Document Identifier:

DSP2058 Date: 2020-09-04

Version: 1.0.0

Security Protocol and Data Model (SPDM) Architecture White Paper

Supersedes: None

Document Class: Informative

Document Status: Published

Document Language: en-US

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

20 CONTENTS

21 1 Foreword

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

23 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.

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

25 1.1 Acknowledgments

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

27 **Editors:**

- Brett Henning — Broadcom Inc.
- Masoud Manoo — Lenovo
- Viswanath Ponnuru — Dell Technologies

28 **Contributors:**

- Richelle Ahlvers — Broadcom Inc.
- Lee Ballard — Dell Technologies
- Patrick Caporale — Lenovo
- Yu-Yuan Chen — Intel Corporation
- Nigel Edwards — Hewlett Packard Enterprise
- Daniil Egranov — Arm Limited
- Philip Hawkes — Qualcomm Inc.
- Jeff Hilland — Hewlett Packard Enterprise
- Yuval Itkin — Mellanox Technologies
- Theo Koulouris — Hewlett Packard Enterprise
- Luis Luciani — Hewlett Packard Enterprise
- Donald Matthews — Advanced Micro Devices, Inc.
- Mahesh Natu — Intel Corporation
- 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
- Hemal Shah — Broadcom Inc.
- Srikanth Varadarajan — Intel Corporation

29
2 Abstract

30 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.

31
> **Note:** This white paper refers to this architecture as the Security Protocol and Data Model (SPDM) architecture, or SPDM.

32 The PMCI architecture focuses on intercommunications 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.

33 Servers, desktop systems, mobile systems, thin clients, bladed systems, and other types of devices might contain a platform management subsystem. 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.

34

3 References

35 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 DSP0274, *Security Protocol and Data Model (SPDM) Specification 1.0.0*
- DMTF DSP0275, *Security Protocol and Data Model (SPDM) over MCTP Binding Specification 1.0.0*
- DMTF DSP2015, *Platform Management Component Intercommunication (PMCI) Architecture White Paper 2.0.0*
- 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*
- RFC5280, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*
- *USB Authentication Specification Rev 1.0 with ECN and Errata through January 7, 2019*

36 4 Terms and definitions

37 This white paper uses the terms that *Security Protocol and Data Model (SPDM) Specification 1.0.0* and *Security Protocol and Data Model (SPDM) over MCTP Binding Specification 1.0.0* describe.

38

5 Introduction

39

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.

40

5.2 Authentication

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

42 An SPDM-compliant device generates, or is provisioned with, an asymmetric *device key pair*. The device 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 device-generated signature. For more details about the message exchanges, see SPDM messages.

43

5.3 Security Platform and Data Model (SPDM) architecture

44 A platform management subsystem in a modern enterprise computer platform is comprised of a set of components, which communicate to perform management functions within the platform. In many cases, these communications occur between components that are comprised of 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 are comprised of fixed logic or fixed firmware or software.

45 In such a platform management subsystem, stakeholders have a desire to establish trust, and to reestablish trust over time, with a component before you can securely communicate with that component.

46 The DMTF SPDM provides an *authentication* mechanism to establish trust, which uses proven cryptographic methods that protect the authentication process.

47 For the purposes of this white paper, a component can encompass a number of device types, including PCIe adapters, Baseboard Management Controllers, purpose built authentication devices, 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 devices, the compromise of firmware, or other attacks.

48 The SPDM enables these mechanisms to authenticate a component:

1. 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.
2. A certificate authority (CA) provisions and manages the retrieval of a device certificate, which provides a protocol to `CHALLENGE` a component to prove that it is the device that provided the certificate.

