Document Identifier: DSP2058

Date: 2022-01-04

Version: 1.1.0

Security Protocol and Data Model (SPDM) Architecture White Paper

Supersedes: 1.0.0

Document Class: Informational

Document Status: Published

Document Language: en-US

CONTENTS

14 1 Abstract

15 This white paper presents an overview of the SPDM architecture, its goals, and a high-level summary of its use within a larger solution. The intended target audience for this white paper includes readers interested in understanding the use of the SPDM to facilitate security of the communications among components of platform management subsystems.

16 > **Note:** This white paper refers to this architecture as the Security Protocol and Data Model (SPDM) architecture or SPDM.

17 The PMCI architecture focuses on intercommunication among different platform-management subsystem components in a standards-based manner across any management component implementation, independent of the operating system state. The SPDM architecture focuses on security relative to these communications.

18 This white paper is not a replacement for the individual SPDM specifications but provides an overview of how the specifications operate within a larger solution.

19 2 Foreword

20 The Platform Management Communications Infrastructure (PMCI) Working Group of the DMTF prepared the *Security Protocol and Data Model (SPDM) Architecture White Paper* (DSP2058).

21 DMTF is a not-for-profit association of industry members that promotes enterprise and systems management and interoperability. For information about the DMTF, see DMTF.

22 The PMCI Working Group defines standards to address *inside the box* communication interfaces among the components of the platform-management subsystem.

23 This version supersedes version 1.0 and its errata versions. For a list of the changes, see ANNEX A (informative) change log.

24 2.1 Acknowledgments

25 The authors want to acknowledge the following people:

26 **Editors:**

- Brett Henning — Broadcom Inc.
- Raghupathy Krishnamurthy — NVIDIA Corporation
- Masoud Manoo — Lenovo
- Viswanath Ponnuru — Dell Technologies

27 **Contributors:**

- Richelle Ahlvers — Broadcom Inc.
- Jeff Andersen — Google
- Lee Ballard — Dell Technologies
- Steven Bellock — NVIDIA Corporation
- Heng Cai — Alibaba Group
- Patrick Caporale — Lenovo
- Yu-Yuan Chen — Intel Corporation
- Andrew Draper — Intel Corporation
- Nigel Edwards — Hewlett Packard Enterprise
- Daniil Egranov — Arm Limited
- Philip Hawkes — Qualcomm Inc.
- Jeff Hilland — Hewlett Packard Enterprise

- Yi Hou — Microchip
- Guerney Hunt — IBM
- Yuval Itkin — NVIDIA Corporation
- Theo Koulouris — Hewlett Packard Enterprise
- Benjamin Lei — Lenovo
- Luis Luciani — Hewlett Packard Enterprise
- Donald Matthews — Advanced Micro Devices, Inc.
- Mahesh Natu — Intel Corporation
- Chandra Nelogal — Dell Technologies
- Edward Newman — Hewlett Packard Enterprise
- Jim Panian — Qualcomm Inc.
- Scott Phuong — Cisco Systems Inc.
- Jeffrey Plank — Microchip
- Xiaoyu Ruan — Intel Corporation
- Nitin Sarangdhar — Intel Corporation
- Vidya Satyamsetti — Google
- Hemal Shah — Broadcom Inc.
- Srikanth Varadarajan — Intel Corporation
- Peng Xiao — Alibaba Group
- Qing Yang — Alibaba Group
- Jiewen Yao — Intel Corporation

28
3 References

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

- DMTF DSP0236, *MCTP Base Specification 1.3.0*
- DMTF DSP0239, *Management Component Transport Protocol (MCTP) IDs and Codes*
- DMTF DSP0274, *Security Protocol and Data Model (SPDM) Specification 1.1.1*
- DMTF DSP0275, *Security Protocol and Data Model (SPDM) over MCTP Binding Specification 1.0.0*
- DMTF DSP0276, *Secured Messages using SPDM over MCTP Binding Specification 1.0.0*
- DMTF DSP0277, *Secured Messages using SPDM Specification 1.0.0*
- DMTF DSP2015, *Platform Management Components Intercommunication (PMCI) Architecture White Paper 2.0.0*
- IETF TLS DTLS13-43, *The Datagram Transport Layer Security (DTLS) Protocol Version 1.3 draft-ietf-tls-dtls13-43*, 30 April 2021
- RFC5280, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*
- NIST SP 800-57, *NIST SP 800-57 Part 1 Rev. 4, Recommendation for Key Management, Part 1: General*
- NIST SP 800-90, *NIST SP 800-90A Rev. 1, Recommendation for Random Number Generation Using Deterministic Random Bit Generators*
- NIST SP 800-193, *NIST SP 800-193, Platform Firmware Resiliency Guidelines*
- *USB Authentication Specification Rev 1.0 with ECN and Errata through January 7, 2019*

30 # 4 Terms and definitions

31 This white paper uses terms that the following specifications define:

- *Security Protocol and Data Model (SPDM) Specification 1.1.1*
- *Security Protocol and Data Model (SPDM) over MCTP Binding Specification 1.0.0*
- *Secured Messages using SPDM over MCTP Binding Specification 1.0.0*
- *Secured Messages using SPDM Specification 1.0.0*

32 5 Introduction

33 5.1 Typographical conventions

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

34 5.2 Authentication

35 Enterprise computer platforms include many components that contain mutable elements. Each mutable component presents a potential vector for attack against the component itself, or even the use of a component to attack another component in the computer. To defend against these attacks, the *Security Protocol and Data Model (SPDM) Specification* enables conformant implementations to challenge a component to prove its identity and the correctness of its mutable component configuration.

36 An SPDM-conformant component generates, or is provisioned with, an asymmetric device public/private key pair. The component uses the device private key to sign requests, which proves knowledge of the private key. The Requester uses the device public key to authenticate the component-generated signature. For more details about the message exchanges, see Message details.

37 An SPDM-conformant component that is acting as a Responder can also perform authentication of the Requester, which is *mutual authentication*. By performing mutual authentication, the Responder can establish two-way trust with the Requester so that the two parties can establish a session.

38 5.3 Authenticated Encryption with Associated Data (AEAD)

39 SPDM-conformant components can establish an Authenticated Encryption with Associated Data (AEAD) session. When a Requester and Responder have established an AEAD session, the Requester and Responder establish shared keys that are used to protect communication between the two endpoints. The keys can be used for authenticated communication, or for authenticated and encrypted communication.

40 Components can establish a session to protect messages from unauthorized alteration (authenticated communication) or to protect messages from unauthorized observation and alteration (authenticated and encrypted communication). This protection of messages might be used for SPDM defined messages or messages defined by another specification, such as PLDM.

41 ## 5.4 Security Platform and Data Model (SPDM) architecture

42 A platform management subsystem in a modern enterprise computer platform comprises a set of components, which communicate to perform management functions within the platform. In many cases, these communications occur between components that comprise one or more mutable elements, such as firmware or software, re-programmable logic (FPGA), and re-programmable microcode. Further, a computer platform might contain immutable components, which comprise fixed logic or fixed firmware or software.

43 In such a platform management subsystem, stakeholders have a desire to establish trust, and to reestablish trust over time, with a component before securely communicating with that component.

44 The DMTF SPDM provides an authentication mechanism to establish trust, which uses proven cryptographic methods that protect the authentication process. As part of establishing trust between two endpoints, the SPDM specification enables the creation of a session to exchange secured messages between the endpoints.

45 For the purposes of this white paper, a component can encompass a number of component types, including PCIe adapters, Baseboard Management Controllers, purpose-built authentication components, Central Processing Units, platform components that are attached over I2C, and more. Each of these components represents a potential attack vector, through the insertion of counterfeit components, the compromise of firmware, or other attacks.

46 The SPDM enables these mechanisms to authenticate and secure communication with a component:

1. The retrieval of a public key certificate from a component, and a protocol to challenge the component to prove that it is the component whose identity is uniquely described by that certificate.
2. The retrieval of a signed measurement payload of mutable components from a component. These measurements can represent a firmware revision, component configuration, the Root of Trust for Measurements, hardware integrity, and more.
3. The negotiation of session keys with a component, enabling Secured Message exchanges with that component.

47 Finally, SPDM includes provisions for future expansion, by adding operations and capabilities while maintaining compatibility with existing deployments.

48 ## 5.5 SPDM standards overview

49 SPDM specifies a method for managed device authentication, firmware measurement, and certificate management. SPDM defines the formats for both request and response messages that enable the end-to-end security features among the platform-management components.

50 The SPDM specifications include:

- *Security Protocol and Data Model (SPDM) Specification 1.1.1 (DSP0274)*

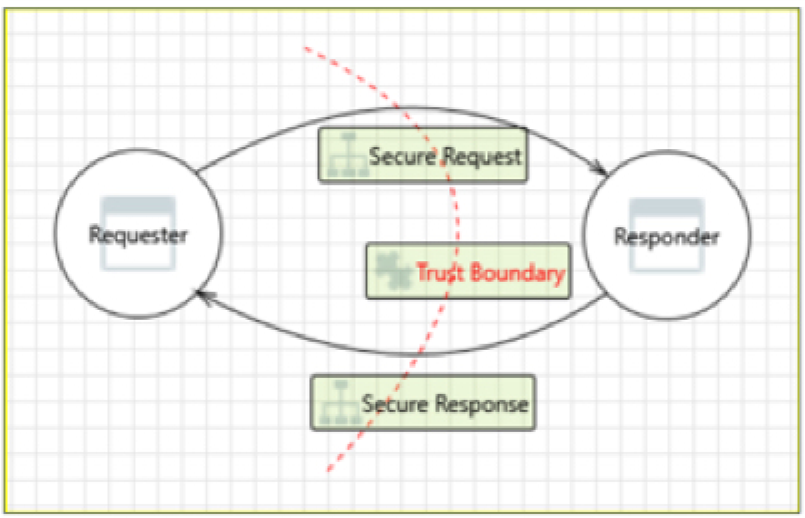
- *Security Protocol and Data Model (SPDM) over MCTP Binding Specification 1.0.0 (DSP0275)*
- *Secured Messages using SPDM over MCTP Binding Specification 1.0.0 (DSP0276)*
- *Secured Messages using SPDM Specification 1.0.0 (DSP0277)*

51 5.6 Threat model

52 The risk assessment identifies threats and vulnerabilities related to the SPDM interactions between components. Figure 1 — SPDM threat model shows the SPDM interaction between components. The following threat model follows the STRIDE model. See *STRIDE (security)* for more details.

53 **Figure 1 — SPDM threat model**

54



> 55 **Scope of this risk assessment:**
>
> 56 The scope of this assessment includes the security controls of the component as it comprises data model security and authentication. Any limitations of the physical I2C, I3C, PCIe, GenZ, CXL, or any other network channel shall not apply to this threat assessment.

