details .

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

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

- User-Specific Authorization Process
- SPDM Endpoint Authorization Process

130

### 8.8.1 User-Specific Authorization Process

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

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

133 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 SPDM secure session for that specific User. This portion of an SPDM session is called an Authorization session.
- It initializes critical cryptographic parameters for all messages requiring authorization in the corresponding SPDM session and corresponding User. Messages that traverse the corresponding SPDM session can be messages of any protocol and not restricted to SPDM or Authorization messages.
- The message format of all messages requiring authorization changes to accommodate authorization data for the corresponding User. The format for such messages is defined in the Authorization Record Section.

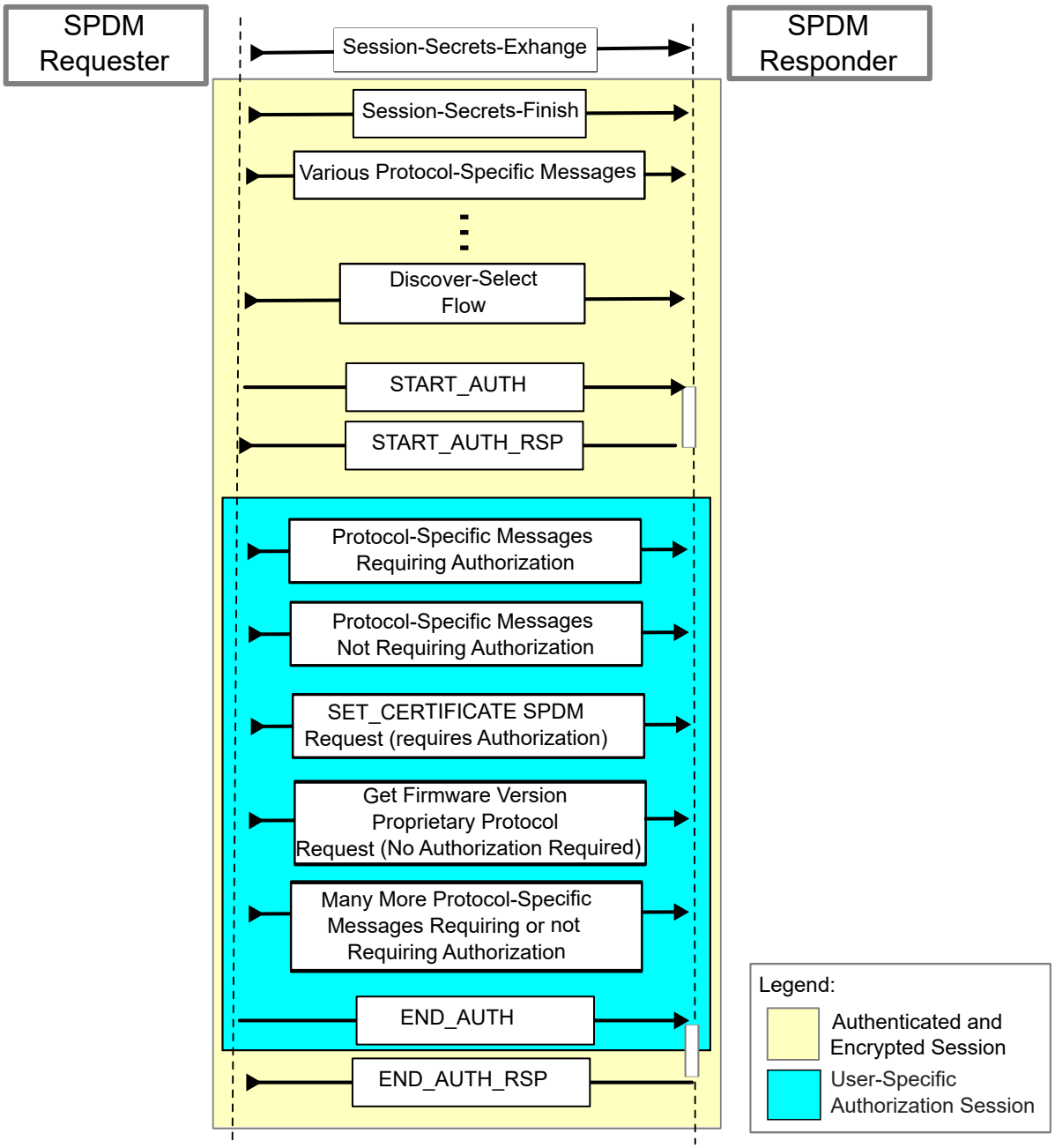
134 The successful completion of this request and response effectively establishes the Authorization session for the corresponding User. While the authorization session is active, messages requiring authorization shall contain authorization data, called Authorization tag, for the corresponding User.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

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

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

137 **Figure 4 — Authorization Process**

138




139 **8.8.1.1 USAP error handling, requirement and notes**

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

target shall respond with an `AUTH_ERROR` or silent discard the request if a `START_AUTH` is received for a User with a corresponding active User-Specific Authorization Session.

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

142 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.

### 143 8.8.2 SPDM Endpoint Authorization Process

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

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

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

147 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. If the SPDM responder can support SEAP for this new session, it shall return `SEAP_INVOKED` data structure to the `OpaqueData` field in the Session-Secrets-Exchange response; otherwise, this data structure shall be absent. Additionally, if the SPDM responder wants to send messages requiring authorization to the SPDM requester using SEAP in the same session, the SPDM responder shall also add the `INVOKE_SEAP` data structure to the `OpaqueData` field of the Session-Secret-Exchange response. If the SPDM responder provides the `INVOKE_SEAP` data structure and the SPDM requester supports authorizing the SPDM responder using SEAP, the SPDM requester shall add the `SEAP_INVOKED` data structure to the Session-Secrets-Finish request. Lastly, the SPDM endpoints shall populate all fields appropriately in a Session-Secrets-Exchange request and response message to perform mutual authentication.

