TRUSTED® COMPUTING GROUP

SPECIFICATION

TCG DICE Concise Evidence Binding for SPDM

Version 1.0
Revision 0.54
January 17, 2024

Contact: admin@trustedcomputinggroup.org

PUBLISHED

DISCLAIMERS, NOTICES, AND LICENSE TERMS

ACKNOWLEDGEMENT

TCG wishes to thank all those who contributed to this specification. This document builds on work done in various working groups in TCG and the industry at large.

NAME	COMPANY
Henk Birkholz	Fraunhofer SIT
Yogesh Deshpande	ARM Holdings
Andrew Draper	Intel Corporation
Thomas Fossati	ARM Holdings
Brett Henning	Broadcom Inc.
Dennis Mattoon	Microsoft Corporation
Chandra Nelogal	Dell Technologies Inc.
Ned Smith	Intel Corporation

1 CONTENTS

1 SCOPE

This specification was developed in collaboration with the Secure Protocol and Data Model (SPDM) committee in the DMTF to define Evidence binding for SPDM that uses an SPDM Measurement Manifest and aligns with TCG defined Evidence and Endorsement schemas, see [1], [2], and [3]. This specification extends the Evidence and Endorsements schemas to describe both direct and indirect measurements. This specification accommodates indirect measurements by making them a hybrid of the SPDM Measurement Block and an SPDM Measurement Manifest containing a concise evidence schema.

This specification defines data structures for conveying Evidence using SPDM Measurement Manifests as defined in SPDM versions 1.1 or greater, see [4], and using certificates, see [1]. SPDM does not mandate use of SPDM Measurement Manifests. This specification does not restrict SPDM normative requirements.

Normally, Evidence is conveyed to a remote Verifier without modification, but in the case of indirect measurements, the SPDM Requestor endpoint may interact with a local lead Attester that then completes the measurement exchange with a remote Verifier.

This specification anticipates that the attesting device consists of multiple Target Environments, see [5], and a Root of Trust [6] that follows DICE layering [7] principles. Reference Values are assumed to follow the DICE Endorsement Architecture for Devices (CoRIM) [2] schema and the schema extensions defined by this specification.

This specification defines an Evidence table of contents and an Evidence binding for an SPDM Measurement Manifest. SPDM can convey Evidence using the Measurement Block without using an SPDM Measurement Manifest or may use an SPDM Measurement Manifest other than the manifest defined in this specification. These alternatives are out of scope for this specification.

API and protocol definition of or modification to SPDM Requesters / Responders for interactions with a local lead Attester or a remote Verifier is out of scope.

1.1 Key Words

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in this document normative statements are to be interpreted as described in RFC-2119, Key words for use in RFCs to Indicate Requirement Levels.

1.2 Statement Type

Please note a very important distinction between different sections of text throughout this document. There are two distinctive kinds of text: informative comment and normative statement. Because most of the text in this specification will be of the kind normative statement, the authors have informally defined it as the default and, as such, have specifically called out text of the kind informative comment. They have done this by flagging the beginning and end of each informative comment and highlighting its text in gray. This means that unless text is specifically marked as of the kind informative comment, it is considered a normative statement.

EXAMPLE: Start of Informative Comment

This is the first paragraph of 1–n paragraphs containing text of the kind *informative comment* ...

This is the second paragraph of text of the kind *informative comment* ...

This is the nth paragraph of text of the kind *informative comment* ...

To understand the TCG specification the user must read the specification. (This use of MUST does not require any action).

EXAMPLE: End of Informative Comment

2 References

[1] Trusted Computing Group, "DICE Attestation Architecture," [Online]. Available: https://trustedcomputinggroup.org/resource/dice-attestation-architecture/.

[2] Trusted Computing Group, "DICE Endorsement Architecture," 2021. [Online]. Available: https://trustedcomputinggroup.org/resource/dice-endorsement-architecture-for-devices-v1-0-r0-38/.

[3] Internet Engineering Task Force, IETF, "Concise Reference Integrity Manifest," 2023. [Online]. Available: https://datatracker.ietf.org/draft-ietf-rats-corim/.

[4] DMTF, "Security Protocol and Data Model (SPDM) Specification," 21 December 2021. [Online]. Available: https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.2.0.pdf.

[5] Internet Engineering Task Force, "Remote Attestaton Procedures Architecture," 2020. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-rats-architecture/.

[6] Trusted Computing Group, "Hardware Requirements for a Device Identifier Compostion Engine," 2020. [Online]. Available: https://trustedcomputinggroup.org/resource/hardware-requirements-for-a-device-identifier-composition-engine/.

[7] Trusted Computing Group, "DICE Layering Architecture," 2020. [Online]. Available: https://trustedcomputinggroup.org/resource/dice-layering-architecture/.

[8] Trusted Computing Group, "DICE Certificate Profiles, Version 1.0, Revision 0.01," [Online]. Available: https://trustedcomputinggroup.org/resource/dice-certificate-profiles/.

[9] Internet Engineering Task Force, "Internet Security Glossary, Version 2," 2007. [Online]. Available: https://tools.ietf.org/html/rfc4949.

[10] Trusted Computing Group, "TCG Glossary," 2017. [Online]. Available: https://trustedcomputinggroup.org/resource/tcg-glossary/.

[11] Internet Engineering Task Force, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures," 2019. [Online]. Available: https://datatracker.ietf.org/doc/rfc8610/.