57 Table 1 — Threat modeling assessment and mitigations describes the threat modeling assessment and mitigations:

58 **Table 1 — Threat modeling assessment and mitigations**

STRIDE category	Description	Justification mitigation
Spoofing	Packets or messages without sequence numbers or timestamps can be captured and replayed in a wide variety of ways. Implement or use a communication protocol that supports anti-replay techniques, which investigate sequence numbers before timers, and strong integrity.	To prevent replay attacks, the Requester and Responder shall use a random nonce.
Tampering	Attackers who can send a series of packets or messages might overlap data. For example, packet 1 might be 100 bytes starting at offset 0. Packet 2 might be 100 bytes starting at offset 25. Packet 2 overwrites 75 bytes of packet 1. Ensure that you both reassemble data before filtering it and explicitly handle these sorts of cases.	To prevent intruders from tampering with exchanged data, use one or more of these strategies: • Strong authorization schemes • Hashes • Message authentication codes • Digital signatures
Information Disclosure	Custom authentication schemes are susceptible to common weaknesses, such as weak credential change management, credential equivalence, easily guessable credentials, absent credentials, downgrade authentication, or a weak credential change management system. Consider the impact and potential mitigations for your custom authentication scheme.	To prevent attacks, use one or more of these strategies as supported by the endpoint components: • Stronger authentication schemes • Versions • Cryptographic algorithms
Elevation of Privilege	Requester or Responder might be able to impersonate the context of the Requester or Responder to gain additional privilege.	Out of scope. The endpoint that receives the request or response must mitigate this activity. The contents of the message are not interpreted at the MCTP layer.
Repudiation	Requester or Responder claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.	To mitigate attacks, use one or more of these strategies: • Digital signatures • Timestamps • Audit trails
Information Disclosure	Credentials on the wire are often subject to sniffing by an attacker. Are the credentials re-usable or re-playable? Are the credentials included in a message? For example, sending a ZIP file with the password in the email. Use strong cryptography for the transmission of credentials. Use the OS libraries, if possible, and consider cryptographic algorithm agility rather than hard-coding a choice.	To mitigate this attack, use stronger authentication schemes and cryptographic algorithms.
Denial of Service	Requester or Responder crashes, halts, stops, or runs slowly. In all cases, an availability metric is violated.	Out of Scope. To address uncorrectable errors or any type of crash, the Requester or Responder shall implement recovery mechanisms.

STRIDE category	Description	Justification mitigation
Denial of Service	External agent interrupts data flowing across a trust boundary in either direction.	If physical access is possible and the Start of Message and End of Message bits are not protected, a message can be dropped for one of the following reasons: 1. Receipt of the end packet for a message. 2. Receipt of a new start packet. 3. Timeout waiting for a packet. 4. Out-of-sequence packet sequence number. 5. Incorrect transmission unit. 6. Bad message integrity check. Only the whole MCTP message is secure. The individual MCTP packets are not secure.
Elevation of Privilege	Requester or Responder might be able to remotely execute code for the Responder.	Out of scope. The endpoint that receives the request or response must mitigate this activity. The contents of the message are not interpreted at the MCTP layer.
Elevation of Privilege	Attacker might pass data into The Requester or Responder to change the flow of program execution within Requester or Responder to the attacker's choosing.	Out of scope. The endpoint that receives the request or response must mitigate this activity. The contents of the message are not interpreted at the MCTP layer.

59

6 SPDM concepts

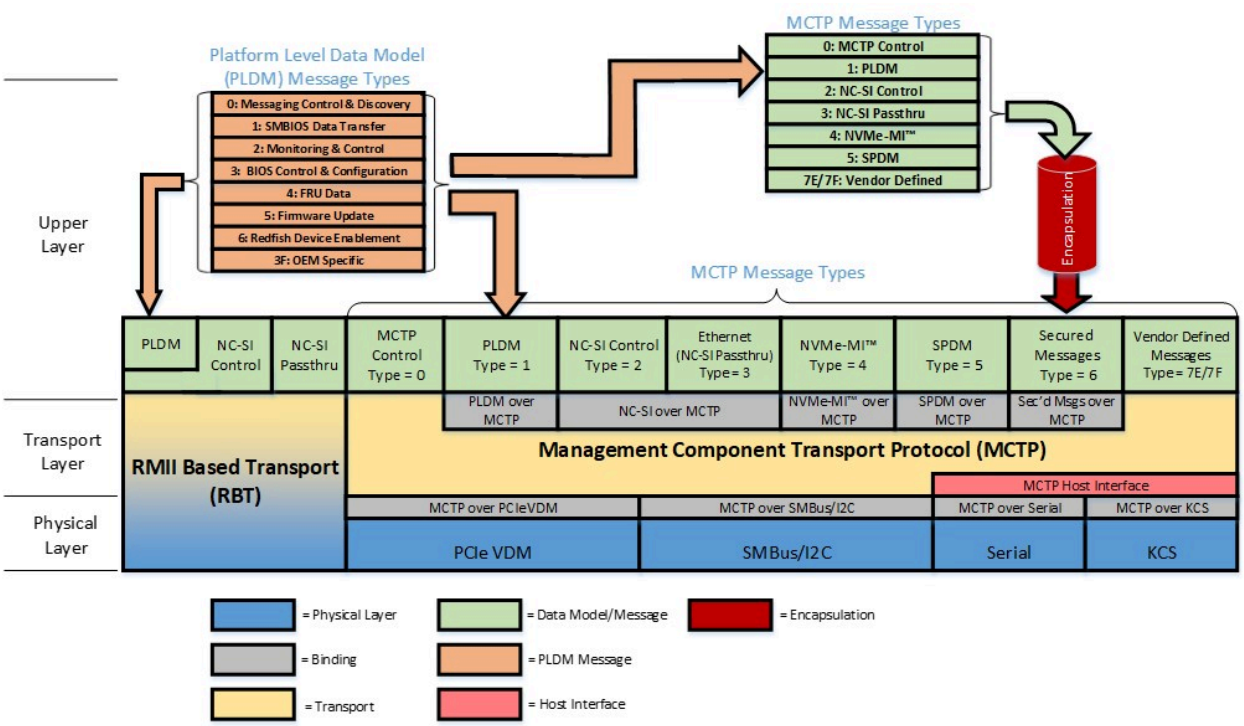
60

6.1 PMCI stack

61 Figure 2 — SPDM over MCTP shows the relationship among SPDM messages and other messages that use MCTP. Messages that the SPDM specification defines use MCTP message type 5, and might be used in conjunction with other MCTP message types. Messages that provide authentication support use MCTP message type 5. MCTP message type 6 is used in conjunction with other MCTP message types to enable Secured Messages.

62 **Figure 2 — SPDM over MCTP**

63

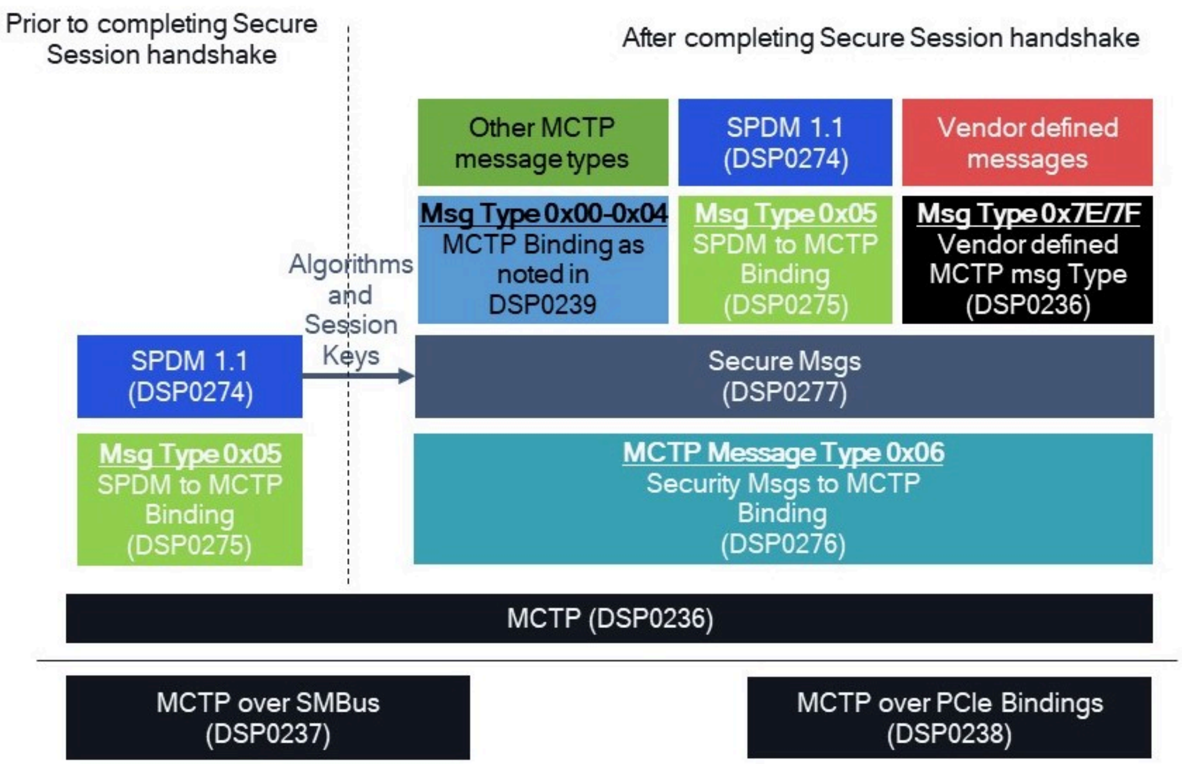


64 For details on the relationships among PMCI specifications, see the *Platform Management Components Intercommunication (PMCI) Architecture White Paper* (DSP2015).

65 Figure 3 — SPDM security stack shows the relationship among the security related specifications produced by the PMCI Working Group, and the relationships to other specifications produced by the PMCI Working Group.

66 **Figure 3 — SPDM security stack**

67



68 The *Security Protocol and Data Model Specification* (DSP0274) defines the contents of the messages, supported exchanges, and requirements.

69 The *Security Protocol and Data Model (SPDM) over MCTP Binding Specification* (DSP0275) defines the method for transporting SPDM messages over an MCTP transport.

70 The *Secured Messages using SPDM over MCTP Binding Specification (DSP0276)* binds Secured Messages using SPDM specification (DSP0277) to the MCTP transport.

71 The *Secured Messages using SPDM Specification (DSP0277)* defines the methodology that various PMCI transports can use to communicate various application data securely by utilizing SPDM.

72

6.2 Other bindings

73 Other standards bodies can create binding specifications that enable SPDM on transports other than those defined by DMTF. While many of the concepts in this white paper might apply to those implementations, the details of non-DMTF SPDM bindings are beyond the scope of this white paper.

74 For more information related to other binding specifications, see Partner implementations.

75 7 SPDM trusted computing base

76 The SPDM protocol provides authentication of devices and attestation of firmware running on a device. This means that the SPDM software stack becomes a part of the trusted computing base (TCB) for a device and a verifier, and the code must be implicitly trusted. As is typical of any TCB, a compromise in the TCB is undetectable and the trustworthiness of attestation reports are only as trustworthy as the TCB. There is no mechanism prescribed by the SPDM specification for protection, detection and recovery of the TCB. To provide higher security assurances around the TCB, device manufacturers and implementers can use methods outside the specification to protect, detect, and recover the TCB.