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

149 Before the second part of SEAP can begin, the Authorization Initiator should send `GET_AUTH_VERSION` to query for the supported version followed by the `SELECT_AUTH_VERSION` to select the version to be used for subsequent messages and then followed the `GET_AUTH_CAPABILITIES` to obtain the supported capabilities of the Authorization target. The Authorization Initiator can skip these messages if it already knows the information beforehand such as from a prior or

concurrent SPDM session or from an earlier request in the same session. The Authorization Initiator should send `GET_AUTH_VERSION` and `GET_AUTH_CAPABILITIES` before the second part of SEAP in each SPDM session to ensure the Authorization Initiator has the most up to date information. Additionally, the SPDM requester and the SPDM responder may not support the same versions or capabilities even though they can be both Authorization Initiators in the same session.

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

151 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.

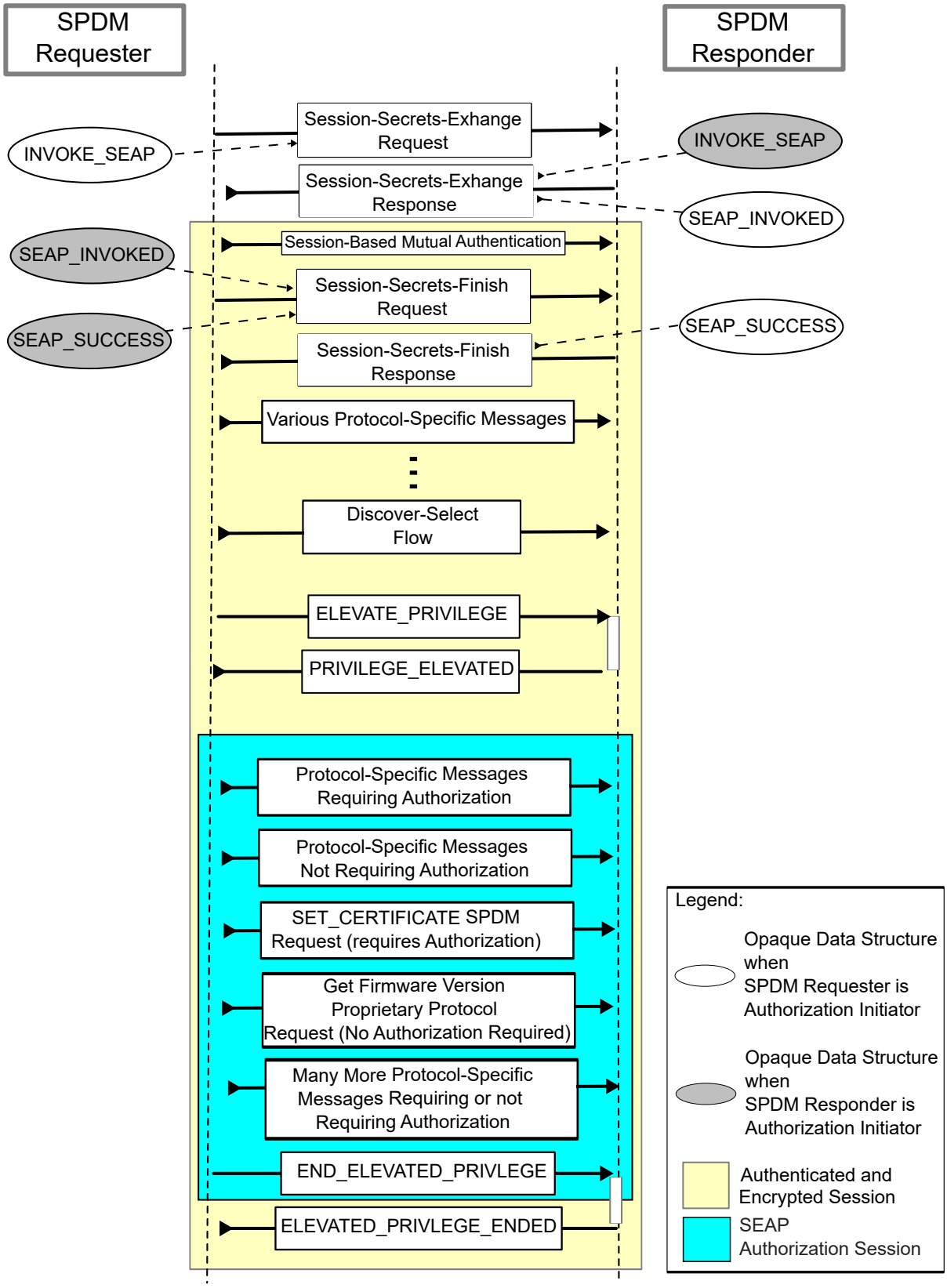
152 The successful completion of this request and response effectively establishes the Authorization session for the corresponding Authorization Initiator. In an Authorization session, when the Authorization target receives a message from any protocol in the corresponding SPDM session, the Authorization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

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

154 Figure 5 — SPDM Endpoint Authorization Process (SEAP) illustrates the SPDM Endpoint Authorization Process (SEAP).

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

156


SPDM Requester
SPDM Responder
INVOKE_SEAP
Session-Secrets-Exhange Request
Session-Secrets-Exhange Response
INVOKE_SEAP
SEAP_INVOKED
SEAP_INVOKED
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_PRIVLEGE
ELEVATED_PRIVLEGE_ENDED
Legend:
Opaque Data Structure when SPDM Requester is Authorization Initiator
Opaque Data Structure when SPDM Responder is Authorization Initiator
Authenticated and Encrypted Session
SEAP Authorization Session

157 #### 8.8.2.1 SEAP error handling, requirement and notes

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

159 If SEAP uses SPDM Pre-shared key exchange flow, then `SEAP_SUCCESS` cannot be supported because there is no `OpaqueData` Field in the `PSK_FINISH` or `PSK_FINISH_RSP`. Thus, if the first part of SEAP fails, the Authorization target shall return an `AUTH_ERROR` in the `ELEVATE_PRIVILEGE` request.