[12] Trusted Computing Group, "Reference Integrity Manifest Specification," 2020. [Online]. Available: https://trustedcomputinggroup.org/resource/tcg-reference-integrity-manifest-rim-information-model/.

[13] Internet Assigned Numbers Authority, "Concise Binary Object Representation (CBOR) Tags," [Online]. Available: https://www.iana.org/assignments/cbor-tags/cbor-tags.xhtml.

[14] Internet Engineering Task Force, "Concise Binary Object Representation (CBOR)," [Online]. Available: https://www.rfc-editor.org/rfc/rfc8949.html.

[15] Internet Engineering Task Force, "Concise Software Identification Tags," [Online]. Available: https://www.rfc-editor.org/rfc/rfc9393.html.

3 INTRODUCTION

This specification describes how the attestation capability of a DICE device integrates with the attestation capability of SPDM endpoints. The DICE Layering Architecture [7] defines layered attestation and the DICE Attestation Architecture [1] defines an Evidence format for use with X.509 certificates. The DICE Certificate Profiles [8] defines key purpose OIDs for devices with layered attestation. The SPDM protocol [4], defines a payload structure for conveying attestation Evidence contained in certificates and the SPDM Measurement Block. Processing certificates containing Evidence implies Evidence is extracted from the certificate as part of Evidence appraisal.

This specification defines an SPDM Measurement Manifest structure consisting of a measurement table of contents and the concise evidence schema that can be applied to Evidence collected by an SPDM Responder layer. Beginning with SPDM v1.1, an SPDM Measurement Manifest can be included in an SPDM Measurement Block that formats Evidence according to an Evidence schema.

3.1 Motivation

A device that supports DICE layering may encode layer-specific measurements according to Evidence and Endorsements schemas, see [1] and [2], and may wish to encode SPDM Measurement Block measurements using a similar Evidence schema.

This specification defines the concise evidence schema that is compatible with the CoRIM schema [2] and the schema extensions defined in §7.

Attesters that implement DICE layering typically protect layer-specific Evidence in layer-specific certificates that are dynamically generated when a device boots. One of the layers typically implements the SPDM protocol that is used to transport the certificates to the SPDM Requester, that may forward them to a remote Verifier. The DICE layer that implements the SPDM protocol may also be an Attesting Environment that collects additional measurements from one or more Target Environments that are conveyed using an SPDM Measurement Block.

The SPDM protocol conveys Evidence in several ways as described in §6.4 that may be formatted as defined in §6.3.

This specification defines indirect Evidence that a lead Attester converts to direct Evidence. Direct Evidence is checked against Reference Values that comply with the CoRIM schema and its extensions. This specification does not define ways of appraising direct Evidence.

Some of the terminology and concepts contained in SPDM and DICE specifications overlap but have subtle differences. This specification clarifies the subtleties, provides guidance and, where appropriate, specifies normative behavior for implementations that support both DICE and SPDM capabilities.

3.2 Terminology Convention

This specification uses terminology that is defined by TCG specifications and by the DMTF SPDM specification. To help the reader understand the specific term being used, this specification uses the adjective “SPDM” before terms that should be interpreted according to the SPDM specification. Terms that should be interpreted according to TCG specifications are presented with the adjective “TCG” or do not use an adjective.

For example, the term “SPDM Responder” is interpreted according to the SPDM specification’s term “Responder”.

For example, both the terms “TCG Attester” and “Attester” are to be interpreted according to the use of “Attester” in TCG specifications.

3.2.1 Notation

This section contains notation used in this specification to describe data and operations related to device layering and cryptography.

LEGEND	EXPLANATION
K_{NAME}, K_{Ln}	A private key (K) represents a device or component where NAME, in subscript, names the key or Ln, in subscript, names the key by its layer. Normally, a key named by a layer (Ln) implies the key was derived from a CDI.

PK_{NAME}, PK_{Ln}	A public key (PK) corresponding to a device or component where NAME, in subscript, names the public key or Ln, in subscript, names the key by its layer. Normally, a public key named by a layer (Ln) implies the key was derived from a CDI.
TCI_{Ln}	A Trusted Computing Base (TCB) Component Identifier (TCI) is a layer-specific measurement, where Ln identifies the layer-specific TCI.
CDI_{Ln}	A Compound Device Identifier (CDI) is a cryptographic measurement of a layer, derived by the previous layer, where the subscript $_{Ln}$, names the layer-specific CDI.
[m]K	A message (m) that is digitally signed using the key (K).
$[PK_{Ln+1}]K_{Ln}$	A digital certificate containing a subject public key (PK_{Ln+1}) in layer (n+1) is signed using an issuer private key (K_{Ln}) from the preceding layer (n).

Table 1: Legend for notational conventions

3.3 Mapping Between SPDM and DICE

This specification uses the SPDM extension for defining an SPDM Measurement Manifest. The manifest contains Evidence that is conveyed from an Attester to a Verifier, see Figure 1. The SPDM Responder and SPDM Requester are not shown in Figure 1 but are located between the Attester and the Verifier.

Start of informative comment

The SPDM protocol is the conveyance mechanism for Evidence contained in DICE certificates and the SPDM Measurement Block. An Attester might have other Evidence that is formatted according to other specifications. This specification does not limit the use of alternative Evidence formats.