77 # 8 Certificates

78 If a Responder supports the certificate-related SPDM `GET_DIGESTS`, `GET_CERTIFICATE`, and `CHALLENGE` requests, the Responder must be provisioned with at least one certificate chain. If a Responder only supports the `GET_MEASUREMENTS` request, but cannot perform signature generation, it does not require a certificate chain or need to follow the guidance in the rest of this clause. A less capable component might be implemented in such a manner so that it does not require as much processing power or because such an implementation is conformant to the component's requirements. Whether a Requester accepts such a component is dependent on the Requester's security policy.

79 ## 8.1 Background on certificates

80 The SPDM specification uses X.509 v3 certificates, as defined in RFC5280, to communicate identity information between two components. The use of X.509 v3 certificates has the following advantages:

- When properly validated, X.509 v3 certificates are resistant to tampering.
- X.509 v3 certificates are standards based and widely supported.
- X.509 v3 certificates can use extensions to capture and convey other information, including information structures that DMTF defines.

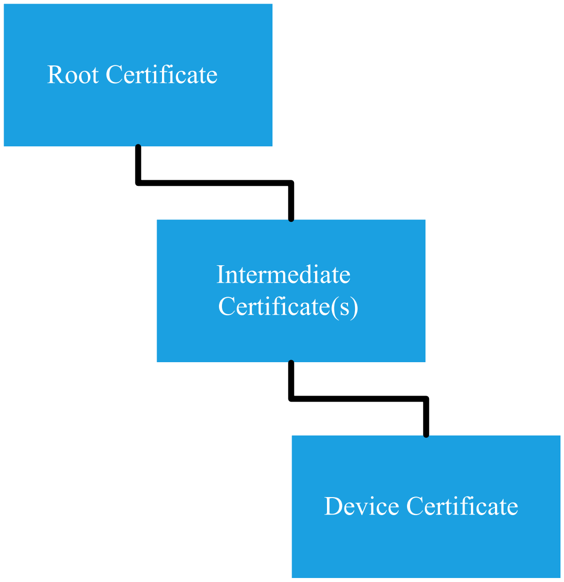
81 ## 8.2 Certificate overview

82 During the certificate-related SPDM request sequence, the Requester attempts to determine the identity of the Responder based on the certificate chain that the Responder returns. To report its identity, the Responder returns a chain of linked certificates that include at least a device certificate and a certificate issued by a CA that the Requester trusts. The certificate that the Requester trusts could be a root certificate or an intermediate certificate.

83 Figure 4 — Example certificate chain shows an example certificate chain:

84 **Figure 4 — Example certificate chain**

85



86 Table 2 — Certificate chain elements summarizes the roles of the elements that Figure 4 — Example certificate chain shows.

87 **Table 2 — Certificate chain elements**

Certificate chain element	Description
Root certificate	Conceptually the highest certificate in the chain. Contains a record of the issuing authority and is self-signed.
Intermediate certificate	A certificate chain typically contains one or more of these certificates, which enable the allocation of separate intermediate certificates to different device families or product divisions within a company. This enables flexibility in establishing complex hierarchies of certificates for easier revocation and to protect the root certificate private key which might be kept offline.
Device certificate	Uniquely identifies the component. Should not change over the life of a component, unless the component is re-provisioned. If an operation changes the Device key pair, then the Device certificate must be replaced. The lowest level certificate in a certificate chain is called the leaf certificate.

88 8.2.1 Certificate chain validation

89 Before a Requester uses the contents of a certificate chain, it must validate the certificate chain to ensure that it is properly formed. RFC5280 specifies the detailed process for validating a certificate chain. To assist the reader, the process is summarized here (note, the discussion in this section is based on the diagram in Figure 4 — Example certificate chain):

- Check each certificate to ensure that it references the certificate above it in the chain.

- Validate the signature in each certificate using the public key from the certificate above it in the chain.
- Read the validity dates, key usage policies, and other constraining information from the certificates to verify that the certificate and its associated key pair are being used correctly.
- Ensure that the root certificate is a known and trusted certificate.

90 After the RFC5280 based certificate chain validation is complete, the Requester knows that the certificate chain is correctly formed but this information is insufficient. The Requester still must ensure that the Responder is the component that should be returning this certificate chain. This check is performed by verifying that the Responder has knowledge of the private key associated with the public key in the leaf certificate by using the `CHALLENGE` message exchange.

91 8.3 SPDM certificate slots

92 The SPDM specification defines a total of eight slots for storing certificate chains, with each slot storing a complete and independent certificate chain. Further, the SPDM specification states that the component uses the same asymmetric key pair for the leaf certificate located in each slot. The certificate chain for each slot can contain different root certificates. While SPDM supports up to eight certificate slots, only slot 0 is required to be present for components that use certificates. Further, a component can implement fewer than eight certificate slots, such as three slots.

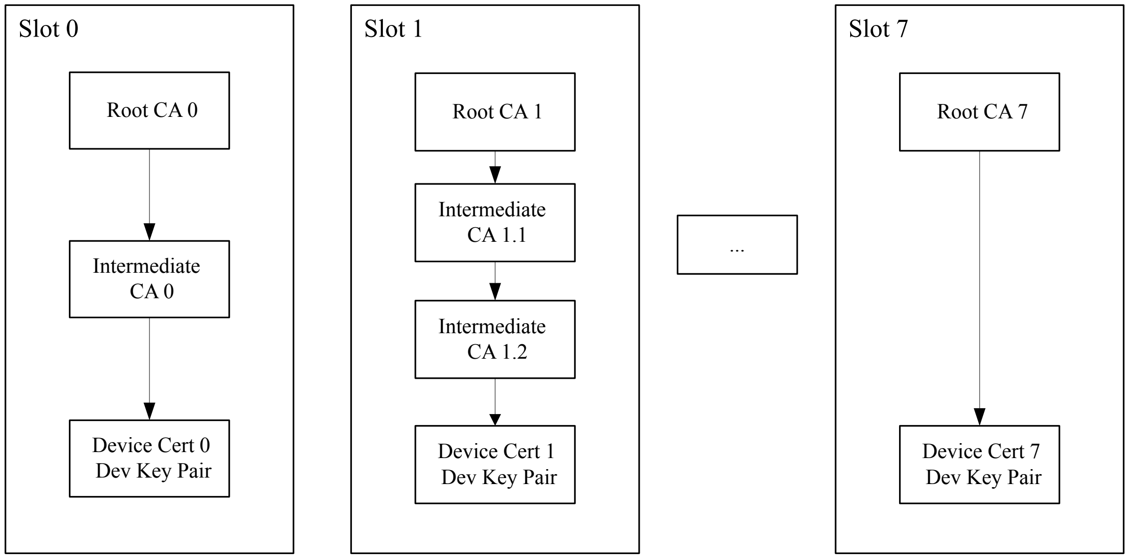
93 The certificate chain in slot 0 has a special role in the system because the component manufacturer provisions the contents of slot 0 during manufacturing. The certificate chain in slot 0 represents the manufacturer, and this certificate chain is often immutable, though immutability is not required by the SPDM specification. This certificate chain is also known as the *manufacturer certificate chain*.

94 Some deployment use cases might make use of certificate slots 1 to 7. For instance, an administrator can claim ownership of a component by installing a certificate chain belonging to the administrator in one or more of the additional slots (certificate slots 1 to 7). The use of these additional slots enables the administrator to authenticate the component using a certificate chain that is owned and managed by the administrator. Another use of additional certificate slots is to set certificate validity ranges that expire in a shorter time-frame than the certificate chain installed by the component vendor.

95 Figure 5 — Example certificate slots shows an example of the use of certificate slots:

96 **Figure 5 — Example certificate slots**

97



98 8.3.1 Stored certificate chain format

99 The SPDM specification indicates that the certificate chain returned to the Requester are formatted where the first certificate is signed by the root certificate, or is the root certificate itself, and each subsequent certificate is signed by the preceding certificate until the leaf certificate. The returned certificate chain is also to include a hash of the root certificate. Implementers are recommended to store the entire certificate chain in a slot, including the root certificate, so that the hash can be generated with the currently negotiated algorithm.

100 A Responder can choose to send one of the two certificate chain formats (with or without the root certificate) depending on the situation. For instance, a Responder could send the certificate chain formatted without the root certificate when using a slower transport.

101 8.4 Certificate chain algorithms

102 A certificate chain is implicitly tied to a pairing of `BaseAsymAlgo` and `BaseHashAlgo`, as the `ALGORITHMS` message exchange defines. The negotiated `BaseAsymAlgo` and `BaseHashAlgo` fields must match the algorithms used to create the certificate chain on the Responder. For compatibility purposes, a component vendor can provision a component with certificate chains that correspond to multiple `BaseAsymAlgo` and `BaseHashAlgo` pairings. For instance, a component can have one set of certificate chain slots that it uses to pair `TPM_ALG_ECDSA_ECC_NIST_P384` and `TPM_ALG_SHA3_384`, and another set of certificate chain slots that it uses to pair `TPM_ALG_RSASSA_3072` and `TPM_ALG_SHA_256`. In this case, the Responder uses the negotiated algorithm set to select among its different sets of certificate chain slots. In such an implementation, it's feasible that the populated certificate slots could differ between

the different sets of certificate chain slots. The definition and reporting of a slot management mechanism such as this is outside of scope for the SPDM specification.

103 8.4.1 Certificate chain verifier compatibility

104 The set of cryptographic algorithms that the Requester and Responder negotiate during the `ALGORITHMS` exchange match the cryptographic algorithms use the in the leaf certificate. However, a Responder typically returns a certificate chain with multiple certificates in the `CERTIFICATE` response. When validating the returned certificate chain, the Requester should not assume that all certificates in the certificate chain use the same cryptographic algorithms as the leaf certificate. For the sake of compatibility, a Responder should constrain itself to use cryptographic algorithms specified in the SPDM `NEGOTIATE_ALGORITHMS` exchange, and Requesters should support the use of all cryptographic algorithms specified in the SPDM `NEGOTIATE_ALGORITHMS` exchange.

105 8.5 Certificate requirements

106 Certificate chains follow the X.509 v3 format, and are DER-encoded. Certificate chains can be long compared to other SPDM messages, so Requesters should ensure that buffers are large enough to receive them. The maximum length of a certificate chain that can be conveyed by SPDM is 64 KiB. The support to verify signatures of different cryptographic algorithms on the certificate chains remain the responsibility of Requester and Responder software stacks. It is expected that they support verification of commonly accepted algorithms to promote interoperability.

107 The leaf certificate in the certificate chains must conform to the SPDM specification, *Leaf certificate* clause defined format. The certificate format guidance in SPDM is based on RFC5280. Table 2 — Optional leaf certificate attributes describes the leaf certificate attributes that the SPDM specification specifies as optional.