160 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 the `SEAP_INVOKED` data structure is present in Session-Secret-Exchange response, then the corresponding SPDM session shall prohibit the use of USAP.

161 ### 8.8.3 Other error handling, requirements and notes

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

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

163 The Authorization sessions does not limit the types of messages that can traverse an SPDM session but rather enables explicit validation of authority for all messages according to provisioned credentials and policies. Furthermore, this specification strongly recommends that messages requiring authorization be denied access for Users or Authorization entities outside of an Authorization session.

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

165 ### 8.8.4 Authorization Tag

166 The authorization tag is data that accompanies every message that requires authorization in a User-specific authorization session. The 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.

167 This specification supports asymmetric signature algorithms.

#### 168 8.8.4.1 Authorization Tag Signature Generation and Verification

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

170 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.

171 To compute the signature, first, 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

172 The sequence number shall start at 1 with the successful completion of `START_AUTH` request and shall increment by one after each message requiring authorization corresponding to the User. For the Authorization target, the sequence number shall increment by one after receiving a message containing an Authorization tag from the corresponding User.

173 The actual contents of the message body shall be the bytes of length `PayloadLen` in the `Payload` 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.

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

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

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

176 where:

- The `UserPrivKey` shall be the private key associated with the corresponding User.

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

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

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

179 where:

- The `UserPublicKey` shall be the public key associated with the User associated with the corresponding credential ID.
- The `AuthSignature` shall be the signature in the Authorization tag which accompanied the message.

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

181 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.

182 Otherwise, the message fails authorization.

# 9 SPDM authorization messages

## 9.1 Authorization messages overview

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

### 9.1.1 Bi-directional Authorization message processing

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

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

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

The following clause assumes that an SPDM endpoint is the Authorization Initiator.

### 9.1.2 Requirements for Authorization Initiators

An Authorization Initiator shall not have multiple outstanding requests to the same authorization target, within a single SPDM session.

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

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

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

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

### 9.1.3 Requirements for Authorization Targets

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

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

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

### 9.1.4 Authorization Messages bits-to-bytes mapping

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

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

**Figure 6 — One-byte field bit map**

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

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

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

**Figure 7 — Two-byte field bit map**

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

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

### 9.1.5 Version encoding

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

| Version | Matches | Incremented when |
|---|---|---|
| Major | Major version field in the `AuthVersion` field in the `SELECT_AUTH_VERSION` message. | Protocol modification breaks backward compatibility. |
| Minor | Minor version field in the `AuthVersion` field in the `SELECT_AUTH_VERSION` message. | Protocol modification maintains backward compatibility. |

210 EXAMPLE:

211 Version 3.7 → `0x37`

212 Version 1.0 → `0x10`

213 Version 1.2 → `0x12`

214 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.

215 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`.

216 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 change 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 for the `GET_AUTH_VERSION` request and `AUTH_VERSION` response messages) clarify existing behaviors in the authorization specification. They maintain bitwise compatibility with the base version, except as required to fix security vulnerabilities or to correct mistakes from the base version.

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

### 9.1.6 Authorization Record

An authorization record is a wrapper structure that shall be used to carry all message defined in this specification, and any protocol-specific messages that need to be authorized using the authorization tag, when authorization uses USAP. The authorization record should be used when authorization uses SEAP since it does not use an authorization tag and the `CredentialID` is implicit in the SPDM session used. The authorization record provides a transport and protocol agnostic way to send and receive authorized messages compliant with this specification. Each transport shall define a specific mechanism to indicate that a given payload is an authorization record, so that it can be identified and forwarded to the authorization logic for further processing. An authorization target shall use the `CredentialID` field to locate the credential to verify the authorization tag, if present in the authorization record.

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

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | Flags | 1 | Bits [7:1] - Reserved<br>Bit[0] - If bit is set, `AuthTag` is present, if not, `AuthTag` is not present |
| 1 | CredentialID | 2 | A unique identifier to identify the credential and the credential slot. |
| 3 | AuthTagLen | 2 | Length, in bytes, of `AuthTag`. Shall be 0 when Bit 0 of `Flags` is not set. |
| 5 | AuthTag | `AuthTagLen` | Authorization Tag for the `Payload`, as defined in Authorization Tag. |
| 5 + AuthTagLen | PayloadLen | 4 | Length, in bytes, of `Payload`. |
| 9 + AuthTagLen | Payload | `PayloadLen` | The payload for the authorized message. |

### 9.1.7 Generic Authorization message format

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

**Table 11 — Generic authorization message field definitions**

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

224 ## 9.2 Authorization messages

225 This section discusses all authorization request and response messages.

226 ### 9.2.1 Authorization message request codes

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

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

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

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

| Request | Code value | Implementation requirement | Authorization Requirements | Message format |
|---|---|---|---|---|
| GET_AUTH_VERSION | 0x81 | Required | None | Table 16 — GET_AUTH_VERSION request message format |

| Request | Code value | Implementation requirement | Authorization Requirements | Message format |
|---|---|---|---|---|
| SELECT_AUTH_VERSION | 0x82 | Required | None | Table 19 — SELECT_AUTH_VERSION request message format |
| SET_CREDENTIAL_INFO | 0x86 | Optional | Conditional | Table 25 — SET_CREDENTIAL_INFO request message format |
| GET_CREDENTIAL_INFO | 0x87 | Required | Conditional | Table 28 — GET_CREDENTIAL_INFO request message format |
| SET_CREDENTIAL_POLICY | 0x88 | Optional | Conditional | Table 30 — SET_CREDENTIAL_POLICY request message format |
| GET_CREDENTIAL_POLICY | 0x89 | Required | Conditional | Table 33 — GET_CREDENTIAL_POLICY request message format |
| START_AUTH | 0x83 | Optional | None | Table 35 — START_AUTH request message format |
| END_AUTH | 0x84 | Optional | None | Table 37 — END_AUTH request message format |
| ELEVATE_PRIVILEGE | 0x85 | Optional | None | Table 39 — ELEVATE_PRIVILEGE request message format |
| END_ELEVATED_PRIVILEGE | 0x86 | Optional | None | Table 41 — END_ELEVATED_PRIVILEGE request message format |
| GET_AUTH_CAPABILITIES | 0x8B | Required | None | Table 21 — GET_AUTH_CAPABILITIES request message format |
| AUTH_RESET_TO_DEFAULT | 0x8C | Optional | Mandatory | Table 43 — AUTH_RESET_TO_DEFAULT request message format |
| Reserved | All other values | Reserved | Reserved | Authorization implementations compatible with this version shall not use the reserved request codes. |

### 9.2.2 Authorization message response codes

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

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.