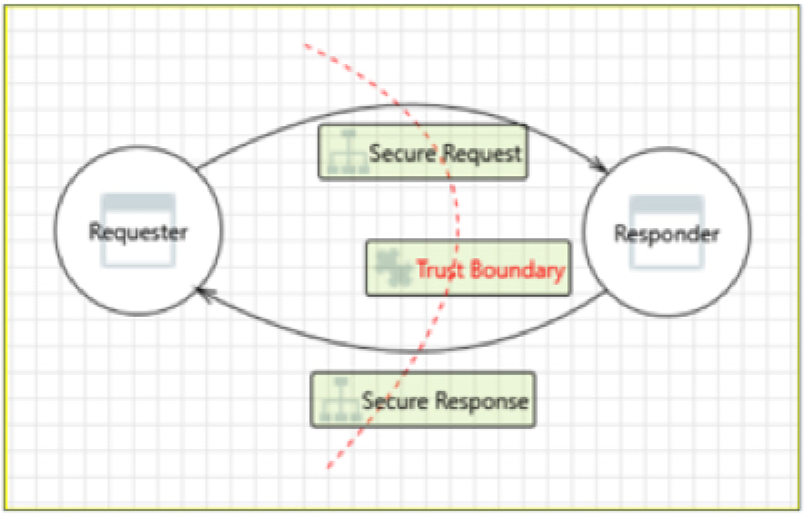
49 Finally, SPDM includes provisions for future expansion, by adding operations and capabilities while maintaining compatibility with existing deployments.

50 5.4 Threat model

51 The risk assessment identifies threats and vulnerabilities related to the SPDM interactions between device endpoints. Figure 1 — SPDM threat model shows the SPDM interaction between device endpoints.

52 **Figure 1 — SPDM threat model**

53



> 54 **Scope of this risk assessment:**
>
> 55 The scope of this assessment includes security controls of device comprises data model security, authentication and authorization. Any limitations of the physical I2C, I3C, PCIe, GenZ, or CXL network channel shall not apply to this threat assessment.

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

57

Table 1 — Threat modeling assessment and mitigations

Item number	Description	STRIDE category	Justification mitigation
1	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.	Tampering	To prevent replay attacks, the Requester and Responder shall use the nonce of a random number.
2	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.	Tampering	To prevent intruders from tampering with exchanged data, use one or more of these strategies: • Strong authorization schemes • Hashes • Message authentication codes • Digital signatures
3	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.	Information Disclosure	To prevent attacks, use one or more of these strategies as supported by the endpoint devices: • Stronger authentication schemes • Versions • Cryptographic algorithms
4	Requester or Responder might be able to impersonate the context of the Requester or Responder to gain additional privilege.	Elevation of 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.
5	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.	Repudiation	To mitigate attacks, use one or more of these strategies: • Digital signatures • Timestamps • Audit trails

Item number	Description	STRIDE category	Justification mitigation
6	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.	Information Disclosure	To mitigate this attack, use stronger authentication schemes and cryptographic algorithms.
7	Requester or Responder crashes, halts, stops, or runs slowly. In all cases, an availability metric is violated.	Denial of Service	Out of Scope. To address uncorrectable errors or any type of crash, the Requester or Responder shall implement recovery mechanisms.

Item number	Description	STRIDE category	Justification mitigation
8	External agent interrupts data flowing across a trust boundary in either direction.	Denial of Service	67 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: 68 a. Receipt of the end packet for a message. 69 b. Receipt of a new start packet. 70 c. Timeout waiting for a packet. 71 d. Out-of-sequence packet sequence number. 72 e. Incorrect transmission unit. 73 f. Bad message integrity check. 74 Only the whole MCTP message is secure. The individual MCTP packets are not secure.
9	Requester or Responder might be able to remotely execute code for the Responder.	Elevation of 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.
10	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.	Elevation of 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.