108 8.5.1 Certificate retrieval

109 If a Requester cannot allocate a buffer for the maximum certificate chain size of 64 KiB, the Requester can issue a `GET_CERTIFICATE` request with the `Length` field set to a small number, such as four bytes. In this case, the Responder returns the requested portion of the certificate chain and the remaining length in the `RemainderLength` field. SPDM provides a mechanism to segment a certificate chain using the `Offset` and `Length` fields in the `GET_CERTIFICATE` request to retrieve the certificate chain in smaller increments. This mechanism can compensate for Requesters, Responders, or transports that cannot transfer an entire certificate chain in one response message.

110 A Requester should anticipate that a Responder might not be capable of sending the entire certificate chain in one transaction, even if the Requester is capable of allocating a sufficiently large buffer.

111 8.5.2 Certificate fields

112 X.509 v3 certificates contain multiple fields, as defined by RFC5280. In addition, the SPDM specification specifies usage of some X.509 v3 defined fields.

113 **Table 2 — Optional leaf certificate attributes**

Attribute	Description
`Validity` (`notBefore`)	If present, it is recommended that the `notBefore` field of the `Validity` attribute should be set to `19700101000000Z` , which is the minimum `Validity` date. Because most Requester and Responder pairs do not contain a real-time clock, the use of the minimum `Validity` date ensures that the Requester ignores the `notBefore` field.
`Validity` (`notAfter`)	If present, it is recommended that the `notAfter` field of the `Validity` attribute should be set to `99991231235959Z` , which is the maximum `Validity` date. Because most Requester and Responder pairs do not contain a real-time clock, the use of the maximum `Validity` date ensures that the Requester ignores the `notAfter` field.
`Subject Alternative Name`	Recommended. It enables reporting of more detailed and standardized component identification.

114 Though not required, the SPDM specification details the `Subject Alternative Name` for components that are SPDM conformant. Standards bodies that create additional binding specifications for SPDM should specify appropriate guidelines for the `Subject Alternative Name` and Common Name fields (see Partner implementations). All standards bodies that use the SPDM specification should retain the `Serial Number` field in the certificate definition.

115 A certificate should use the `otherName` field in the `Subject Alternative Name` to provide detailed information about the manufacturer, product, and serial number.

116 The OID in the `othername` field is `1.3.6.1.4.1.412.274.1` . This value represents a UTF8String in the `<manufacturer>:<product>:<serialNumber>` format.

117 The following example string shows the format of the SPDM defined `Subject Alternative Name` `otherName` field:

```
othername:1.3.6.1.4.1.412.274.1;UTF8STRING:ACME:WIDGET:0123456789
```

118 The X.509v3 certificates can include the `Authority Key Identifier` , which assists authentication of the certificate chain. This assistance is especially important for the certificate that is immediately below the root certificate because the `Authority Key Identifier` can help the Requester locate the root certificate in its trust store. The presence of the `Authority Key Identifier` can also help with debug of certificate chain problems, by illustrating how certificates are intended to connect.

119 8.6 Interpreting certificate contents

120 A certificate chain contains information that a Requester can interpret to make policy decisions about a given Responder. Once a certificate chain has been validated, as described in Certificate chain validation, a Requester can

use the Certificate fields to interpret the information contained in the certificate chain. While many of the fields are interpreted as defined in RFC5280, some fields are defined by the SPDM specification.

121 Table 3 — Interpretation of select certificate fields summarizes potential use cases for select SPDM specification defined Certificate fields.

122 **Table 3 — Interpretation of select certificate fields**

Field	Required or optional	Interpretation
`Subject Alternative Name` `otherName`	Optional	The `otherName` field provides identifying details for the component in a machine parsable manner. A Requester could use this field to match the identity of the component with the same information obtained through other channels, to create an entry for the component in a database, or to display information about the component to a user.

123 8.7 Example leaf certificate

124 The following example shows a leaf certificate:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4097 (0x1001)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = US, ST = NC, L = City, O = ACME, OU = ACME Devices, CN = CA
        Validity
            Not Before: Jan  1 00:00:00 1970 GMT
            Not After : Dec 31 11:59:59 9999 GMT
        Subject: C = US, ST = NC, L = City, O = ACME Widget Manufacturing, OU = ACME Widget Manufacturing Unit, CN
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:cc:41:73:a3:f1:ff:78:ff:78:f5:e1:a7:3c:2e:
                    ae:40:82:db:04:eb:ad:e8:54:e7:8f:4a:76:3c:a2:
                    21:77:72:e7:70:a6:0a:b3:7a:a3:e8:af:49:5c:ec:
                    57:00:6b:6e:0b:09:b7:f0:be:35:c4:ec:e8:f8:28:
                    0c:0a:b8:59:48:a7:14:47:88:05:c5:8c:1e:e5:79:
                    5a:2b:31:fe:14:27:12:eb:ba:53:40:74:43:5b:e0:
                    f4:be:45:93:f8:87:b6:a3:13:f1:7c:72:5f:c1:aa:
                    a6:be:fd:e8:c4:3a:ae:24:0e:81:25:c6:f2:6c:fd:
                    53:27:89:4c:f6:37:22:cf:25:5d:51:b9:30:54:61:
                    fe:0b:23:2f:dd:e3:1b:87:30:a4:b3:16:41:48:51:
                    1e:17:29:3a:2b:57:1c:41:67:27:62:15:08:6e:c1:
                    59:8d:d7:c3:0f:33:05:26:a0:1b:b9:f5:b4:36:0d:
```

```
                bb:ec:24:5d:bb:c9:0b:b2:57:1b:7b:18:21:d4:c0:
                ec:fd:0a:03:33:4e:b0:55:e7:3f:26:b1:96:1f:b3:
                2a:18:2d:88:4d:cd:9c:26:08:2c:d7:fc:5f:87:b4:
                e8:06:ad:6d:ce:65:0f:88:26:85:7d:aa:54:6d:57:
                34:34:ae:40:83:15:ee:cf:2c:06:ee:69:52:92:9b:
                b0:77
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Basic Constraints:
            CA:FALSE
        X509v3 Authority Key Identifier:
            CB:0C:55:D9:4F:18:EE:B9:54:25:3D:08:1A:4C:02:24:80:BF:CF:FE
        X509v3 Key Usage: critical
            Digital Signature
        X509v3 Subject Alternative Name:
            othername: 1.3.6.1.4.1.412.274.1::ACME:WIDGET:0123456789
Signature Algorithm: ecdsa-with-SHA256
Signature Value:
    30:44:02:20:3d:c9:e5:59:43:a5:f1:56:3e:8f:cb:ef:96:e1:
    bc:4d:bd:ca:d1:a7:69:7e:10:0e:58:74:5b:89:2a:b4:b2:59:
    02:20:2a:0d:95:4e:52:05:c0:fe:44:7b:61:ec:38:f7:87:95:
    8b:60:c5:89:03:d8:4e:c4:1c:0b:57:a3:de:67:45:83
```

125 8.8 Certificate provisioning

126 If a component supports the SPDM certificate related commands, the manufacturing process for that component must provision a certificate chain to each component instance.

127 A possible sequence of commands to create a certificate chain include:

- Generate a certificate signing request (CSR) using the firmware of the component.
- Export the information required to form a CSR to an external utility, which generates the CSR.
- If a component uses an externally provisioned key, generate the necessary certificate as part of the external key-generation process and load the generated key and certificate chain into the component. See Key provisioning.
- After import, the component checks the certificate chain to ensure that its public key matches the components Device public key.

128 This type of mechanism could be used to provision a certificate chain to one of the slots numbered 1-7. Such mechanisms are outside of scope for this white paper and are not part of the SPDM specification.

129 Any approach for generating a certificate chain should occur as part of a secure manufacturing process. Keep intermediate certificates above the device certificate in a secure environment that is not directly accessible to the component so that the component cannot sign a device CSR.

130 ## 8.9 Device key pair

131 Each component must contain a public and private key pair, or a device key pair, that is statistically unique to that component. The component must retain the same device key pair for the life of the component. Any operation that alters the device key pair invalidates any certificate chain that uses it, which causes the component to fail any authentication request that depends on the current certificate chain.

132 Only one device key pair should be used for any of the occupied certificate chain storage slots. The SPDM specification supports multiple encryption and hashing algorithms. The component manufacturer chooses the algorithm for the leaf certificate from the available list in accordance with the needs of the manufacturer.

133 ### 8.9.1 Key provisioning

134 There are two primary methods for provisioning a device key pair to a component, though there are multiple mechanisms available to accomplish each of the methods. Any component that supports SPDM certificate or measurement-related command sets must provision device key pairs.

135 #### 8.9.1.1 Internal key generation

136 If capable, a component should generate its own device key pair. A component can better protect a device private key that it generates on the component by ensuring that the device private key never made visible outside of the component.

137 This process must be a repeatable process that always results in the generation of the same device key pair because this is the foundation of the identity of the component. A component that generates its own device key pair can follow a model, such as the DICE model of the Trusted Computing Group, that results in a key pair of similar quality.

138 A component that generates its own device key pair must:

- Be provisioned with or generate and retain a cryptographically strong random number that can be used as the Unique Device Secret (UDS).
  - All random numbers and entropy sources should conform to the NIST SP800-90 standards.
- Have sufficient processing power or hardware support to generate a key pair by using the chosen algorithm.
- Protect the source data that the key generation process uses, as discussed in Key protection.

139 #### 8.9.1.2 External key provisioning

140 If a component cannot meet the requirements for internal key generation, it must use an external provisioning process. The external provisioning process allows the component manufacturer to rely on external tools and components, such as a Hardware Security Module (HSM), to meet requirements that the component cannot meet on

its own. For instance, a manufacturer can use an external tool to provide a true random number to a component that cannot generate sufficient entropy on its own, and use the component to complete the rest of the process.

141 External key provisioning has a trade-off because the component is in an open state until the component is provisioned with the device key pair. To maintain trust in the component, the supply chain and manufacturing facilities must be highly secure.

142 Any random number used as part of the key generation process should be generated in a manner that conforms to the NIST SP800-90 standards.

143 In some cases, a user might need to re-provision a device key pair that has been provisioned to a component. However, a component must ensure that re-provisioning cannot occur except when authorized by the user or the component is subject to a key hijack attack. The user must also ensure that the device key pair is only re-provisioned in a secure environment. The means to provide these protections is outside the scope of the SPDM specification.

144 8.9.2 Key protection

145 When using SPDM, the device key pair forms the foundation for proof of identity, and the device private key must be protected from disclosure to an unauthorized party. A component should ensure that the device key pair cannot be accessed, regenerated, or replicated if an attacker gains access to the component. The protection mechanisms should protect the secret values from access through debug ports, an API, or other interfaces.

146 Some items that the component should protect are:

- The basis of the component identity, such as the UDS.
- The device private key.
- Any values that were used to derive or store other protected values, such as a key encryption key for the device private key.
- When processing the SPDM specified Key Schedule, a component should erase input key material, such as `Salt_1` and the handshake secrets, as soon as they are no longer needed.

147 When the device private key is in plaintext form, it should only be stored in the internal memory of the component. To protect the device private key, the component should clear it from memory as soon as it is no longer needed. A component can use non-volatile memory to store its device private key but the non-volatile memory should be protected against unauthorized access, including attempts to gain physical access to the non-volatile memory, such as removing a flash part.

148 Any session keys should be protected from external observation and should be erased when no longer needed. Because the session keys typically exist during runtime, the protection should include protection against reads from a debug facility and reads through an API.

149 This protection can be implemented through a hardware mechanism that prevents unauthorized access. If the device key pair storage is protected through encryption, the encryption key must not be one of the device keys because this violates the NIST SP800-57 requirement that a key is used for only one purpose.

150 ## 8.10 Alternatives to certificate chains

151 ### 8.10.1 Pre-Shared Key

152 Components provisioned with a Pre-Shared Key might not require an asymmetric key pair or the use of X.509v3 certificates. Because the use of a Pre-Shared Key requires that the Requester and Responder both have knowledge of the Pre-Shared Key, the Requester can use the Responder's knowledge of the Pre-Shared Key as proof of the Responder's identity.

153 ### 8.10.2 Provisioned public key

154 As an alternative to certificates, a Requester can support the ability to export a public key. This capability is reported by setting `PUB_KEY_ID_CAP=1` in the `CAPABILITIES` exchange. The use case for this capability includes enabling devices that are not able to manage X.509 certificates. In this mode, the Responder's public key is provisioned to the Requester. Following is an example sequence for this provisioning process:

1. The Responder generates or is provisioned with a key pair. See Key provisioning for more details.
2. In a secure environment, the Responder's public key is provisioned to the Requester. The means by which the Responder's public key is provisioned to the Requester is outside of the scope of the SPDM specification, and might use a component's private API.
3. After deployment, the Responder signs responses (when required) using the private key that corresponds to the public key that was provisioned to the Requester. To maintain security, the Responder must protect the private key, as noted in Key protection.

155 > **Note:** The previous provisioning step must occur in a secure environment. Because the public key is not part of a certificate, which is endorsed by a trusted root certificate, the source of the public key cannot be programmatically verified. Instead, the security associated with the public key must be enforced through physical security. Vendors should also provide protections to ensure that once a public key has been provisioned, another one cannot be provisioned for the same purpose unless authorized to do so. Further, the user should ensure that all affected components are placed back in a secure environment before any re-provisioning occurs.

156 # 9 SPDM messages

157 ## 9.1 Compatibility between versions

158 ***Version encoding*** in the SPDM specification discusses the standard for determining whether changes are considered backwards compatible when determining whether a change causes a minor or major version update. This section provides additional discussion of the thought process behind this standard.

159 As the SPDM specification is a security specification, it is not reasonable to expect the SPDM specification to allow implementations that use different versions of the SPDM specification to interoperate without any modifications. Instead, the SPDM specification requires both the Requester and Responder to agree on the same major and minor versions in order to interoperate. This requirement can require a component to implement a solution that supports multiple versions of the SPDM specification, taking into account the behavioral differences between them.

160 Other than the `VERSION` exchange, the SPDM specification does not impose a requirement for backwards compatibility to previous specification versions (major or minor). A component vendor can choose to remove support for earlier versions of the SPDM specification for reasons of solution simplification or due to the vendor's security policy.

161 The SPDM specification might change computations and other operations between different minor versions of the specification. These changes are only allowed when the differences are dependent on the value in the `SPDMVersion` field. With this standard in place, an implementation might need to perform different operations depending on the SPDM specification version in use. See the following pseudo-code for an example of the type of operational difference that is considered acceptable under this standard.