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

**Table 13 — Authorization message response codes**

| Response | Value | Implementation requirement | Message format |
|---|---|---|---|
| AUTH_VERSION | 0x01 | Required | Table 17 — Successful AUTH_VERSION response message format |
| SELECT_AUTH_VERSION_RSP | 0x02 | Required | Table 20 — Successful SELECT_AUTH_VERSION_RSP response message format |
| SET_CREDENTIAL_INFO_RSP | 0x06 | Optional | Table 27 — Successful SET_CREDENTIAL_INFO_RSP response message format |
| GET_CREDENTIAL_INFO_RSP | 0x07 | Required | Table 29 — Successful GET_CREDENTIAL_INFO_RSP response message format |
| SET_CREDENTIAL_POLICY_RSP | 0x08 | Optional | Table 32 — Successful SET_CREDENTIAL_POLICY_RSP response message format |
| GET_CREDENTIAL_POLICY_RSP | 0x09 | Required | Table 34 — Successful GET_CREDENTIAL_POLICY_RSP response message format |
| START_AUTH_RSP | 0x03 | Optional | Table 36 — Successful START_AUTH_RSP response message format |
| END_AUTH_RSP | 0x04 | Optional | Table 38 — Successful END_AUTH_RSP response message format |
| ELEVATE_PRIVILEGE_RSP | 0x05 | Optional | Table 40 — Successful ELEVATE_PRIVILEGE_RSP response message format |

| Response | Value | Implementation requirement | Message format |
|---|---|---|---|
| END_ELEVATED_PRIVILEGE_RSP | 0x6 | Optional | Table 42 — Successful END_ELEVATED_PRIVILEGE_RSP response message format |
| AUTH_CAPABILITIES | 0xB | Required | Table 22 — Successful AUTH_CAPABILITIES response message format |
| AUTH_DEFAULTS_APPLIED | 0xC | Optional | Table 46 — AUTH_DEFAULTS_APPLIED response message format |
| AUTH_ERROR | 0x7F | Required | Table 14 — AUTH_ERROR response message format |
| Reserved | All other values | Reserved | Authorization implementations compatible with this version shall not use the reserved request codes. |

## 9.2.3 Error handling

This section discusses general error handling for all authorization messages.

### 9.2.3.1 AUTH_ERROR response message

For an authorization request message that results in an error, the authorization target should send an `AUTH_ERROR` message to the Requester.

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

Table 15 — Error code and error data shows the detailed error code, error data, and extended error data.

**Table 14 — AUTH_ERROR response message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x7F` = `AUTH_ERROR`. See Table 13 — Authorization message response codes. |
| 1 | Reserved | 1 | Reserved. |
| 2 | ErrorCode | 1 | Shall be the ErrorCode. See Table 15 — Error code and error data. |
| 3 | ErrorData | 0-32 | Shall be the Error data. See Table 15 — Error code and error data. |

**Table 15 — Error code and error data**

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

### 9.2.4 Discovery message

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

246 #### 9.2.4.1 GET_AUTH_VERSION request and AUTH_VERSION response messages

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

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

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

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

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

252 **Table 16 — GET_AUTH_VERSION request message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x81` = `GET_AUTH_VERSION`. See Table 12 — Authorization Message request codes. |
| 1 | Reserved | 1 | Reserved. |

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

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

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x01` = `AUTH_VERSION`. See Table 13 — Authorization Message response codes. |
| 1 | Reserved | 1 | Reserved. |

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 2 | VersionNumberEntryCount | 1 | Number of version entries present in this table (=n). |
| 3 | VersionNumberEntry1:n | 2 * n | 16-bit version entry. See Table 18 — VersionNumberEntry definition. Each entry should be unique. |

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

256 **Table 18 — VersionNumberEntry definition**

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

257 #### 9.2.4.2 SELECT_AUTH_VERSION request and SELECT_AUTH_VERSION_RSP response messages

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

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

**Table 19 — SELECT_AUTH_VERSION request message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x82` = `SELECT_AUTH_VERSION`. See Table 12 — Authorization Message request codes. |
| 1 | Reserved | 1 | Reserved. |
| 2 | AuthVersion | 1 | The version that shall be used for all subsequent communication between the Authorization Initiator and target. |

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

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

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x02` = `SELECT_AUTH_VERSION_RSP`. See Table 13 — Authorization Message response codes. |
| 1 | Reserved | 1 | Reserved. |

### 9.2.4.3 GET_AUTH_CAPABILITIES request and AUTH_CAPABILITIES response messages

The `GET_AUTH_CAPABILITIES` request and `AUTH_CAPABILITIES` response shall retrieve capability information from the Authorization target. The request and response parameters for this message are listed in Table 19 and Table 20. While this request can be sent multiple times at any time, the request should be sent as the Discovery section describes.