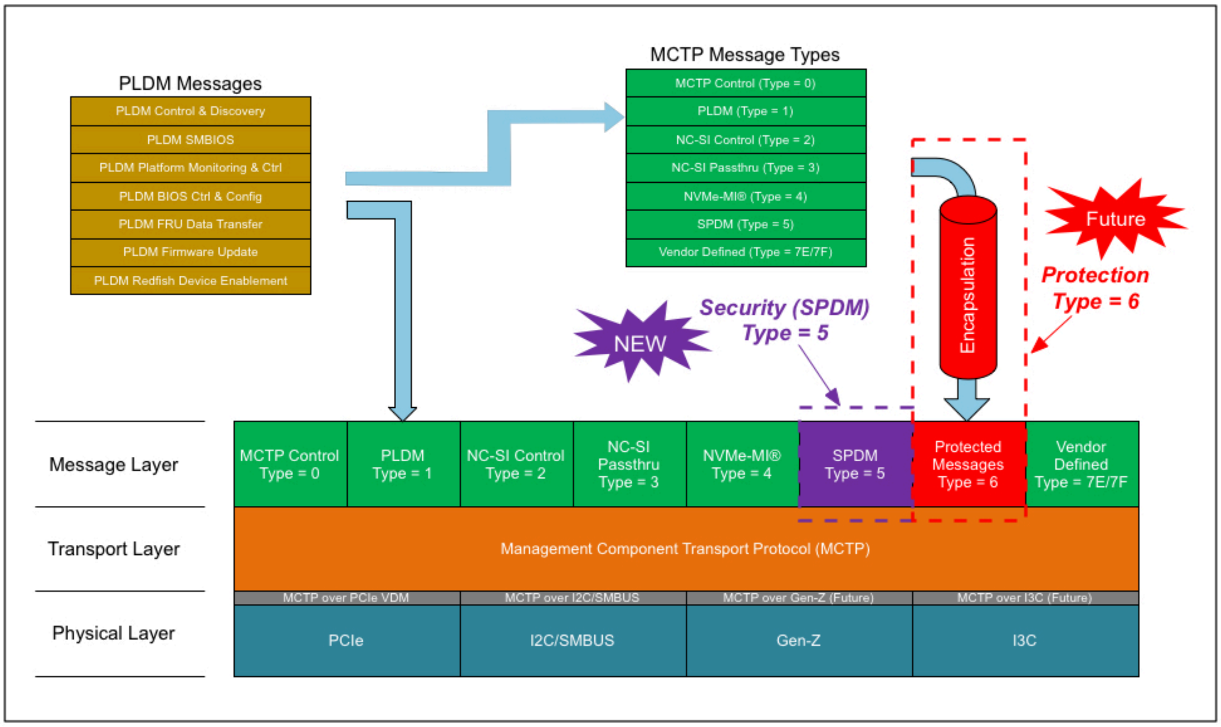
75 6 SPDM concepts

76 6.1 PMCI stack

77 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 reserved for future versions of the *SPDM Specification*.

78 **Figure 2 — SPDM over MCTP**

79



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

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

82 For details on the relationships among PMCI specifications, see the *Platform Management Component Intercommunications (PMCI) Architecture White Paper* (DSP2015).

83 6.2 Other bindings

84 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. For links to other binding specifications, see Other binding specifications.

85

7 Certificates

86 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 `GET_MEASUREMENTS`, it does not require a certificate chain or need to comply with the rest of this clause.

87 SPDM supports the concept of certificate slots, where each certificate slot can contain a different certificate chain. SPDM 1.0 supports up to eight certificate slots, though only slot 0 is required.

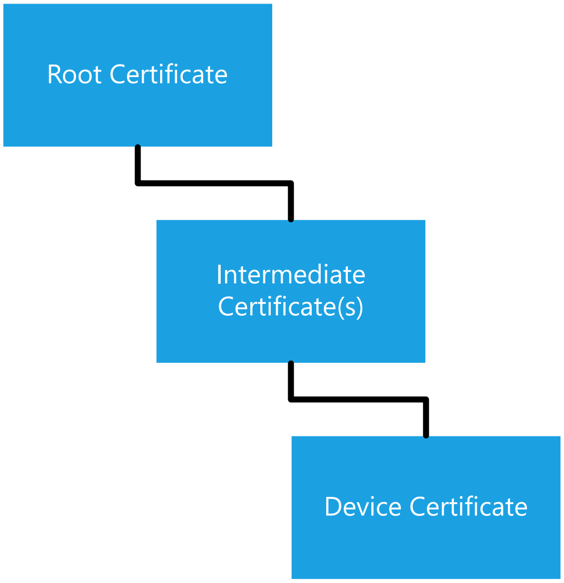
88 The certificate chain in slot 0 has a special role in the system because it is provisioned during manufacturing. The certificate chain in slot 0 represents the manufacturer, and this certificate chain is immutable and cannot be changed. This certificate chain is also known as the *manufacturer's certificate chain*.

89 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 that the Requester trusts. The certificate that the Requester trusts could be a root certificate or an intermediate certificate.

90 Figure 3 — Example certificate chain shows an example certificate chain:

91 **Figure 3 — Example certificate chain**

92



93 Table 2 — Certificate chain elements summarizes the roles of the elements that Figure 3 — Example certificate chain shows.

94 **Table 2 — Certificate chain elements**

Certificate chain element	Description
Root certificate	First, or highest, certificate in the chain. Contains a record of the issuing authority and is self-signed.
Intermediate certificate	Certificate chain typically contains one or more of these certificates, which protect the root certificate because the root certificate might be kept offline. Also, enables a more complex hierarchy of certificates; for instance, enables the allocation of separate intermediate certificates to different product divisions within a company.
Device certificate	Identifies the component. Should not change over the life of a component, unless the component is re-provisioned.