```
/* compute a signature over input 'data' */
if (spdm_version == 0x10)
    spdm10_compute_signature(data);
else if (spdm_version == 0x11)
    spdm11_compute_signature(data);
```

162 The SPDM specification can add new values to bit fields and enumerations in newer minor versions though the existing values are retained (though possibly deprecated). The SPDM specification makes every effort to ensure bit-wise compatibility with previous versions to ease the implementation burden. Implementers should take care to use fields as defined. For instance, if an enumeration only provides 0 and 1 as possible values, an implementer should be careful not to use bit-wise operations with the field as future versions of the SPDM specification might expand the list of enumerated values to 0, 1, and 2.

163 The SPDM specification can add functionality to fields that were reserved in previous minor versions. Because reserved fields are defined as being set to 0, newer minor versions of the SPDM specification can safely add

functionality to reserved fields, using the value of 0 to indicate previous behavior. The following guidelines apply to reserved fields:

- A component always sets reserved fields to 0.
- Do not check the contents of reserved fields. The SPDM specification states that the contents of reserved fields are ignored by the receiver, which means that a receiver does not generate an error when a reserved field contains a non-zero value.
- Do not modify the contents of a reserved field, as this changes transcript hashes.

164 This behavior accommodates cases where a component that supports multiple minor versions of the SPDM specification might fill in information in reserved fields while operating at less than its highest supported minor version number, thus simplifying implementations.

165 Functionality that is no longer recommended for use is marked as deprecated. A component might receive a message with a value in a deprecated field, and the component can either process the message properly or return an error. Field and value definitions associated with deprecated items are not reused within minor revisions of the same major version.

166 9.2 Message details

167 9.2.1 GET_VERSION and VERSION exchange

168 The `VERSION` exchange creates an agreement between the Requester and the Responder on the major and minor SPDM version that they use for subsequent messages. The `VERSION` exchange remains backwards compatible in all future versions of SPDM.

169 A Requester must not issue commands or include parameters that the Responder does not support. The supported command and parameter set is determined by the agreed SPDM version and the Requester's and Responder's supported capabilities.

170 9.2.2 GET_CAPABILITIES and CAPABILITIES exchange

171 The `CAPABILITIES` exchange enables a Requester to query the SPDM capabilities that the Responder supports. The goals of the message exchange are:

- Enable a Requester and Responder to discover which optional message exchanges and capabilities the Responder and Requester supports
- Allow a Responder to inform the Requester of its cryptographic timeout requirements

172 The `CTExponent` enables a Responder to return its required cryptographic operation time. Because cryptographic operations can take longer than a non-cryptographic exchange, `CTExponent` enables the cryptographic timeout to

respond to the needs of the individual Responder. Because the SPDM supports a variety of component types, the `CTExponent` values for separate components in a system can vary greatly.

173 A Requester only issues commands that the Responder supports, with the supported command set determined by the agreed SPDM version and the Requester's and Responder's supported capabilities.

174 Per the `CAPABILITIES` flags, most commands in the SPDM specification are optional. These commands are optional to allow implementation flexibility for Responders. The Requester has responsibility to ensure that the Responder supports enough optional commands to satisfy the Requester's security policy.

175 9.2.2.1 CAPABILITIES flags

176 This clause provides background information on each of the optional capabilities in the `Flags` field in the `CAPABILITIES` response message.

177 Table 4 — Optional Flag field capabilities describes the optional capabilities in the `Flags` field in the `CAPABILITIES` response message:

178 **Table 4 — Optional Flag field capabilities**

Capability	Description
`CACHE_CAP`	If the Responder can cache certain messages through a reset, the Requester might skip issuing the cached requests after a reset and instead rely on cached values. If a Responder that sets `CACHE_CAP=1` has invalidated or lost its cached values, it responds to the next request, other than `GET_VERSION`, with an `ERROR` of `RequestResynch`, which indicates to the Requester that it is required to restart from `GET_VERSION`. See CACHE_CAP flag for more details.
`CERT_CAP`	`GET_DIGESTS` and `GET_CERTIFICATE` requests are related to each other. If a Responder supports `CERT_CAP`, it should also support `CHAL_CAP` and/or `MEAS_CAP`.
`CHAL_CAP`	Indicates support for `CHALLENGE`. Support for the `CHALLENGE` exchange is optional because a Responder might not support the cryptographic operations or other capabilities required for the `CHALLENGE_AUTH` response. A Requester might support a standalone `CHALLENGE` or use `MEASUREMENTS` to accomplish a challenge. However, Requesters should remember that if a Requester sends a `GET_MEASUREMENTS` without first completing a `CHALLENGE` exchange, the transcript is nullified and the Requester does not know whether an entity altered the response data.
`MEAS_CAP`	Indicates support for `MEASUREMENTS`. Support is optional because a Responder might not support the cryptographic operations or other capabilities required for the `MEASUREMENTS` response. A Requester might either support a standalone `CHALLENGE` or use `MEASUREMENTS` to accomplish a challenge operation.
`MEAS_FRESH_CAP`	Indicates whether the Responder supports the ability to recompute measurements in response to a `GET_MEASUREMENTS` request. The value of this capability can influence the Requester's policy. A device that does not support fresh measurements must be reset to capture new measurements.
`ENCRYPT_CAP`	Indicates support for encryption. Requires either `PSK_CAP` or `KEY_EX_CAP` so that keys can be established for the secure session. Use of `ENCRYPT_CAP` also requires use of `MAC_CAP`.

Capability	Description
MAC_CAP	Indicates support for authenticated messages. Requires either PSK_CAP or KEY_EX_CAP so that keys can be established for the secure session. Can be used with ENCRYPT_CAP.
MUT_AUTH_CAP	Indicates support for mutual authentication. If set, it requires support for encapsulated requests.
KEY_EX_CAP	Indicates support for key exchange, which is used with ENCRYPT_CAP or MAC_CAP.
PSK_CAP	Indicates support for Pre-Shared Key. Pre-Shared Key enables the use of Secured Messages by less capable devices. If supported, ENCRYPT_CAP or MAC_CAP are set.
HANDSHAKE_IN_THE_CLEAR_CAP	If set, the Responder can only send and receive SPDM defined messages without encryption and message authentication during the Session Handshake Phase. Whether a Requester accepts a Responder that does not set this bit is a function of the Requester's security policy.
PUB_KEY_ID_CAP	If set, the public key of the Responder was provisioned to the Requester using a mechanism that is outside of scope for the *SPDM Specification*.

179 9.2.2.2 CACHE_CAP flag

180 9.2.2.2.1 Multiple caching Requesters

181 For components that support CACHE_CAP, the support of a cached Negotiated State requires the component to be able to distinguish between Requesters so that it can correctly associate the cached Negotiated State with the appropriate Requester. Per the SPDM specification, the Negotiated State is between a given Requester and Responder pair, and remains valid until the next issuance of GET_VERSION or until the Responder decides to delete the associated Negotiated State. The mechanism to identify that a request originated from a different Requester is out of scope for the SPDM specification because it might require information from the transport layer. Any implementation of such a mechanism is transport specific but an example of a mechanism is that an MCTP-based implementation can track the Source Endpoint ID associated with a state identifier (using a mechanism that is out of scope for the SPDM specification) and invalidate the cached Negotiated State on any request that originates from a different Source Endpoint ID. Note that implementations should take care to reliably identify devices across resets, especially on buses that re-enumerate themselves and might allocate different identifiers to devices after each reset.

182 9.2.2.2.2 Negotiated State validity

183 Support for CACHE_CAP requires both the Requester and Responder to manage the validity of the Negotiated State. Requesters and Responders should only save a Negotiated State after a successful CHALLENGE exchange. Prior to a successful CHALLENGE exchange, a Negotiated State is subject to attack.

184 After a Negotiated State has been established, a Requester should take steps to detect a firmware update on the Responder. If the Requester detects a firmware update, the Requester should invalidate the current Negotiated State, issue the GET_VERSION request through CHALLENGE_AUTH request, and establish a new Negotiated State.

185 9.2.3 NEGOTIATE_ALGORITHMS and ALGORITHMS exchange

186 The `ALGORITHMS` exchange enables the Requester and Responder to agree on the cryptographic algorithms that the components use for subsequent exchanges. The Responder should select the strongest algorithms that the Requester provides. After the `ALGORITHMS` exchange is complete, the Requester and Responder have an agreed set of algorithms to use in subsequent message exchanges. Certain values in the response message depend on fields in the `CAPABILITIES` exchange.

187 The extended `ExtAsym` and `ExtHash` algorithm fields in the `ALGORITHMS` exchange enable expansion to additional algorithms to meet custom requirements. The Requester and Responder should prefer the `BaseAsymAlgo` and `BaseHashAlgo` fields if they can agree on them.

188 If the Responder has set `CERT_CAP=1` and/or `CHAL_CAP=1`, the Responder must select algorithms that correspond to a certificate chain that the Responder possesses. To ensure compatibility, the Requester should support a variety of algorithms.

189 9.2.4 GET_DIGESTS and DIGESTS exchange

190 The `DIGESTS` exchange enables the Requester to retrieve the digests (hashes) of the certificate chain(s) stored on the Responder. The Requester can use the `DIGESTS` exchange to determine whether the certificate chain(s) stored on the Responder have changed. The Requester should store at least the public key from the leaf certificates along with the digest(s). The Requester can use the `DIGESTS` exchange as a shortcut to skip the retrieval of individual certificate chains, as the retrieval process can be slow on slower interfaces.

191 The `DIGESTS` response is not signed, so it is susceptible to replay attacks. It should be followed with a `CHALLENGE` or `GET_MEASUREMENTS` command to ensure that the Responder knows the private key.

192 9.2.5 GET_CERTIFICATE and CERTIFICATE exchange

193 The `CERTIFICATE` exchange enables a Requester to retrieve one or more certificate chains from the Responder. The `CERTIFICATE` response is potentially very large so a Requester might use the `Offset` and `Length` fields in the `GET_CERTIFICATE` request to issue multiple requests.

194 9.2.6 CHALLENGE and CHALLENGE_AUTH exchange

195 The `CHALLENGE` exchange enables the Requester to ensure that the Responder knows the private key associated with a certificate chain. The `CHALLENGE` request and `CHALLENGE_AUTH` response contain several fields of note:

- Both the request and response messages contain `Nonce` fields, to protect against replay and chosen message attacks.
- The response contains a `CertChainHash` field, which the Requester can use to refute the `DIGESTS` or `CERTIFICATE` response.

- The response might contain a `MeasurementSummaryHash` field, which is a measurement of the concatenation of all elements of the TCB for the Responder.
- The `OpaqueDataLength` and `OpaqueData` fields are intended to be defined by a binding specification. The specific location of these fields ensures that they are included in the `CHALLENGE_AUTH` signature.
- The `Signature` field is generated according to the signature-generation process in the `CHALLENGE_AUTH` signature generation clause of the SPDM specification. The goal of the signature is to show that the Responder is the entity that has been responding to the Requester for earlier message exchanges, and that the Responder knows the private key associated with the public key in the leaf certificate of the certificate chain.

196 Although the use of `Nonce` fields in both the `CHALLENGE` request and the `CHALLENGE_AUTH` response messages protects against replay attacks, an adversary with physical access to the component can leverage the fact that a component responds to any correctly formed `CHALLENGE` with a signed response to perform side channel analysis, chip-clip attacks, or similar approaches to extract the component's private key.

197 Mitigations to such concerns should be applied at the implementation level, for example through steps such as those that the Key protection clause discusses. The SPDM protocol can require that request messages are authenticated, that is signed, as an additional protection for this class of threats. However, this requirement results in a significantly more complex protocol overall, increases message overhead unnecessarily in cases where Requester authentication is not supported, such as feature-limited Responders, and, ultimately, does not prevent adversaries who can produce `CHALLENGE` messages signed by a certificate chain trusted by the Responder from pursuing such avenues of attack.

198 9.2.6.1 Unique MeasurementSummaryHash