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

**Table 21 — GET_AUTH_CAPABILITIES request message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x8B` = `GET_AUTH_CAPABILITIES`. See Table 12 — Authorization Message request codes. |
| 1 | Reserved | 1 | Reserved. |

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

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

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x0B` = `AUTH_CAPABILITIES`. See Table 13 — Authorization Message response codes. |
| 1 | Reserved | 1 | Reserved. |
| 2 | MessageCaps | 2 | The format of this field shall be the format as Table 23 — Message Supported Bit Definitions defines. |
| 4 | AuthProcessCaps | 2 | The format of this field shall be the format as Table 24 — Authorization Process Supported Bit Definitions defines. |
| 6 | BaseAlgoSupported | Len0 | If a bit is set, the Authorization target supports the corresponding asymmetric algorithm. Otherwise, the bit shall be clear.<br><br>The format and size of this field shall be the format and size as the `BaseAlgo` field in Table 2 — Credential structure defines. |
| 6 + `Len0` | BaseHashAlgoSupported | Len1 | If a bit is set, the Authorization target supports the corresponding hash algorithm. Otherwise, the bit shall be clear.<br><br>The format and size of this field shall be the format and size as the `BaseHashAlgo` field in Table 2 — Credential structure defines. |
| 6 + `Len0` + `Len1` | SupportedPolicyCount | 2 | The value of this field shall be the number of policy owners in `SupportedPolicyList`. If the value of this field is zero, then the `SupportedPolicyList` field shall be absent. |
| 8 + `Len0` + `Len1` | SupportedPolicyList | Variable | This field summarizes the policies the Authorization target supports by only listing the policy owners ( `PolicyOwnerID` ).<br><br>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 `SupportedPolicyCount` field. If more than one policy has the same `PolicyOwnerID`, then this list shall only contain one instance of this `PolicyOwnerID`. Finally, this list shall be considered to be unordered.<br><br>To retrieve more details of policy support, the Authorization initiator can use the `GET_CREDENTIAL_POLICY` and the corresponding response. |

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

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

| Byte Offset | Bit Offset | Field | Description |
|---|---|---|---|
| 0 | 0 | ChangeCredIDParamsCap | If the Authorization target supports `SET_CREDENTIAL_INFO_RSP`, then this bit shall be set. Otherwise, this bit shall not be set. |

| Byte Offset | Bit Offset | Field | Description |
|---|---|---|---|
| 0 | 1 | ChangeCredPolicyCap | If the Authorization target supports `SET_CREDENTIAL_POLICY_RSP`, then this bit shall be set. Otherwise, this bit shall not be set. |
| 0 | [7:2] | Reserved | Reserved. |
| 1 | [7:0] | Reserved | Reserved |

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

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

| Byte Offset | Bit Offset | Field | Description |
|---|---|---|---|
| 0 | 0 | USAPcap | If the Authorization target supports USAP, then this bit shall be set. Otherwise, this bit shall not be set.<br><br>If this bit is set, `START_AUTH_RSP`, `END_AUTH_RSP` response message shall be supported. |
| 0 | 1 | SEAPcap | If the Authorization target supports SEAP, then this bit shall be set. Otherwise, this bit shall not be set.<br><br>If this bit is set, `PRIVILEGE_ELEVATED` and `ELEVATED_PRIVILEGE_ENDED` response messages shall be supported. |
| 0 | [7:2] | Reserved | Reserved. |
| 1 | [7:0] | Reserved | Reserved |

273 ## 9.2.5 Credential provisioning

274 ### 9.2.5.1 SET_CREDENTIAL_INFO request and SET_CREDENTIAL_INFO_RSP response messages

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

276 The Authorization Initiator shall use the `Flags` parameter to specify if the request is to provision the list of credentials or to erase the list of existing credentials. An authorization target shall ensure that the operation is atomic, that is, all credentials in the `CredentialList` can successfully be provisioned or erased, and fail if that is not possible. All erase operations, including on credential slot 0, shall be authorized by an existing credential.

277 Table 25 — SET_CREDENTIAL_INFO request message format shows the `SET_CREDENTIAL_INFO` request message format:

278 **Table 25 — SET_CREDENTIAL_INFO request message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x86` = `SET_CREDENTIAL_INFO`. See Table 12 — Authorization Message request codes. |
| 1 | Reserved | 1 | Reserved. |
| 2 | SetCredInfoOp | 1 | The field indicates the requested operation. The format of this field shall be the format as Table 26 defines. |
| 3 | CredInfoParams | Variable | This field represent 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. |

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

| Value | Operation Name | Description |
|---|---|---|
| 0 | Reserved | Reserved |
| 1 | ParameterChange | Shall indicate an operation that modifies credential parameters associated with the given credential IDs. |
| 2 | Erase | Shall indicate an operation that erases all credential parameters associated with the given credential IDs. All fields except for `StructVersion` and `CredentialID` shall be absent. |
| All other values | Reserved | Reserved |

280 Table 27 — Successful SET_CREDENTIAL_INFO_RSP response message format shows the successful `SET_CREDENTIAL_INFO_RSP` response message format:

281 **Table 27 — Successful SET_CREDENTIAL_INFO_RSP response message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x06` = `SET_CREDENTIAL_INFO_RSP`. See Table 13 — Authorization Message response codes. |
| 1 | Reserved | 1 | Reserved. |

**282** **9.2.5.1.1 Additional Requirements on SET_CREDENTIAL_INFO**

283 An Authorization target shall prohibit an Authorization initiator from erasing their own credentials.

**284** **9.2.5.2 GET_CREDENTIAL_INFO request and GET_CREDENTIAL_INFO_RSP response messages**

285 The `GET_CREDENTIAL_INFO` request shall be used to retrieve information about credentials provisioned in a credential slot. If an invalid credential slot or credential slot that is not provisioned is provided as input, the authorization target shall respond with `AUTH_ERROR` and `ErrorCode=InvalidRequest`.