95 Each certificate in the chain can be verified by checking the signature in the certificate, because the private key of the certificate above it in the chain signs each certificate, working its way back to the root certificate. If each certificate in the chain verifies correctly, the individual certificates in the chain may be trusted. By performing this verification process, the Requester can determine whether the device certificate has been tampered with, and by extension, whether the Responder is the expected individual component. Additional details related to certificate chain validation are found in USB Authentication.

96 Slot 0 is typically immutable but can be cleared with a re-provisioning command (out of scope), after which it must be loaded with a new certificate chain.

97 7.1 Certificate requirements

98 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.

99 The leaf certificate in the certificate chains must conform to the *SPDM Specification*, clause 8.2.3 format. The certificate format guidance in SPDM is based on RFC5280. Table 3 — Optional leaf certificate attributes describes the leaf certificate attributes that the *SPDM Specification* specifies as optional.

100 **Table 3 — 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. As 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. As 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.

Attribute	Description
`Subject Alternative Name`	Recommended. It enables reporting of more detailed and standardized component identification.

101 The *SPDM Specification* details the `Subject Alternative Name` for components that use a DMTF-specified transport mechanism. Bodies that create additional binding specifications for SPDM should specify appropriate guidelines for the `Subject Alternative Name` and Common Name fields. All bodies that use the *SPDM Specification* should retain the `Serial Number` field in the certificate definition.

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

103 The OID in the `othername` field is `1.3.6.1.4.1.412.274.1.0`. This value represents a UTF8String in the `<manufacturer>:<product>:<serialNumber>` format.

104 The X.509v3 certificates can include the `Authority Key Identifier`, which assists authentication of the certificate chain, which 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.