199 To prevent a potential length extension attack, a Responder should ensure that each `MeasurementBlock` used in a `MeasurementSummaryHash` is unique from any other `MeasurementBlock` in the given `MeasurementSummaryHash`. This applies to all uses of `MeasurementSummaryHash`. The exposure to a potential length extension attack is only in cases where the Requester does not issue `GET_MEASUREMENTS` and instead relies on the `MeasurementSummaryHash` alone to determine the state of the Responder.

200 9.2.7 GET_MEASUREMENTS and MEASUREMENTS exchange

201 The `MEASUREMENTS` exchange enables the Requester to query the measurements of the firmware, the software, or configuration of a Responder.

202 In the `GET_MEASUREMENTS` request, the signature is optional. In some cases, Responders might not be able to create signatures, but can still return measurements. A Requester might refuse to operate with a Responder that does not support signed measurements. When specified, the `MEASUREMENTS` response is signed, showing that the Responder originated all `MEASUREMENTS` responses and has knowledge of the private key that is associated with the public key in the leaf certificate of the specified certificate chain.

203 The `MEASUREMENTS` exchange is designed to work with measurements of static data, which is data that does not change except in response to a user action. The `MEASUREMENTS` exchange does not handle measurement of dynamic values that can change without user action, such as the speed of a fan.

204 **9.2.7.1 Summary measurements**

205 The `MEASUREMENTS` exchange does not support a mechanism to request a summary measurement option, meaning that there is not a mechanism to request that a Responder hash together all of its measurements and return a single hash of those measurements. A Requester might want to implement a summary measurement mechanism on its own to periodically check for changes in the underlying measurements, such as firmware configuration changes that happen outside of the purview of Requester. Another use case for a summary measurement mechanism is to monitor a component for firmware updates that happen outside of the purview of the Requester, though a firmware update and component reset also causes the component to return `ErrorCode` = `RequestResync`. Note, periodic polling for measurements and use of summary measurements are optional behaviors.

206 If a Requester requires a summary measurement capability, the Requester should assemble its own summary measurement from the `MEASUREMENTS` responses from a given Responder. The Requester can check the stored summary by issuing one or more `GET_MEASUREMENTS` requests, regenerating the summary measurement, and checking the new summary measurement against the previous summary measurement.

207 In addition, a Requester that only needs a summary of the Responder's measurements can retrieve the summary through the `MeasurementSummaryHash` field in the `CHALLENGE_AUTH` response. By doing so, a Requester can avoid sending the `MEASUREMENTS` request and complete authentication faster.

208 **9.2.7.2 Firmware debug indication**

209 The `MEASUREMENTS` response includes a mechanism to return a measurement of firmware configuration. If a component typically operates in a mode that restricts debug access, it is recommended that the component use at least one measurement to indicate whether debug restrictions are in place. In this case, the component should alter a firmware configuration measurement when it enters debug mode. This measurement should remain altered until the component is reset. If the user subsequently disables debug mode, the component should continue to report an altered firmware configuration measurement until reset to ensure that a Requester can detect a case where a debug capability has been enabled and disabled before the Requester can detect it. The measurement index and definition of any debug mode measurement is vendor specific.

210 **9.2.7.3 MEASUREMENTS only components**

211 Some components might only support the `MEASUREMENTS` capability, but not support the ability to sign the measurements. Such a component sets `CERT_CAP=0`, `CHAL_CAP=0`, and `MEAS_CAP=1` in the `CAPABILITIES` response message. This capabilities configuration is desirable in some cases, such as in a component with minimal processing capabilities. If a component like this exists, a Requester should carefully consider whether to trust the measurement that is returned by the Responder.

212 ## 9.2.8 Encapsulated request flows

213 In certain use cases, such as mutual authentication, the Responder needs the ability to issue its own SPDM request messages to the Requester. Certain transports prohibit the Responder from asynchronously sending out data on that

transport. Message encapsulation, which preserves the roles of Requester and Responder as far as the transport is concerned but enables the Responder to issue its own requests to the Requester, addresses cases like these.

214 The `GET_ENCAPSULATED_REQUEST` and `DELIVER_ENCAPSULATED_RESPONSE` request messages, (`ENCAPSULATED_REQUEST`) and `ENCAPSULATED_RESPONSE_ACK` response messages facilitate the encapsulated request flow.

215 The encapsulated requests flow is used in limited scenarios, such as mutual authentication, and cannot be used for general purpose SPDM message encapsulation. Only certain requests and their corresponding responses, including `ERROR` , can be encapsulated. For details, see DMTF DSP0274.

216 9.2.9 Secure session messages

217 A number of capabilities `Flags` are related to managing secure sessions, and many of the capabilities are used in conjunction with each other. The Secured Messages-related capabilities `Flags` are:

- `ENCRYPT_CAP`
- `MAC_CAP`
- `MUT_AUTH_CAP`
- `KEY_EX_CAP`
- `PSK_CAP`
- `ENCAP_CAP`
- `HBEAT_CAP`
- `KEY_UPD_CAP`
- `PUB_KEY_ID_CAP`

218 Many of the capabilities `Flags` have dependencies on each other, which are explained in the *SPDM Specification*. One dependency relationship of note is that the use of `ENCRYPT_CAP` requires the use of `MAC_CAP` . SPDM-Secured Messages that use encryption require the use of message authentication because the *SPDM Specification* does not support any use case for messages that are only encrypted because the use of such messages can result in a receiver decrypting messages from an attacker.

219 9.2.10 VENDOR_DEFINED_REQUEST and VENDOR_DEFINED_RESPONSE exchange

220 The `VENDOR_DEFINED_RESPONSE` exchange enables a Requester and Responder pair to exchange information that the SPDM specification does not otherwise cover. A component vendor or another standards body can define request and response messages. For more information on implementations by other standards bodies, see Partner implementations.

221 9.2.11 RESPOND_IF_READY sequence

222 The `RESPOND_IF_READY` sequence allows for situations when the Responder cannot respond in a reasonable time.

The time to a final response, which fulfills a `RESPOND_IF_READY` request, is still bound by the timing parameters that the SPDM specification defines.

223 The design intent of the `RESPOND_IF_READY` sequence is to enable components to cooperate with a larger system while performing long operations, such as signing. One reason to use `RESPOND_IF_READY` during a long operation is to release a shared bus to enable other components to use the bus during the operation.

224 9.3 Message exchanges

225 The SPDM specification specifies ordering rules for message exchanges and the transcript hash that is generated from those message exchanges. To reduce the complexity associated with message sequencing, the SPDM specification defines valid sequences including options for use cases that cache certain responses.

226 During the SPDM message exchanges, the Requester can drop communication with a Responder if the Responder violates a policy that the Requester holds, such as when the Responder negotiates too low of a version or the Responder returns too many errors.

227 The SPDM specification defines some messages as optional, such as `CHALLENGE`, which permits a variety of implementation permutations. Ultimately, the Requester implementation controls the policy that it wants to use and the SPDM specification grants the Requester some degree of implementation latitude. For instance, a security-sensitive Requester might reissue all requests on every reset while a more permissive Requester might cache certificate digests and skip the `CHALLENGE` on each reset. The Responder should make no assumptions about the security policy of the Requester.

228 9.3.1 Multiple Requesters

229 The tracking for message sequences is on the basis of a Requester and Responder pair, and a Responder can optionally support tracking more than one Requester and Responder pair. If a Responder receives requests from Requesters A and B, for instance, the Responder must track message payloads for the successful message exchanges with both Requester A and Requester B. A Responder has limited resources for tracking message exchanges, and might take steps to both limit the number of supported Requesters and reclaim resources that it has used to track exchanges with a given Requester. The exact mechanisms to do so are out of scope of the SPDM specification.

230 If a Responder supports communication with only a single Requester at a time, the Responder does not need to track the Requesters because communication with a new Requester starts with the `GET_VERSION` request and causes the Responder to discard any existing tracked messages. This type of implementation can cause problems in complex environments due to constantly restarting message sequences.

231 For implementations that use an MCTP transport, the `MCTP Endpoint ID` is the recommended method for tracking the Requester (see DMTF DSP0275). For other binding specifications, the binding specification should document the Requester tracking method.

232 ### 9.3.2 Message timeouts and retries

233 The ***Timing specification for SPDM Messages*** table in the SPDM specification lists a number of interrelated timeout values. The RTT value is the worst-case value for a message round trip based on the transport. The RTT value might be less than the CT value. If so, the Responder must respond with `ErrorCode` = `ResponseNotReady` within the RTT-specified time.

234 This mechanism ensures that Responders release the bus in a timely manner. After a Responder returns `ErrorCode` = `ResponseNotReady` , the Requester can issue a request to another Responder or wait for the time specified by `RDTExponent` and issue `RESPOND_IF_READY` . During this time, the Requester should not issue any request to the Responder other than `RESPOND_IF_READY` .

235 The SPDM specification allows for retries of messages after a timeout has occurred. In a retry scenario, a Requester retries the same request as before. Specifically, a retry of a `CHALLENGE` or `GET_MEASUREMENTS` request reuses the same nonce as the request that timed out so that the transcript hash calculation is not disrupted. A paranoid Requester can choose not to retry a request and instead return to `GET_VERSION` and restart the message sequence.

236 Certain SPDM protocol interactions involve the exchange of multiple messages, during which state information is maintained. For example, multiple `GET_MEASUREMENTS` messages might be issued in a sequence requesting individual unsigned measurements, with the Responder maintaining a message transcript to be signed at the end of the sequence. While individual requests and responses might be issued within the permitted timeout parameters, a malicious or buggy Requester might consume resources at the Responder by starting but never completing such multi-message interactions. This issue might be accentuated if a Responder interacts with more than one Requester in parallel, maintaining a number of active states. It is advised that SPDM implementations implement protections against such resource exhaustion scenarios by maintaining session limits, timeouts or similar mechanisms to detect and reset a misbehaving session when necessary. In this context, a session denotes an ongoing exchange of SPDM messages between a Requester and Responder pair.

237 #### 9.3.2.1 Secured Messages retries

238 The *Secured Messages using SPDM Specification 1.0.0 (DSP0277)* indicates that it is permissible for a component to include the sequence number in a message to help the receiver process a retry or out of order delivery if the transport protocol does not provide a mechanism to reconstruct the proper message order. SPDM Secured Messages are based on *IETF TLS DTLS13-43*, which indicates that including the sequence number is not considered a potential attack vector because section 3 of *IETF TLS DTLS13-43* adds the sequence number to the datagram record.

239 10 Attestation and security policies

240 This clause provides guidelines on:

- Attestation policies that can be implemented using the *SPDM Specification*.
- Security policies that can accompany such an implementation.

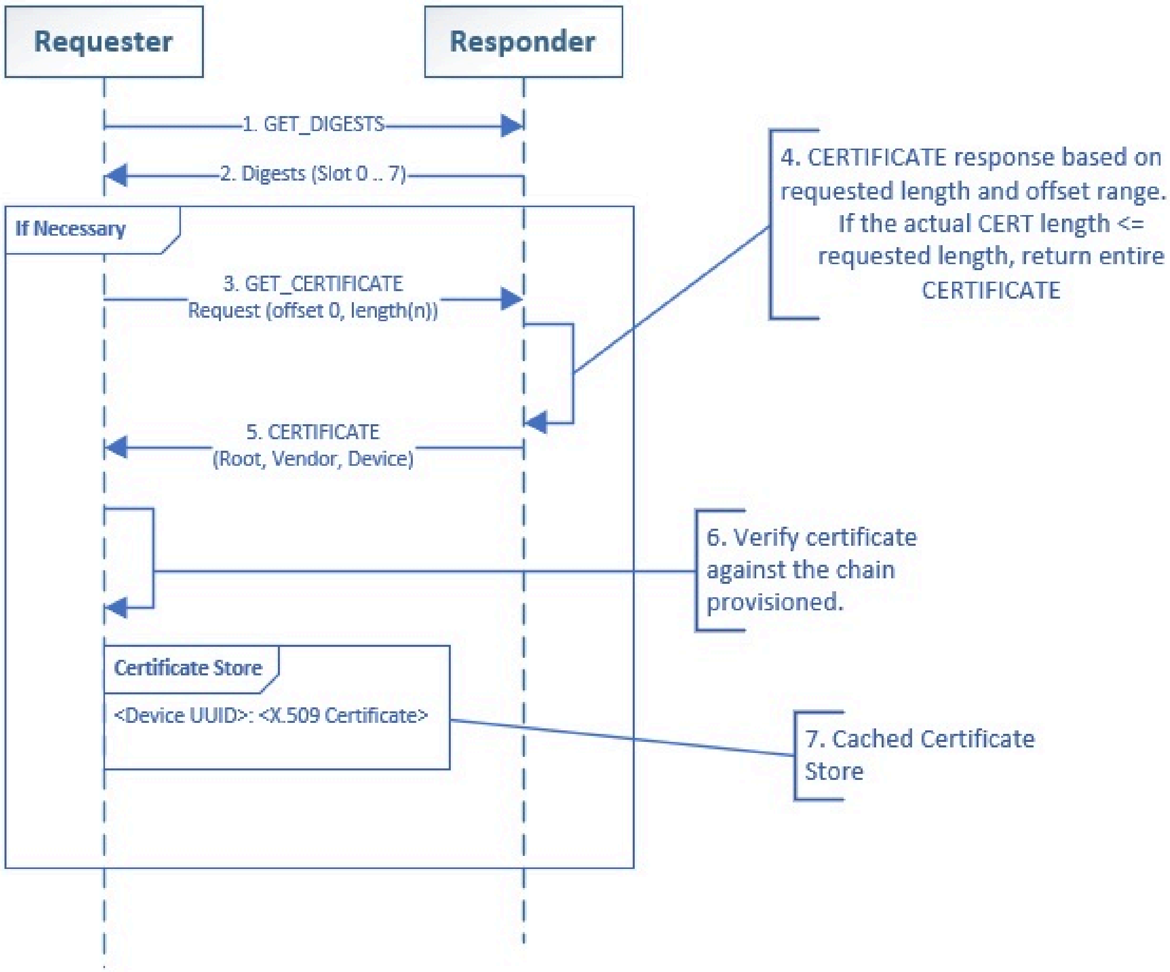
241 This clause is not exhaustive and should be considered informative. The details of any policy are vendor defined.

242 10.1 Certificate authorization policy

243 Trusting the device certificate and its security policy is confined to the authentication initiator's security policies. The SPDM authentication process involves retrieving the device certificate digests first and comparing them with the cached digests, or the trust store database. If not found in the cached trust store database, the Requester sends the `GET_CERTIFICATE` request. The responder returns the certificate based on the requested length and offset, as Figure 5 — Example certificate authentication policy shows. It is recommended that the Requester perform certificate verification procedures before storing the corresponding digest to the trust store.

244 **Figure 6 — Example certificate authentication policy**

245



246 The following initiator security policies can verify device certificates:

1. Generate warnings for components that do not support the SPDM.
2. Generate warnings for components that have certificate chains where root CA is not in the initiator's trust store database.
3. Quarantine components that have certificate chains where the root CA certificate is not in the trust store database.