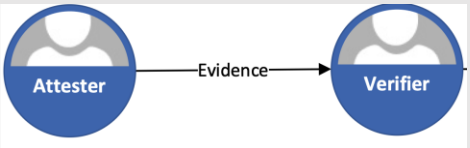

Figure 1: Evidence message exchange between Attester and Verifier roles

End of informative comment

3.3.1 Terminology

The terms and definitions in [9] apply to this specification, along with terms from the TCG Glossary [10] and other referenced documents.

TERM	DEFINITION
Attestation Key	An Attestation Key is a key that can be used to sign attestation Evidence.
DeviceID Key	The Key Pair associated with CDI of Layer 0 is the DeviceID Key. The expectation is that the DeviceID Key remains constant for the useful life of the Attester. Note that a DeviceID Key (or a component of the DeviceID Key, such as PK_{L0}), can be relayed by any higher layer to the Verifier.
ECA Key	An Embedded Certificate Authority (ECA) key is an asymmetric key that is used to sign a certificate for a subsequent layer in a device.
HW RoT	Hardware Root of Trust – A component that performs one or more security-specific functions, such as measurement, storage, reporting, verification, and/or update. It is trusted always to behave in the expected manner, because its misbehavior cannot be detected (such as by measurement) under normal operation [4].

Table 2: Terminology definitions

3.3.2 TCG DICE and DMTF SPDM Terminology Mapping

Terminology commonly used by TCG DICE and DMTF SPDM specifications that have similar meanings are compared in Table 3: SPDM to DICE Terminology Mapping

DICE TERM	SPDM TERM	EXPLANATION
Alias Key	Leaf Certificate Key	The key (either public or private) associated with the end entity certificate.
Attester, lead Attester	Responder	The SPDM Responder is a protocol endpoint role, while the DICE Attester has a role in an Attestation framework. An SPDM Responder endpoint may also implement the Attester role, or these roles may be performed separately. The lead Attester role coordinates collection and conveyance of Evidence obtained from other Attesters. The SPDM Responder performs a lead Attester role when it collects DICE certificates containing Evidence and conveys them to a Verifier via an SPDM session.
DeviceID Key	Device Certificate Key	The key (either public or private) associated with the device certificate that represents a hardware identity that is generated or provisioned during device manufacturing. Typically, this key is unique per instance of the hardware, and is immutable and valid for the useful life of the device.
DeviceID Certificate (Device Identity Certificate)	Device Certificate CA	The device certificate that identifies the device. Typically, it is valid for the useful life of the device. It may contain extensions that describe immutable properties of the device.
Embedded Certificate Authority (ECA)	Alias Intermediate CA	An Embedded Certificate Authority is a device layer that issues certificates to another device layer. The procedure for obtaining certificate signing requests (CSR) from a subsequent layer is documented in the DICE Layering Architecture Specification [7].
ECA Certificate	Alias Intermediate CA Certificate	An intermediate certificate in a DICE Certificate Model has a layered ECA certificate (a.k.a., SPDM Alias Intermediate CA) certificate path. Typically, ECA certificates contain layer identity information and are issued by another ECA or an Intermediate CA.
Alias or End Entity Certificate	Alias or Leaf Certificate	The final certificate in a certificate chain.
Evidence	Measurement Block	The DICE specifications describe Evidence as measurements that are to be matched to Reference Values. Typically, an ECA includes Evidence in an X.509 certificate as a certificate extension, e.g., DiceTcbInfo. The SPDM specifications refer to measurements as typed values that are written into the SPDM Measurement Block.
Verifier	Requester	The SPDM Requester is a protocol endpoint, while the DICE Verifier is an attestation role. An SPDM Requester could be a lead Attester, Verifier or both. For example, host firmware could be a lead Attester for a peripheral device that is an SPDM Responder.

Table 3: SPDM to DICE Terminology Mapping

In addition to a protocol for the conveyance of Attestation Evidence, the SPDM specification defines a mechanism to establish a session for Authenticated Encryption with Associated Data (AEAD) based secured messages. While some operations referenced in this specification depend on the use of secured messages, policies related to the use of secure messages are out of scope for this specification.

3.3.3 Certificate Model Comparison

SPDM v1.2 defines the SPDM Alias Certificate Model that consists of a Root CA, zero or more Intermediate CAs, a Device Certificate CA, zero or more Alias Intermediate CAs, and an Alias Certificate, see Figure 2.

The DICE certificate model is most like the SPDM Alias Certificate Model. It consists of a Root CA, zero or more Intermediate CAs, a DeviceID certificate or layer 0 ECA, zero or more ECAs, and an End Entity. The final intermediate CA issues a DeviceID certificate. The DeviceID layer 0 ECA issues an ECA certificate, and so forth. The final ECA issues an Alias certificate.

SPDM v1.1 introduced the SPDM Device Certificate Model that is incompatible with DICE implementations.