105 The following example string shows the format:

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

106 7.2 Example leaf certificate

107 The following example shows a leaf certificate:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 8 (0x8)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = CA, 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 23:59:59 9999 GMT
        Subject: C = US, ST = NC, O = ACME Widget Manufacturing, OU = ACME Widget Manufacturing Unit, CN = w0123456789
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:ba:67:47:72:78:da:28:81:d9:81:9b:db:88:03:
                    e1:10:a4:91:b8:48:ed:6b:70:3c:ec:a2:68:a9:3b:
                    5f:78:fc:ae:4a:d1:1c:63:76:54:a8:40:31:26:7f:
                    ff:3e:e0:bf:95:5c:4a:b4:6f:11:56:ca:c8:11:53:
                    23:e1:1d:a2:7a:a5:f0:22:d8:b2:fb:43:da:dd:bd:
                    52:6b:e6:a5:3f:0f:3b:60:b8:74:db:56:08:d9:ee:
                    a0:30:4a:03:21:1e:ee:60:ad:e4:00:7a:6e:6b:32:
                    1c:28:7e:9c:e8:c3:54:db:63:fd:1f:d1:46:20:9e:
                    ef:80:88:00:5f:25:db:cf:43:46:c6:1f:50:19:7f:
                    98:23:84:38:88:47:5d:51:8e:11:62:6f:0f:28:77:
                    a7:20:0e:f3:74:27:82:70:a7:96:5b:1b:bb:10:e7:
                    95:62:f5:37:4b:ba:20:4e:3c:c9:18:b2:cd:4b:58:
                    70:ab:a2:bc:f6:2f:ed:2f:48:92:be:5a:cc:5c:5e:
                    a8:ea:9d:60:e8:f8:85:7d:c0:0d:2f:6a:08:74:d1:
                    2f:e8:5e:3d:b7:35:a6:1d:d2:a6:04:99:d3:90:43:
                    66:35:e1:74:10:a8:97:3b:49:05:51:61:07:c6:08:
                    01:1c:dc:a8:5f:9e:30:97:a8:18:6c:f9:b1:2c:56:
                    e8:67
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Subject Alternative Name:
                othername:1.3.6.1.4.1.412.274.1;UTF8STRING:ACME:WIDGET:0123456789
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:45:02:21:00:fc:8f:b0:ad:6f:2d:c3:2a:7e:92:6d:29:1d:
        c7:fc:0d:48:b0:c6:39:5e:c8:76:d6:40:9a:12:46:c3:39:0e:
        36:02:20:1a:ea:3a:59:ca:1e:bc:6d:6e:61:79:af:a2:05:7c:
        7d:da:41:a9:45:6d:cb:04:49:43:e6:0b:a8:8d:cd:da:e
```

108 7.3 Certificate provisioning

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

110 Some methods to create a certificate chain include:

- Generate a certificate signing request (CSR) by using a component's firmware.
- 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.

111 Any approach for generating a certificate chain should occur in a secure facility. 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.

112 7.4 Certificate slots

113 In a component, each certificate chain occupies a slot and the certificate chain that is stored in slot 0 is provisioned by the component manufacturer.

114 Some use cases use certificate slots 1 to 7. For instance, an administrator can claim ownership of a component by installing a certificate chain belonging to the component owner in one or more of the additional slots (certificate slots 1 to 7). The use of these additional slots would allow the administrator to authenticate the component using a certificate chain that is owned and managed by the administrator.

115 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` .

116 The mechanisms to provision a certificate chain to any slot or to manage multiple sets of certificate chain slots are beyond the scope of this white paper and are not part of the SPDM.

117 7.5 Device key pair

118 Each component must contain a public and private key pair, or the *device key pair*, that is statistically unique to that component.

119 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. The *SPDM Specification* details support for up to eight certificate chain storage slots.

120 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 from the available list in accordance with the manufacturer’s needs.

121 7.5.1 Key provisioning

122 There are two primary options for provisioning a device key pair to a component, though there are multiple

mechanisms available to accomplish each of the options. Any component that supports SPDM certificate or measurement-related command sets must provision device key pairs.

123 7.5.1.1 Internal key generation

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

125 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).
- Have sufficient processing power or hardware support to generate a key pair by using the chosen algorithm.
- Be able to protect some, or all, source data that the key generation process uses.
- Have a sufficiently strong source of entropy, if using an ECDSA algorithm.

126 7.5.1.2 External key provisioning

127 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. It is also possible to use the external tools and components to meet the portion of the requirements that the component cannot meet. 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.

128 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.

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

130 7.5.2 Key protection

131 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 foundation for the device key pair cannot be accessed 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.