247 10.2 Measurement

248 In addition to providing the hardware identity through a certificate, an authenticated endpoint can also be queried to provide the firmware identity. The firmware identity in this case refers to firmware code and configuration data. The value provided by the endpoint is a *measurement*. Using the `GET_MEASUREMENTS` command, the Requester can use

a single command to ask for an individual or all measurements. The returned values can be in form of a hash value or a bit stream and the Requester can specify whether the measurements must be signed to verify that the measurements originated with the Responder endpoint.

249 The Requester can, in turn, compare the returned measurements to known values. The Requester can either verify the measurements locally or remotely. The mechanism to obtain reference measurement values is outside of scope for the *SPDM Specification*.

250 10.3 Secured Messages policy

251 The addition of Secured Messages enables Requesters and Responder to apply policies surrounding their use. For example, a Responder might not accept certain vendor defined messages that it deems to be potentially destructive unless it receives those commands in a Secured Message. Another example is that a Requester might not support communication with a Responder of a certain component class unless the component supports authenticated encrypted Secured Messages.

252 11 Secured Messages

253 Version 1.1.1 of the SPDM specification enables the use of Secured Messages.

254 11.1 Secured Message layering

255 This section discusses the layering of secure messages. The examples in the section are presented to illustrate the concepts used in secure message layering, but are not intended to prescribe an implementation.

256 11.1.1 Secured Message send

257 Figure 7 — Secured Message send shows how layers are assembled when sending a Secured Message. The following describes the steps in the message assembly, moving from the top of the diagram to the bottom.

1. The component builds the message to be sent. This message can be any MCTP message type, as DMTF DSP0239 defines.
2. The component adds an MCTP header, setting the MCTP type and Integrity Check (IC) for the message. The result is the message to be encapsulated, which is the Application Data.
3. The component adds the `Application Data Length` and `Random Data` fields, as DMTF DSP0277 defines.
4. The component adds the Associated Data to the message, which is comprised of `Transport Version`, `Length` and `Session ID` as DMTF DSP0277 defines.
5. The component encrypts the message contents that were built over the previous steps, resulting in the ciphertext of the message. The component then generates a MAC over the message contents, including the ciphertext and the Associated Data. The encryption and MAC generation are typically handled by the AEAD algorithm.
6. The component appends the MAC to the message.
7. The component adds the MCTP header for the Secured Message, which is set to MCTP message type 6, as DMTF DSP0276 describes. This results in the Secured Message.
8. The component transmits the message as a sequence of one or more packets, as DMTF DSP0236 describes.

258 **Figure 7 — Secured Message send**

259

260 11.1.2 Secured Message receive

261 Figure 8 — Secured Message receive shows how layers are disassembled and authenticated when receiving a Secured Message. The following describes the steps in the message disassembly, moving from the top of the diagram to the bottom.

1. The component receives the message as a sequence of one or more packets, as DMTF DSP0236 describes, and reassembles the packets in an MCTP message.
2. The component reads the MCTP header to determine whether this is a Secured Message, which is indicated by MCTP message type 6 as DMTF DSP0276 describes. The MCTP header of the Secured Message is removed.
3. The component verifies the MAC by computing a MAC of the message and comparing it to the MAC field from the message.
4. The component removes the MAC field from the message.
5. The component uses the Associated Data from the message to decrypt the ciphertext. The decryption and MAC verification are typically handled by the AEAD algorithm.
6. The component removes the Associated Data from the message, leaving the plaintext of the Secured Message, as DMTF DSP0277 describes.
7. The component removes the `Application Data Length` and `Random Data` fields, as DMTF DSP0277 defines. The result is the encapsulated message.
8. The component processes the message.

262 **Figure 8 — Secured Message receive**

263

264 11.2 Secured Message error handling

265 If an error occurs during the Session Handshake phase or if an error happens during a secure session, the `Negotiated State` is preserved. The `Negotiated State` is preserved through errors unless the Requester sends an `END_SESSION` request with `Negotiated State Preservation Indicator=1` to terminate the session or sends a `GET_VERSION` request to reset the session.

266 If a timeout occurs during a secure session, the Requester can retry the message that failed. The retry is sent without modification. In this case, the Requester technically has more than one message outstanding to the same Responder but this is allowed because the second message is only a retry of the first message. Optionally, the Requester can send a `GET_VERSION` request to reset all sessions.

267 11.3 Random data

268 DSP0277 specifies that a component should set the `Random Data` field to a random length and fill it with random data. A component is allowed to set the length of `Random Data` to 0, or to fill the `Random Data` field with fixed values. However, there are benefits to using `Random Data` as DSP0277 suggests, including:

- Setting the length of `Random Data` to a random value can obfuscate the data being transmitted. An observer might gather information about the communication by observing the length of messages between two components and including data of random length hides the transmission from such an observation.

- Setting the contents of `Random Data` to random values ensures that all inputs to the encryption algorithm are unique. In the case of repeated encapsulated messages, the inclusion of random data values ensures that input plaintext to each encryption operation is unique.

269 12 Protection of internal secrets

270 This section describes recommended practices to protect a component's internal secrets.

271 13 Root of Trust

272 A Root of Trust provides the basis for trust in one or more security related function. All Root of Trust functions that the following clauses list can be implemented in one or multiple entities. An implementer should consider the following roots of trust when implementing an SPDM solution.

273 13.1 Root of Trust for detection

274 The foundation of component trust relies on the internal security of the component. During the component-boot process, the component performs a signature verification of each firmware stage to ensure that the firmware is authentic and no unauthenticated code has been injected into the firmware image. Examples of how to accomplish this task include using a static Root of Trust for detection that can authenticate subsequent stages of the boot process. If the signature verification fails during the boot process, the component can halt, boot to a recovery partition, or follow another recovery path for the platform that also conforms to the security policy. For more details on a Root of Trust for firmware authentication, see NIST SP800-193. As NIST SP800-193 indicates, the

> 275 ...central tenet to the firmware protection guidelines is ensuring that only authentic and authorized firmware update images may be applied to platform components.

276 13.2 Root of Trust for measurement

277 The SPDM implementation relies on the integrity of reported measurements. The Root of Trust for measurement is responsible for measurement of the elements, such as firmware images, that the `MEASUREMENTS` response reports, and for storing these measurements in a secure fashion.

278 13.3 Root of Trust for reporting

279 A Root of Trust for reporting ensures that values reported in SPDM responses accurately reflect the reported underlying state or condition. The Root of Trust for reporting ensures that other software in the system or an unauthorized user does not alter reported values.

280 14 Partner implementations

281 DMTF partners with other standards bodies to enable those bodies to use SPDM on other interfaces and protocols.

282 14.1 Partner binding specifications

283 DMTF enables partner standards bodies to create SPDM bindings for their specifications. Other binding specifications should provide the following guidance:

- Alterations to the `Subject Alternative Name` and `Common Name` fields in the certificate.
- Guidance on the vendor identification in the certificate.
- Bus timing and timeout requirements, including RTT.
- Use of `OpaqueData` fields in `CHALLENGE_AUTH` and `MEASUREMENTS` responses.
- Method to track messages from multiple Requesters, as Multiple caching Requesters describes.

284 14.2 Enabling partner implementations

285 The SPDM specification has several mechanisms to enable partner implementations.

286 14.2.1 OpaqueData

287 Many messages include fields for `OpaqueData` and `OpaqueDataLength`. These fields are for partner standards bodies to use to meet their requirements to include additional data in the SPDM messages. By including these fields in the messages, the contents of the fields are also covered by message transcripts and signatures.

288 If a standards body requires the use of the `OpaqueData` fields, then the standards body in question is responsible for documenting the proper use of the `OpaqueData` fields.

289 14.2.2 Registry or standards body ID

290 Several message exchanges include a field for `Registry ID` or `StandardID`, which allows the use of enumerations and field definitions that are defined by partner standards bodies. If a standards body requires an additional Registry or standards body definition, the standards body should work with DMTF to define a new `Registry or standards body ID` in the SPDM specification.

291 14.2.3 Vendor-defined commands

292 The *SPDM Specification* has an allowance for vendor defined commands, using the `VENDOR_DEFINED_REQUEST` and

`VENDOR_DEFINED_RESPONSE` messages. These messages include fields to provide a vendor ID for the vendor that defined the command, and to accommodate vendor IDs that are defined by multiple standards bodies.

293 In addition to the use of vendor-defined commands by component vendors, a standards body itself can define vendor-defined commands, in which case the standards body assigns itself a vendor ID of the type of its vendor ID.

294 If a standards body is not listed in the ***Registry or standards body ID*** table in the SPDM specification and there is a requirement to add a command using an ID from that standards body, then the standards body should work with DMTF to allocate an ID to the table to avoid potential conflicts.

295 14.2.4 Certificates with partner information

296 The SPDM specification defines information that is stored in a certificate, and all such information is identified using a unique OID. Partner standards bodies and component vendors can also define information to be stored in a certificate.

297 The following certificate gives an example of such a certificate. This certificate contains SPDM specification defined information in the `Subject Alternative Name` `otherName` identified by the OID 1.3.6.1.4.1.412.274.1. The partner organization information is found in a second `Subject Alternative Name` `otherName` field, and identified by the OID 1.2.3.4.5.6.7.1. Requesters that process certificates can read the OID for each `Subject Alternative Name` `otherName` to help Requester correctly interpret the associated data.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4098 (0x1002)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = US, ST = NC, L = City, O = ACME, OU = ACME Devices, CN = CA
        Validity
            Not Before: Jan  1 00:00:00 1970 GMT
            Not After : Dec 31 11:59:59 9999 GMT
        Subject: C = US, ST = NC, L = City, O = ACME Widget Manufacturing, OU = ACME Widget Manufacturing Unit, CN
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c7:d6:81:6b:16:fa:c9:a9:de:60:8a:3b:3e:c6:
                    11:a2:fd:48:d2:e9:e8:d2:f5:d4:10:08:06:ad:ee:
                    14:76:b7:41:15:88:c9:c1:d0:5a:58:08:b7:f0:04:
                    bb:85:31:43:2f:3a:c9:53:67:99:9e:fc:b6:af:70:
                    bb:1d:ef:b1:6d:69:fb:38:57:c7:71:da:fe:2b:fd:
                    bf:18:81:15:c6:e1:cb:1c:65:54:5f:de:04:f7:f6:
                    a1:f9:b3:8b:40:12:69:05:23:7c:15:41:27:ac:65:
                    6c:d9:66:f4:eb:3c:b8:4f:f6:5a:4d:7a:26:ad:2f:
                    66:2b:cd:28:7c:d6:a6:ae:71:70:c8:0e:a8:3e:a3:
                    a1:96:d4:65:41:e2:01:a8:34:15:ef:50:ce:99:3f:
                    1d:38:ba:5c:53:37:d2:f3:46:94:08:ee:22:87:e2:
```

```
                90:7b:25:cf:6e:b0:cd:05:f1:e3:b7:5a:ee:f7:4f:
                9d:70:74:81:86:8d:5e:14:af:37:24:d0:39:71:3c:
                05:c2:a5:1c:a3:a1:5e:6b:f7:9e:5d:cf:c2:67:b9:
                a3:f2:e6:62:c9:96:97:e3:5e:83:c6:14:dd:4c:8b:
                53:87:7e:43:a2:81:28:4d:41:d1:48:b2:c9:c8:b2:
                53:ff:ce:82:d8:f9:ed:48:5a:87:fd:85:19:dc:ea:
                07:e5
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Basic Constraints:
            CA:FALSE
        X509v3 Authority Key Identifier:
            CB:0C:55:D9:4F:18:EE:B9:54:25:3D:08:1A:4C:02:24:80:BF:CF:FE
        X509v3 Key Usage: critical
            Digital Signature
        X509v3 Subject Alternative Name:
            othername: 1.3.6.1.4.1.412.274.1::ACME:WIDGET:0123456789, othername: 1.2.3.4.5.6.7.1::Vendor=ACME:D
Signature Algorithm: ecdsa-with-SHA256
Signature Value:
    30:46:02:21:00:f1:a5:9b:1f:6e:ac:9d:11:24:d5:da:6f:2c:
    ea:c1:93:e8:0c:58:38:c9:66:38:5c:96:20:75:a7:77:5d:20:
    c5:02:21:00:88:30:e4:f0:2e:82:e4:45:93:84:e5:23:58:2d:
    90:c3:32:51:6f:a0:35:c8:7f:a4:6b:21:01:0a:13:db:26:92
```