286 Table 28 — GET_CREDENTIAL_INFO request message format shows the `GET_CREDENTIAL_INFO` request message format:

287 **Table 28 — GET_CREDENTIAL_INFO request message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x87` = `GET_CREDENTIAL_INFO`. See Table 12 — Authorization Message request codes. |
| 1 | Reserved | 1 | Reserved. |
| 2 | CredentialID | 2 | Credential ID for which information is required. |

288 Table 29 — Successful GET_CREDENTIAL_INFO_RSP response message format shows the successful `GET_CREDENTIAL_INFO_RSP` response message format:

289 **Table 29 — Successful GET_CREDENTIAL_INFO_RSP response message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 1 | RequestResponseCode | 1 | Shall be `0x07` = `SET_CREDENTIAL_INFO_RSP`. See Table 13 — Authorization Message response codes. |
| 2 | Reserved | 1 | Reserved. |
| 3 | CredInfoParams | Variable | This field represent identity information associated with the requested Credential ID. The size and format of this field shall be the same size and format as Table 2 — Credential Structure defines. |

290 #### 9.2.5.3 Credential provisioning authorization requirements

291 The Authorization target shall perform authorization checks for `SET_CREDENTIAL_INFO` and `GET_CREDENTIAL_INFO` requests except for the scenarios that Initial provisioning details.

292 ## 9.2.6 Credential policy provisioning and management

293 #### 9.2.6.1 SET_CREDENTIAL_POLICY request and SET_CREDENTIAL_POLICY_RSP response messages

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

295 The Authorization Initiator shall use the `Flags` parameter to specify if the request is to associate the list of credential policies, modify an existing policy or to erase the list of existing policies. An authorization target shall ensure that the operation is atomic, that is, all policies in the `PolicyList` can successfully be provisioned, modified or erased, and fail if that is not possible. All modify and erase operations, including on credential slot 0, shall be authorized by an existing credential.

296 Table 30 — SET_CREDENTIAL_POLICY request message format shows the `SET_CREDENTIAL_POLICY` request message format:

297 **Table 30 — SET_CREDENTIAL_POLICY request message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x88` = `SET_CREDENTIAL_POLICY`. See Table 12 — Authorization Message request codes. |
| 1 | Reserved | 1 | Reserved. |
| 2 | SetCredPolicyOp | 1 | The field indicates the requested operation. The format of this field shall be the format as Table 31 defines. |
| 3 | PolicyList | Variable | This field represents the policy information to change that is associated with the given Credential ID. This field shall only represent the policies associated with a single Credential ID.<br>The size and format of this field shall be the same size and format as Table 3 — Policy List defines. |

298 **Table 31 — Values for `SetCredPolicyOp` field**

| Value | Operation Name | Description |
|---|---|---|
| 0 | Reserved | Reserved |
| 1 | ParameterChange | Shall indicate an operation that modifies the credential policy associated with the given credential IDs. |
| 2 | Erase | Shall indicate an operation that erases all credential polices associated with the given credential IDs. An erase operation shall clear all bits in the policy. |
| All other values | Reserved | Reserved |

299 Table 32 — Successful SET_CREDENTIAL_POLICY_RSP response message format shows the successful `SET_CREDENTIAL_POLICY_RSP` response message format:

300 **Table 32 — Successful SET_CREDENTIAL_POLICY_RSP response message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x08` = `SET_CREDENTIAL_POLICY_RSP`. See Table 13 — Authorization Message response codes. |
| 1 | Reserved | 1 | Reserved. |

**301** **9.2.6.1.1 Additional requirements on SET_CREDENTIAL_POLICY**

302 An Authorization target shall prohibit an Authorization initiator from erasing their own policy.

#### 9.2.6.2 GET_CREDENTIAL_POLICY request and GET_CREDENTIAL_POLICY_RSP response messages

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

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

**Table 33 — GET_CREDENTIAL_POLICY request message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x89` = `GET_CREDENTIAL_POLICY`. See Table 12 — Authorization Message request codes. |
| 1 | Reserved | 1 | Reserved. |
| 2 | CredentialID | 2 | Credential ID for which information is required. |

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

**Table 34 — Successful GET_CREDENTIAL_POLICY_RSP response message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 1 | RequestResponseCode | 1 | Shall be `0x09` = `GET_CREDENTIAL_POLICY_RSP`. See Table 13 — Authorization Message response codes. |
| 2 | Reserved | 1 | Reserved. |
| 3 | PolicyList | Variable | This field represents all the policy information associated with the requested Credential ID. The size and format of this field shall be the same size and format as Table 3 — Policy List defines. |

#### 9.2.6.3 Credential policy authorization requirements

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

### 9.2.7 Authorization state management

#### 9.2.7.1 START_AUTH request and START_AUTH_RSP response messages

The `START_AUTH` request and `START_AUTH_RSP` messages are used to establish a User-specific authorization session as described in USAP. The Authorization target shall respond with an `AUTH_ERROR` with

`ErrorCode=UnexpectedRequest` or silently discard the request if a `START_AUTH` is received for a User with a corresponding active USAS. See USAP Error Handling for more information.

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

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

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x83` = `START_AUTH`. See Table 12 — Authorization Message request codes. |
| 1 | Reserved | 1 | Reserved. |
| 2 | CredentialID | 2 | A unique identifier to identify the credential and the credential slot. This also identifies the user for whom a USAS is started. |
| 4 | 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 user identified by `CredentialID`. |

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

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

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x03` = `START_AUTH_RSP`. See Table 13 — Authorization Message response codes. |
| 1 | Reserved | 1 | Reserved. |
| 2 | CredentialID | 1 | Shall be the `CredentialID` from the corresponding `START_AUTH` request. |
| 3 | NonceLen | 1 | Length of the `Nonce` field. Shall be 32 bytes for this version of the specification |
| 4 | Nonce | NonceLen | Random sequence of bytes chosen by the authorization target. |

### 318 9.2.7.2 END_AUTH request and END_AUTH_RSP response messages

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

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

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

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x84` = `END_AUTH`. See Table 12 — Authorization Message request codes. |
| 1 | Reserved | 1 | Reserved. |
| 2 | CredentialID | 2 | A unique identifier to identify the credential and the credential slot. This also identifies the user for which a user-specific authorization session is started. |