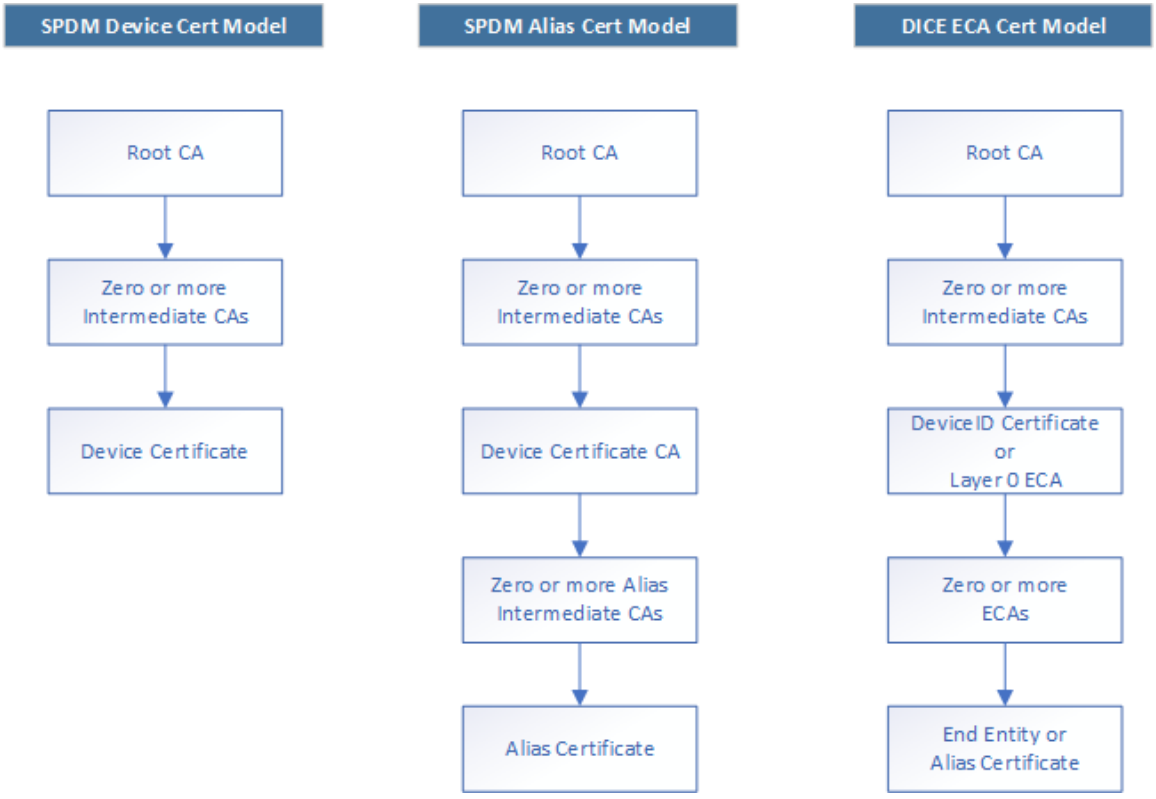

Figure 2: Certificate model comparison

Table 4 compares the SPDM Alias and DICE ECA certificate models as defined by [4] and [7], respectively.

SPDM ALIAS CERTIFICATE MODEL	DICE ECA CERTIFICATE MODEL	COMMENTS
Device Certificate CA Certificate	DeviceID or Layer 0 ECA Certificate	The SPDM Device Certificate CA certificate or DICE DeviceID certificate is the certificate nearest to the root CA certificate that states the device's manufacturer that supplied the device's identifying information. This certificate is signed by a CA that is not embedded in the device. SPDM specifications specify that the Device Certificate CA certificate contains instance unique hardware identity

		information, while DICE specifications specify that the DeviceID certificate contains identity information for the device and is described as the first layer of the device. If the first layer is immutable, then the identity information for the device remains the same for the useful life of the device.
Alias Intermediate CA Certificate	ECA Certificate	The certificate chain may contain SPDM Alias Intermediate CA certificates, a.k.a. DICE ECA certificates. SPDM does not provide guidance on how Alias Intermediate CA certificates are constructed. DICE ECA and Alias Intermediate CA certificates may contain attestation Evidence.
Alias or Leaf Certificate	Alias or End Entity Certificate	The SPDM Alias certificate is the certificate that is furthest, in the certificate path, from the root CA certificate. It is an end-entity certificate. The DICE Alias certificate is also known as the End Entity certificate. Alias certificates may contain attestation Evidence.

Table 4: Comparison of SPDM and DICE certificate models

This specification treats the SPDM Alias Certificate model and DICE ECA Certificate models as equivalent.

4 BEHAVIORAL MODEL

This section describes device capabilities and behavior necessary to be conformant with this specification.

4.1 Minimum Specification Version

Conformant implementations SHALL use SPDM version 1.1 or greater.

Conformant implementations SHALL use DICE Endorsement Architecture for Devices v1.0 [2] or greater, and extensions as defined in this specification, if implementing `concise-evidence-map`, see §6.3.2.

4.2 Required Capabilities for Requesters and Responders

Start of informative comment

The SPDM specification defines a set of capabilities for the SPDM Requester and SPDM Responder, using the SPDM GET_CAPABILITIES request and CAPABILITIES response messages. The SPDM specification does not specify a required set of capabilities for conformant implementations.

End of informative comment

Compliant implementations SHALL use SDPM Capabilities as defined in Table 5 and MAY support other SPDM defined capabilities. Support for SPDM CAPABILITIES not listed in Table 5 is implementation specific.

SPDM CAPABILITY	REQUIREMENT
CERT_CAP	This SPDM capability SHALL be set to 1.
MEAS_CAP	This SPDM capability SHALL be set to 10b.
ALIAS_CERT_CAP[1]	If available, this SPDM capability SHALL be set to 1.
SET_CERT_CAP	If available, this SPDM capability SHOULD be set to 1. If SET_CERT_CAP=1, a certificate chain could be provisioned to a certificate slot with a certificate chain chosen by the SPDM Requester. Provisioned certificate chains could be extended by an ECA.
CSR_CAP	If available, this SPDM capability SHOULD be set to 1. If CSR_CAP=1, a CSR could be used to obtain a certificate for a DICE key that chains to a CA determined by the SPDM Requester.