298 # 15 ANNEX A (informative) change log

299 ## 15.1 Version 1.0.0 (2020-05-13)

- Initial Release

300 ## 15.2 Version 1.1.0 (2022-01-04)

- Update content and diagrams to match *Security Protocol and Data Model (SPDM) Specification 1.1.1 (DSP0274)*
- Restructure several sections to improve readability, including:
  - Certificates.
  - Partner implementations.
- Update references to latest versions.
- Removed statement about possible re-provisioning of the certificate chain in slot 0.
- New:
  - Add Authenticated Encryption with Associated Data (AEAD) to the introduction.
  - Add discussion of new Message details and CAPABILITIES Flags.
  - Add discussion of Pre-Shared Key.
  - Add discussion of details for use of the CACHE_CAP flag.
  - Add discussion of complexities around Certificate chain algorithms and implementation considerations.
  - Discuss validation of certificate chains in Certificate requirements.
  - Clarify use of `MeasurementSummaryHash` versus Summary measurements.
  - Clarify that SPDM code is in the SPDM Trusted Computing Base.
  - Add Figure 3 — SPDM security stack.
  - Add Secured Message layering example.
  - Add an example of Certificates with partner information.
  - Add discussion of Secured Messages and Secured Messages policy.
  - Add a section for Alternatives to certificate chains.
  - Add discussion of Vendor defined commands.

301 # 16 Bibliography

302 DMTF DSP4014, *DMTF Process for Working Bodies 2.6*.