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

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

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x04` = `END_AUTH_RSP`. See Table 13 — Authorization Message response codes. |
| 1 | Reserved | 1 | Reserved. |
| 2 | CredentialID | 1 | Shall be the `CredentialID` from the corresponding `END_AUTH` request. |

**324 9.2.7.3 ELEVATE_PRIVILEGE request and ELEVATE_PRIVILEGE_RSP response messages**

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

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

327 **Table 39 — ELEVATE_PRIVILEGE request message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x85` = `ELEVATE_PRIVILEGE`. See Table 12 — Authorization Message request codes. |
| 1 | Reserved | 1 | Reserved. |

328 Table 40 — Successful ELEVATE_PRIVILEGE_RSP response message format shows the `ELEVATE_PRIVILEGE_RSP` response message format:

329 **Table 40 — Successful ELEVATE_PRIVILEGE_RSP response message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x05` = `ELEVATE_PRIVILEGE_RSP`. See Table 13 — Authorization Message response codes. |
| 1 | Reserved | 1 | Reserved. |

#### 330 9.2.7.4 END_ELEVATED_PRIVILEGE request and END_ELEVATED_PRIVILEGE_RSP response message

331 `END_ELEVATED_PRIVILEGE` request and `END_ELEVATED_PRIVILEGE_RSP` 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.

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

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x86` = `END_ELEVATED_PRIVILEGE`. See Table 12 — Authorization Message request codes. |
| 1 | Reserved | 1 | Reserved. |

333 Table 42 — Successful END_ELEVATED_PRIVILEGE_RSP response message format shows the `END_ELEVATED_PRIVILEGE` response message format:

334 **Table 42 — Successful END_ELEVATED_PRIVILEGE response message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x06` = `END_ELEVATED_PRIVILEGE`. See Table 13 — Authorization Message response codes. |
| 1 | Reserved | 1 | Reserved. |

## 335 9.2.8 Basic Management

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

#### 337 9.2.8.1 AUTH_RESET_TO_DEFAULT request and AUTH_DEFAULTS_APPLIED response

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

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

340 **Table 43 — AUTH_RESET_TO_DEFAULT request message format**

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x8C` = `AUTH_RESET_TO_DEFAULT`. See Table 12 — Authorization Message request codes. |
| 1 | Reserved | 1 | Reserved. |
| 2 | DataType | 2 | This field indicates the type of data to reset back to default. The format of this field shall be the format as Table 44 — Data Type Bit Definition defines. |
| 4 | CredentialID | 2 | The value of this field shall indicate the credential ID(s) to reset back to default. The value of 0xFFFF shall indicate all credential IDs. |
| 6 | SVResetDataTypeCount | 1 | This field shall be the count of Standard or Vendor Reset Data Type Elements in `SVResetDataTypeList`. A value of zero shall indicate the absence of `SVResetDataTypeList`. |
| 8 | SVResetDataTypeList | Variable | This field shall cause data types defined by a standard body or vendor to restore back to their factory defaults. The format of this field shall be the concatenation of Standard or Vendor Reset Data Type Element as Table 45 — Standard or Vendor Reset Data Type Element Format defines.<br><br>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. |

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

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

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

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

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

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | SVResetDataTypeOwner | Len0 | This field shall specify the owner of the `SVResetDataType` field. The format and size of this field shall be the format and size of the SVH as SPDM defines.<br><br>If other DMTF DSP uses the format as this table defines, then the other DMTF DSP specifications shall use the value associated with DMTF-DSP for the `ID` field as SPDM defines. |
| `Len0` | SVResetDataTypeLen | 1 | The value of this field plus 1 shall specify the length of `SVResetDataType`. The value of this field shall not exceed 31, indicating a maximum of 32 bytes. |
| 1 + `Len0` | SVResetDataType | Variable | This field shall indicate the standard or vendor specific data types to restore back to factory defaults.<br><br>The `SVResetDataTypeOwner` defines the format and size for this field. |

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

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

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

| Byte offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | RequestResponseCode | 1 | Shall be `0x0C` = `AUTH_DEFAULTS_APPLIED`. See Table 13 — Authorization Message response codes. |
| 1 | Reserved | 1 | Reserved. |

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

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

350 ## 9.3 Timing Requirements

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

### 9.3.1 Authorization Messages Timing

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

### 9.3.2 All Messages requiring Authorization

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

# 10 Authorization Opaque Data Structures

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

## 10.1 General Authorization Opaque Data Structure

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

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

**Table 47 — AODS General Format**

| Byte Offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | ID | 1 | The value of this field shall be zero to identify DMTF as the standards body. |
| 1 | VendorIDLen | 1 | The value of this field shall be zero to identify DMTF as the owner of the definition of all AODS. |
| 2 | OpaqueElementDataLen | 2 | The value of this field shall be the total size of these fields: `DMTFspecID`, `AODSid` and `AODSbody` field. |
| 3 | DMTFspecID | 2 | The value of this field shall be 289. This field indicates that the definition of the `OpaqueElementData` belongs to this DMTF specification. |
| 5 | AODSid | 1 | This field identifies the AODS and its format in `AODSbody`. The value of this field shall be one of the values in the **AODS ID** column of Table 48 — AODS IDs. |
| 6 | 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. |
| 5 + `AODSbodyLen` | AlignPadding | Variable | See field of the same name in SPDM for definition and requirements. The `OpaqueElementData` are the fields following `DMTFspecID` inclusively but not including this field. |

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

## 10.2 AODS IDs

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

**Table 48 — AODS IDs**

| AODS ID | AODS Name | Description |
|---|---|---|
| 0 | INVOKE_SEAP | Shall invokes the SEAP process for an SPDM endpoint. The format of the `AODSbody` shall be the INVOKE_SEAP AODS. |
| 1 | SEAP_INVOKED | Shall acknowledge the `INVOKE_SEAP` request. The format of the `AODSbody` shall the SEAP_INVOKED AODS. |
| 2 | SEAP_SUCCESS | Shall indicate the SPDM secure session handshake phase of the SEAP process has successfully passed for the corresponding SPDM endpoint. The format of the `AODSbody` shall be the SEAP_SUCCESS AODS. |
| 3 | AUTH_HELLO | Shall indicate the SPDM endpoint supports being an Authorization target |
| All other values | Reserved | Reserved |