Table 5 SPDM Capabilities Requirements

4.3 Certificate Model Requirements

Implementations compliant with this specification SHALL use the SPDM Alias Certificate model for SPDM v1.2 or greater, or the DICE ECA Certificate model for SPDM v1.1.

Start of informative comment

There are different extended key usage OIDs defined for the respective certificate models.

The SPDM specification does not define how to protect keys or the associated input material. Those topics are covered in DICE specifications [7], and [6].

End of informative comment

[1] ALIAS_CERT_CAP is only present in the CAPABILITIES response message.

5 CERTIFICATE REQUIREMENTS

This section defines certificate requirements for SPDM and DICE certificate models given a layered device architecture. It also describes certificate chain construction, examples involving multiple certificate chains, and certificate contents requirements.

5.1 Device Layering

5.1.1 Device Layering and Identity

Start of Informative Comment

The layer containing the SPDM Responder, acting as an Attesting Environment, can measure a Target Environment that is part of the SPDM Responder layer, but not included with the previous layer's measurement of the SPDM Responder layer. The Target Environment measurements are reported using the SPDM Measurement Block.

The layer below *Layer n* collects measurements about the SPDM Responder, or alternatively, the SPDM Responder is part of the device Root of Trust.

End of Informative Comment

The SPDM Responder SHALL be part of a DICE layer.

5.1.2 Device Layering and Attestation Architecture

This section defines DICE layering semantics that are preserved by SPDM. An example device shown in **Error! Reference source not found.** illustrates an SPDM payload containing layer-specific measurements and the key used to protect measurement integrity.

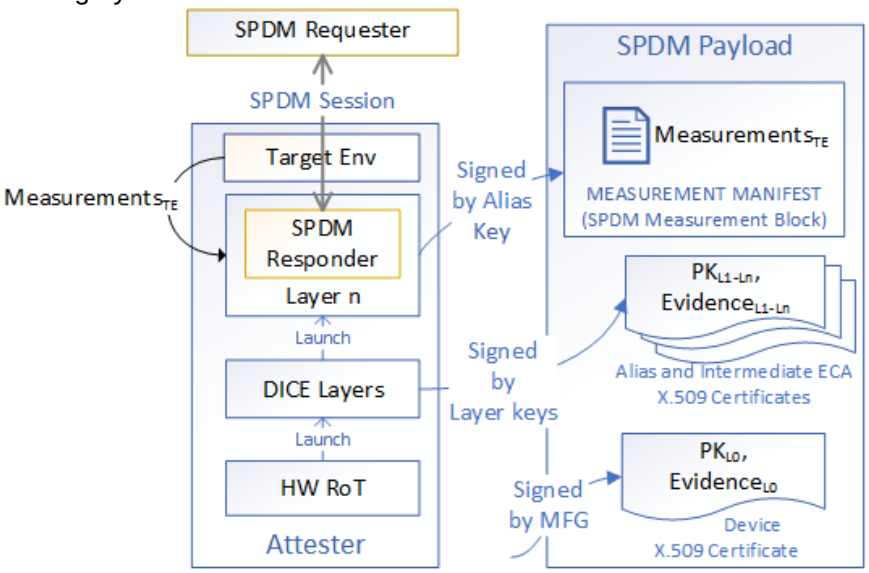

Figure 3: Example Device Layering and Attestation via SPDM

Start of informative comment

Error! Reference source not found. illustrates a layered attestation payload that is conveyed via an SPDM protocol. DICE layer keys are used to protect attestation Evidence and to identify the Attester device.

The Attester has a DICE hardware Root of Trust that launches layer 0 containing a device identity key, K_{L0}, that issues an ECA certificate to the next layer. The layer 0 certificate contains layer 0 Evidence that is signed by the device manufacturer.

The subsequent layers' ECA keys may issue certificates containing attestation Evidence for each layer. The SPDM Responder layer has an alias key that is used to protect the Target Environment measurements collected

by the SPDM Responder layer. The SPDM Measurement Block contains an SPDM Measurement Manifest that contains the Target Environment Evidence collected by the Attesting Environment at the SPDM Responder layer.

End of informative comment

The following DICE layering assumptions are preserved by SPDM:

- The DeviceID Key is provisioned in a secure environment.
- If the Layer 0 implementation of a device is implemented as mutable (even if the implementer policy is that it will never change), then the corresponding DeviceID Key and the DeviceID certificate will change if Layer 0 is changed.
- If the Layer 0 implementation of a device is indeed immutable, then the corresponding DeviceID Key and the DeviceID certificate will be immutable for the useful life of the device.
- The establishment of trust between layers is primarily implemented by an Attesting Environment layer measuring the image of a subsequent Target Environment layer, computing a unique identifier for that layer, and generating a key that can be used to authenticate the Target Environment layer.
- Layered trust extends to the SPDM Responder. This enables the SPDM Responder to convey the Evidence from a preceding layer, such as Layer 0, to a Verifier. In this example, the DeviceID certificate, which represents the Evidence of layer 0 and a unique hardware identity, is conveyed by the SPDM Responder layer.