132 Some items that the component should protect are:

- The basis of the device identity.
- The device private key.
- Any values that were used to derive or store the protected values.

133 When the device private key is in decrypted form, it should only be stored in a component's internal memory. 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 key pair but the non-volatile memory should be protected against unauthorized access, including attempts to directly access the non-volatile memory, such as removing a flash part.

134 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.

135 8 SPDM messages

136 8.1 Message details

137 8.1.1 GET_VERSION and VERSION exchange

138 The `VERSION` exchange creates agreement between the Requester and the Responder on the major SPDM version that they use for future messages. The `VERSION` exchange remain backward compatible in all future version of SPDM.

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

140 8.1.2 GET_CAPABILITIES and CAPABILITIES exchange

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

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

142 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.

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

144 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.

145 8.1.2.1 CAPABILITIES flags

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

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

148 **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 lost its cached values, it responds to the next request, other than `GET_VERSION`, with an `ERROR` of `UnexpectedResult`, which indicates to the Requester that it is required to restart from `GET_VERSION`.
`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`	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.
`MEAS_CAP`	Support for Measurements 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.
`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.

149 8.1.3 NEGOTIATE_ALGORITHMS and ALGORITHMS exchange

150 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.

151 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.

152 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.

153 8.1.4 GET_DIGESTS and DIGESTS exchange

154 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 if 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.

155 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.

156 8.1.5 GET_CERTIFICATE and CERTIFICATE exchange

157 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.

158 8.1.6 CHALLENGE and CHALLENGE_AUTH exchange

159 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, which are random numbers.
- 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 `OpaqueLength` 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.

160 8.1.7 GET_MEASUREMENTS and MEASUREMENTS exchange

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

162 In the `GET_CAPABILITIES` and `GET_MEASUREMENTS` requests, the signature is optional. In some cases, Responders might not be able to create signatures, but can still return measurements. A paranoid 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.

163 **8.1.7.1 Summary measurements**

164 The `MEASUREMENTS` exchange does not support a mechanism to request a summary measurement option, meaning that a Requester cannot request that a Responder hash all its measurements into a single hash of those measurements. A Requester might use such a mechanism to periodically check for changes in the underlying measurements, potentially triggering a policy response. If a Requester requires a summary measurement capability, the Requester should assemble its own summary measurement from the `MEASUREMENTS` responses from a Responder. The Requester can check the stored summary by issuing one or more new `GET_MEASUREMENTS` requests, regenerating the summary measurement and checking the new summary measurement against the previous summary measurement.

165 **8.1.7.2 Firmware debug indication**

166 The `MEASUREMENTS` response includes a mechanism to return a measurement of firmware configuration. If a device typically operates in a mode that restricts debug access, it is recommended that the device use at least one measurement to indicate whether debug restrictions are in place. In this case, the device should alter a firmware configuration measurement when it enters debug mode. This measurement should remain altered until the device is reset. If the user subsequently disables debug mode, the device should continue to report an altered firmware configuration measurement until reset to ensure that both:

167 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 may wish 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. A Requester can also use a summary measurement mechanism 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.

168 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 new `GET_MEASUREMENTS` requests, regenerating the summary measurement, and checking the new summary measurement against the previous summary measurement.

169
8.1.8 VENDOR_DEFINED_REQUEST and VENDOR_DEFINED_RESPONSE exchange