## 10.3 INVOKE_SEAP AODS

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

**Table 49 — INVOKE_SEAP Body Definition**

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

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

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

## 10.4 SEAP_INVOKED AODS

The SEAP_INVOKED AODS shall acknowledge the `INVOKE_SEAP` request which invokes the SEAP process for the requesting SPDM endpoint. Additionally, the presence of this AODS shall indicate the responding SPDM endpoint supports SEAP for the requesting SPDM endpoint for the corresponding session. Table 50 — SEAP_INVOKED body definition defines the format for the `AODSbody` in the AODS general format when AODS ID is 1.

**Table 50 — SEAP_INVOKED 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. |

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

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

## 10.5 SEAP_SUCCESS AODS

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

**Table 51 — SEAP_SUCCESS Body Definition**

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

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

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

## 10.6 AUTH_HELLO AODS

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

**Table 52 — AUTH_HELLO Body Definition**

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

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

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

# 11 Cryptographic Operations

This section describes or defines cryptographic functions specific to Authorization

## 11.1 Signature Generation and Validation

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

### 11.1.1 Signature algorithm references

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

### 11.1.2 Signature generation

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

The signature generation function takes this form:

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

The `AuthSign` function shall take these input parameters:

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

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

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

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

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

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

400 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.

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

402 **Table 53 — Combined SPDM prefix**

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

403 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.

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

#### 405 11.1.2.1 RSA and ECDSA signing algorithms

406 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.

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

408 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`.

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

410 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.

#### 411 11.1.2.2 EdDSA signing algorithms

412 These algorithms are described in RFC 8032.

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

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

##### 415 11.1.2.2.1 Ed25519 sign

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

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

##### 418 11.1.2.2.2 Ed448 sign

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

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

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

#### 422 11.1.2.3 SM2 signing algorithm

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

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

425 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`.

426 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.

427 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.

### 428 11.1.3 Signature verification

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

430 The signature verification function takes this form:

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

431 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

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

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

434 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.

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

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

#### 437 11.1.3.1 RSA and ECDSA signature verification algorithms

438 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.

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

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

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

442 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.

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

#### 444 11.1.3.2 EdDSA signature verification algorithms

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

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

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

448 #### 11.1.3.2.1 Ed25519 verify

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

450 `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.

451 #### 11.1.3.2.2 Ed448 verify

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

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

454 `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.

455 ### 11.1.3.3 SM2 signature verification algorithm

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

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

458 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`.

459 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.

460 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 $ID_A$.

461 `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.

# 12 Authorization event types

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

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.

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

**Table 54 — Authorization event types table**

| Event Type ID | Event Name | Requirement | Description |
|---|---|---|---|
| 0 | Reserved | Reserved | Reserved. |
| 1 | CredInfoChanged | Mandatory | A change to one or more parameters via the `SET_CREDENTIAL_INFO` has occurred for a Credential ID. |
| 2 | CredPolicyChanged | Mandatory | One or more parameters associated with `SET_CREDENTIAL_POLICY` has changed for a Credential ID. |
| All others | Reserved | Reserved | Reserved. |

## 12.1 Event type details

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

### 12.1.1 Credential info Changed event

An Authorization target shall use this event ( `EventTypeId=CredInfoChanged` ) to notify the Event Recipient as SPDM defines that the Authorization target made a change to one or more parameters by the `SET_CREDENTIAL_INFO` request.

The Credential Info Changed format table describes the format for `EventDetail` as SPDM defines.

**Table 55 — Credential Info Changed format**

| Offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | CredentialIdCount | 2 | Shall be the number of Credential IDs in `CredentialIdList` |

| Offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 2 | CredentialIdList | Variable | Shall be a list of Credential IDs whose credential information change through the `SET_CREDENTIAL_INFO` 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` . |

473 The Authorization initiator can issue `GET_CREDENTIAL_INFO` to obtain details of this change.

### 474 12.1.2 Credential policy changed event

475 An Authorization target shall use the Credential policy changed event ( `EventTypeId=CredPolicyChanged` ) to notify the Event Recipient as SPDM defines when one or more credential policies have changed through the `SET_CREDENTIAL_POLICY` request. The `EventDetail` format for this event type shall be as Credential Policy changed event details format defines. This event only indicates a single policy change. If more than one policy changes, then each change will have their own event.

476 Table 56 — Credential policy changed event details format describes the format for `EventDetail` for the `CredPolicyChanged` event.

477 **Table 56 — Credential policy changed event details format**

| Offset | Field | Size (bytes) | Description |
|---|---|---|---|
| 0 | CredentialID | 2 | Shall be the credential ID associated with the credential policy that changed. |
| 2 | PolicyOwnerID | PolicyOwnerIdLen | Shall identify the owner of the definition of the policy that changed. The format of this field shall be the SVH as SPDM defines. The length of this field shall be the length of the SVH. |
| 2 + `PolicyOwnerIdLen` | PolicyID | PolicyIdLen | Shall identify the actual policy, defined by `PolicyOwnerID` , that changed. The length of this field shall be defined by `PolicyOwnerID` . However, the length of this field shall not exceed four bytes to ensure the Authorization initiator retrieves additional details through the `GET_CREDENTIAL_POLICY` request and to prevent data overloading in the overall SPDM eventing mechanism.<br><br>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. |

478 The Authorization initiator can issue `GET_CREDENTIAL_POLICY` to obtain further details on the change.

# 13 ANNEX A (informative) change log

## 13.1 Version 1.0.0 (in progress)

- Initial release

# 14 Bibliography

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