5.2 Certificate Chain Construction and SPDM Certificate Slots

This section describes certificate chain construction and provisioning using SPDM v1.2 certificate slots.

Start of Informative Comment

In **Error! Reference source not found.**, a device supply chain provisions various certificate slots when the device's configuration changes. For example, a device Manufacturer (MFG) creates the device identity key and issues a manufacturing certificate. An Original Equipment Manufacturer (OEM) integrates the device identity key with additional components or firmware and reissues the device identity certificate under the OEM's root CA. A Value-Added Reseller (VAR) configures the device for a customer and reissues the device identity certificate under the VAR's root CA.

The example device in **Error! Reference source not found.** shows SPDM Alias Certificate Model certificate chains in SPDM certificate slots. Multiple certificate chains are provisioned into multiple SPDM v1.2 certificate slots at different times throughout a manufacturing and deployment sequence. Provisioning a certificate chain into an SPDM v1.2 slot asserts device ownership. If the certificate chain contains a device identity certificate, the device can be authenticated using one of its assigned owners. In this example, at time *t*0, the MFG entity certifies the DeviceID Key at layer 0 (PK_{L0}), using its intermediate CA ("MFG_CA2") and provisions its certificate chain into slot 0. The DeviceID Key (K_{L0}) is protected from misuse by the device implementation.

The OEM, at time *t*1, issues a DeviceID certificate for the DeviceID Key at layer 0 (PK_{L0}) so that the OEM can authenticate the device as part of regular maintenance and device lifecycle support. The OEM provisions its certificate chain into slot 1. The device's key protection mechanisms may prevent the OEM from overwriting slot 0.

DeviceID certificates may certify a key from a dynamic DICE layer. In this example, each entity issues a DeviceID certificate for the layer 0 public key (PK_{L0}).

At time *t*0, layers 1 and 2 do not yet exist, the layer 0 public key PK_{L0} is the DeviceID Key, and the Device's Alias certificate is the DeviceID certificate.

At time *t*1, the OEM configures layer 1 and uses the layer 0 private key (K_{L0}) to certify the layer 1 public key (PK_{L1}). The layer 0 public key remains the DeviceID Key, and the layer 1 certificate, ECA_L1, is now the Alias certificate.

At time *t*2, the VAR updates the device, configures layer 2 and issues another DeviceID certificate using the layer 0 DeviceID Key. The layer 0 public key remains the DeviceID Key, the layer 1, ECA_L1, certificate becomes an Alias Intermediate CA certificate, and the layer 2 certificate, ECA_L2, is the Alias certificate.

In this example, the DICE ECAs dynamically issue Alias and Alias Intermediate CA certificates. If an additional layer is configured, the certificate paths for all certificate slots terminate with the same Alias certificate. Other use cases may exhibit different behavior.

The DICE layer 1 key is specific to the layer 1 configuration and the DICE layer 2 key is specific to the layer 2 configuration. Consequently, if the layer 1 or layer 2 configuration changes, different keys are created. A misconfigured layer could result in the device being unable to perform the intended key operation.

End of Informative Comment

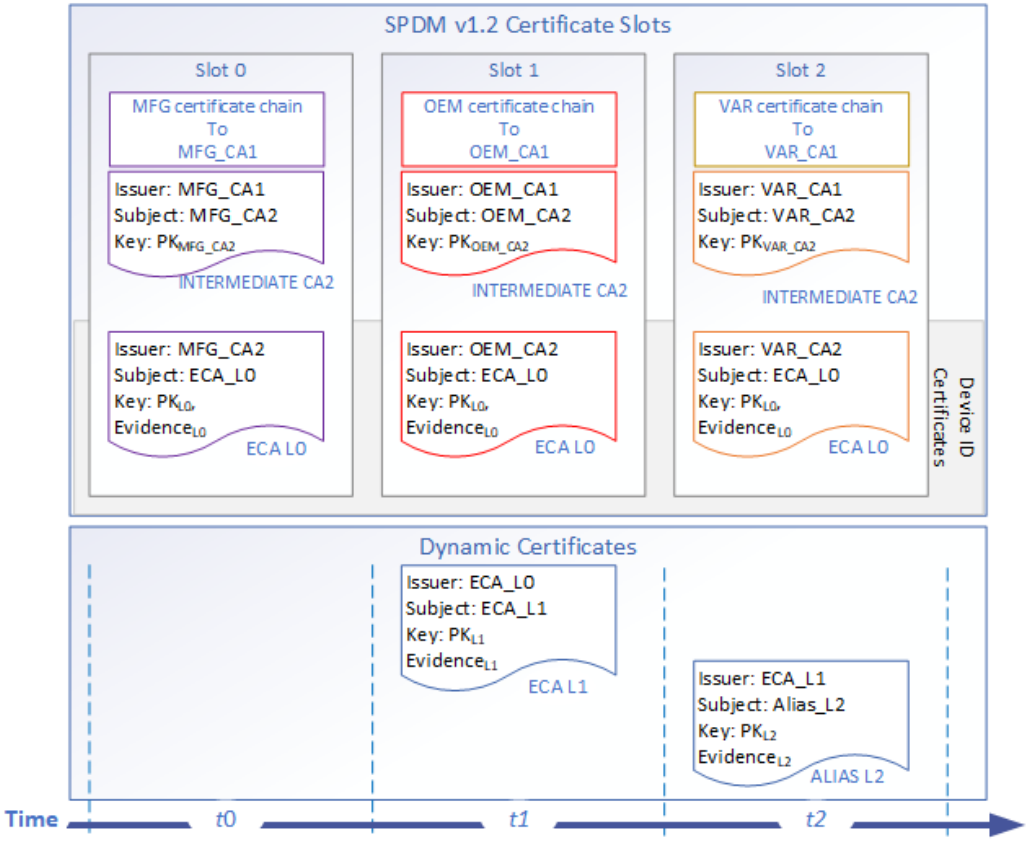

Figure 4: Example usage of SPDM v1.2 certificate slots

5.2.1 Certificate Provisioning

Certificate provisioning of slot 0 SHALL be done in a secure manufacturing environment.