170 The `VENDOR_DEFINED_RESPONSE` exchange enables a Requester and Responder pair to exchange information that the *SPDM Specification* does not otherwise cover.

171
8.1.9 RESPOND_IF_READY sequence

172 The `RESPOND_IF_READY` sequence allows for situations when the Responder cannot respond it 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 defines.

173 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.

174 8.2 Message exchanges

175 The SPDM 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.

176 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.

177 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 wishes to use and the *SPDM Specification* grants the Requester some degree of implementation latitude. For instance, a paranoid 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.

178 8.2.1 Multiple Requesters

179 The tracking for message sequences are on a Requester and Responder pair, and a Responder can optionally support 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 outside of the scope of the *SPDM Specification*.

180 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.

181 For implementations that use an MCTP transport, the MCTP endpoint ID is the recommended method for tracking the Requester. For other binding specifications, the binding specification should document the method.

182 8.2.2 Message timeouts and retries

183 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.

184 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`.

185 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 to not retry a request and instead return to `GET_VERSION` and restart the message sequence.

186 9 Attestation and security policies

187 This clause provides guidelines on:

- Possible attestation policies that can be implemented by using SPDM 1.0.
- Security policies that can accompany such an implementation.

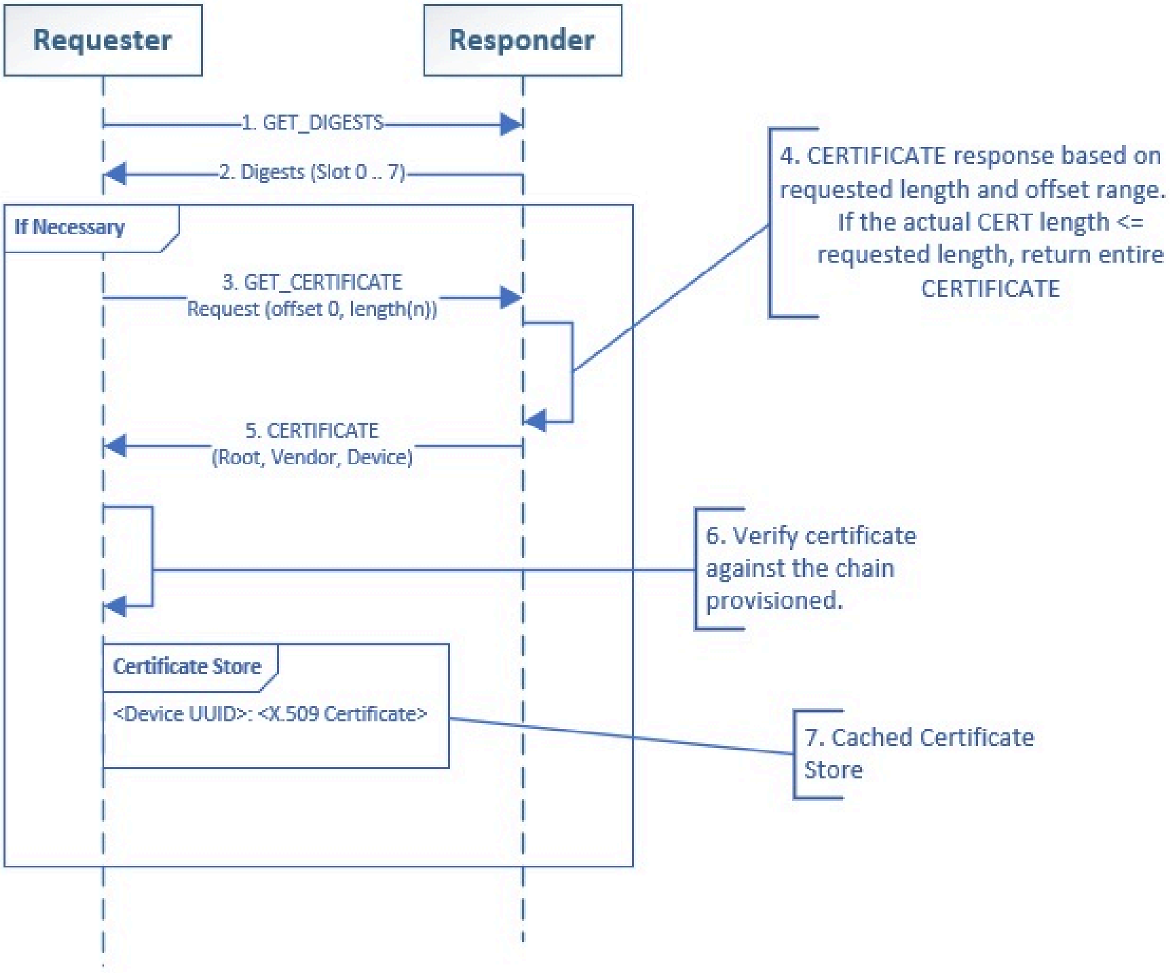
188 This clause is not exhaustive and should be considered informative.

189 9.1 Certificate authorization policy

190 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 4 — Example certificate authentication policy shows. It is recommended that the Requester perform certificate verification procedures before storing the corresponding digest to the trust store.

191 **Figure 3 — Example certificate authentication policy**

192


Requester
Responder
1. GET_DIGESTS
2. Digests (Slot 0 .. 7)
If Necessary
3. GET_CERTIFICATE
Request (offset 0, length(n))
4. CERTIFICATE response based on requested length and offset range. If the actual CERT length <= requested length, return entire CERTIFICATE
5. CERTIFICATE
(Root, Vendor, Device)
6. Verify certificate against the chain provisioned.
Certificate Store
<Device UUID>: <X.509 Certificate>
7. Cached Certificate Store

193 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.

194 9.2 Measurement

195 In addition to providing the hardware identity through a certificate, an authenticated endpoint could also be queried to provide the firmware identity. The firmware identity in this case is a term that is used to refer to firmware code and configuration data. The value provided by the endpoint is called a measurement. Using the `GET_MEASUREMENTS`

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

196 The Requester can, in turn, compare the returned measurements to a known value. The Requester can either compare the measurements locally or use a remote attestation server to validate the results.

197 9.3 Firmware provisioning

198 Care must be taken when the component firmware is being updated. As NIST SP800-193 indicates, the

> 199 ...central tenet to the firmware protection guidelines is ensuring that only authentic and authorized firmware update images may be applied to platform devices.

200 The update process should follow procedures to ensure that only authenticated firmware is installed.

201 9.4 Roots of trust

202 The foundation of component trust relies on the internal security of the component. During the component-boot process, measurement of each firmware stage must be done to ensure that the firmware is authenticate and no malicious code has been injected into the firmware image. Examples of how to accomplish this task include using a static root of trust that can measure subsequent stages of the boot process. If the signature verification fails during the boot process, the component must halt or boot to a recovery partition that also conforms to the measured signatures.

203 10 PMCI standards overview

204 The PMCI standards are composed of technologies defined in a suite of standard specifications.

205 10.1 SPDM

206 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.

207 The *SPDM Specifications* include:

- *Security Protocol and Data Model (SPDM) Specification 1.0.0*
- *Security Protocol and Data Model (SPDM) over MCTP Binding Specification 1.0.0*

208 DMTF partners with other standards bodies to enable those 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.

209

11 Change log

Version	Date	Description
1.0.0	2020-05-13	

210 # 12 Bibliography

211 DMTF DSP4004, *DMTF Release Process 2.4*, https://www.dmtf.org/sites/default/files/standards/documents/DSP4004_2.4.pdf.