Start of Informative Comment

The specifics of securing a manufacturing environment are outside the scope of this specification.

End of Informative Comment

Slot 0 SHALL contain a DeviceID certificate that certifies the device's DeviceID Key that is provisioned during device manufacturing.

5.3 Certificate Contents

The SPDM specification [4] defines optional and required certificate contents. This section describes DICE conformant certificates, as defined by [1] and [8], that are compatible with SPDM v1.1 or greater.

The DICE Certificate Profiles [8] specification and the SPDM standard [4] define specific requirements for extended key usage OIDs in SPDM Device CA, Alias Intermediate, and Alias Certificates.

SPDM and DICE certificate requirements SHALL be as defined in Table 6:

SPDM CERTIFICATE TYPE	DICE CERTIFICATE TYPE	REQUIREMENTS
Device Certificate CA	DeviceID or IdevID Certificate	The certificate SHOULD contain `id-DMTF-hardwareidentity` The certificate SHOULD contain `tcg-dice-kp-identityInit` If the certified key is used to issue ECA or Intermediate CA certificates, the certificate SHOULD contain `tcg-dice-kp-eca` and `basicConstraints:cA` SHALL be TRUE. If the certified key is used to sign Evidence, the certificate SHOULD contain `tcg-dice-kp-attestInit`. If the certified key is used to sign attestation information other than Evidence, the certificate SHOULD contain `tcg-dice-kp-assertInit`.
Alias Intermediate Certificate	ECA Certificate	If the certified key is used by a mutable environment, the certificate MAY `contain id-DMTF-mutable-certificate`. If the certified key is used to issue ECA or Intermediate Alias Certificates, the certificate SHOULD contain `tcg-dice-kp-eca` and `basicConstraints:cA` SHALL be TRUE. If the certified key is used to sign a device identity challenge, the certificate SHOULD contain tcg-dice-kp-identityLoc. If the certified key is used to sign Evidence, the certificate SHOULD contain the `tcg-dice-kp-attestLoc`. If the certified key is used to sign attestation information other than Evidence, the certificate SHOULD contain `tcg-dice-kp-assertLoc`.
Alias or Leaf	Alias or End Entity	If the certified key is used to sign SPDM response messages, then the certificate SHOULD contain `id-DMTF-ekuresponder-auth`.

Certificate	Certificate	If the certified key is used in a mutable environment, the certificate MAY contain `id-DMTF-mutable-certificate`. If the certified key is used to sign a device identity challenge, the certificate SHOULD contain `tcg-dice-kp-identityLoc`. If the certified key is used to sign Evidence, the certificate SHOULD contain `tcg-dice-kp-attestLoc`. If the certified key is used to sign attestation information other than Evidence, the certificate SHOULD contain `tcg-dice-kp-assertLoc`.

Table 6 Certificate constraints and key usage

Start of Informative Comment

The validity of the DeviceID (a.k.a., SPDM Device Certificate CA) certificate is either set to never expire or set to a value that aligns with the planned life of the device.

The same certificate can have multiple key purpose OIDs enabling the same key to be used for multiple purposes.

Implementations should ensure that certificate authorities certify the intended Target Environment. To ensure that the layer requesting the certificate is authentic, an ECA can check the image of the layer making the request before issuing a certificate containing a digest of that image, see §9.2 of [7].

Implementations should certify a Target Environment once per boot or once per firmware activation, to avoid unnecessary or nuisance certifications.

The DICE layering architecture does not require a DICE layer N to authenticate the next layer (layer N+1) before creating CDI_{N+1} or issuing a certificate with PK_{N+1} as subject key. Instead, the ECA within layer N includes measurements of layer N+1 in that certificate (within a `DiceTcbInfo` or `DiceMultiTcbInfo` extension). The extension is marked critical so a Verifier that does not understand the extension will fail certificate verification.

The measurements taken by layer N+1 can be trusted only if layer N is in a well-known state. A Verifier that is relying on layer N+1 measurements, (reported via DICE, SPDM or other protocols) is expected to check appropriate measurements of layer N (reported in the certificate with PK_N as subject key).

The validity of a layer Intermediate ECA certificate [2] is either set to the value indicating no specific expiry date or set to the value that aligns to the useful life of the product.

End of Informative Comment

6 SPDM MEASUREMENT MANIFEST

6.1 Overview

This section defines concise evidence schema and the table of contents structure used by an SPDM Measurement Manifest. The concise evidence schema builds on the CoMID schema. CoMID schema extensions are also defined here. The measurement manifest table of contents supports manifest extensibility and attester context information.

Schemas are defined using Common Data Definition Language (CDDL) [11] because a CBOR encoding is easily realized. CDDL may also be useful for realizing other encodings.

6.2 SPDM Measurement Manifest Configuration

Figure 5 illustrates an SPDM Measurement Block containing a measurement at index location 0xFD. The DMTFSpecMeasurementValueType at byte location 7 contains the value 1b, indicating the measurement is a raw bit stream. The Measurement payload contains a `tagged-spdm-toc` table of contents as the SPDM Measurement Manifest. Within the table of contents structure there is a `tagged-concise-evidence` structure.

Figure 5 also illustrates the same SPDM Measurement Block where the DMTFSpecMeasurementValueType at byte location 7 contains the value 0b, indicating the measurement is a digest of an SPDM Measurement Manifest.

Start of Informative Comment

SPDM Requesters may request measurements in their native raw bit stream format or a digest. The digested form is a performance optimization if values do not change from previous SPDM request messages.

End of Informative Comment

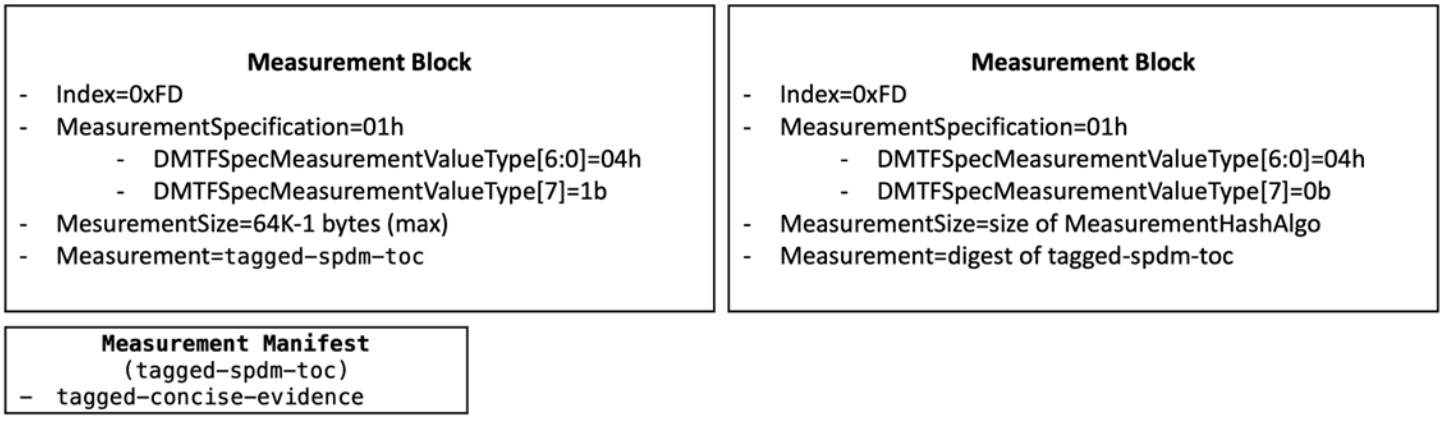

Figure 5: SPDM Measurement Block configured with Measurement Manifest.

The SPDM v1.2 Measurement Block can support Evidence encoded as a raw bit stream SPDM Measurement Manifest. If used, the SPDM Measurement Block index with value 0xFD contains the manifest.

If using SPDM v1.2, the `MeasurementSpecification` bit SHALL be set to 01h and the `DMTFSpecMeasurementValueType` at bit position [6:0] SHALL be set to 04h.

If using SPDM v1.3, the `MeasurementSpecification` bit SHALL be set to 01h and the `DMTFSpecMeasurementValueType` at bit position [6:0] SHALL be set to 0Ah.

If the `DMTFSpecMeasurementValueType` at bit position [12] has the value 1b, the contents SHALL be a raw bit stream containing a measurement manifest consisting of a CBOR tag, followed by a manifest table of contents, that SHOULD include `tagged-concise-evidence`, see §6.3.

Compliant implementations SHALL support the SPDM raw bit stream option.

If the DMTFSpecMeasurementValueType at bit position [12] has the value 0b, the contents SHALL be a cryptographic digest of the CBOR tag, followed by a manifest table of contents containing $tagged-evidence-type-choice entries.

Compliant implementations SHOULD support the SPDM cryptographic digest option.

6.3 SPDM Measurement Manifest Schema

6.3.1 Table of Contents Schema

The SPDM Evidence Manifest table of contents, tagged-spdm-toc, supports multiple Evidence formats and multiple Evidence instances can be supplied.

Start of Informative Comment

If the Attester developer knows the Reference Values manifest location, the rim-locator is a hint that a Verifier might use to find reference manifests or other information needed by a Verifier.

The first 3 bytes in an SPDM Measurement Manifest table of contents are a CBOR tag, tagged-spdm-toc, that is assigned by the Internet Assigned Numbers Authority (IANA) CBOR registry (see [13]) that can indicate any table of contents structure. This specification defines the spdm-toc-map. As a registered CBOR tag, parsers can easily determine which parsing rules to apply. If a different manifest table of contents is desired, a different CBOR tag can be used.

The table of contents, spdm-toc-map, may contain a profile that describes implementation specific conventions, see [2]. A profile identifier indicates whether the Evidence adheres to a profile. A lead Attester or Verifier may use the profile to assist in processing indirect Evidence.

End of Informative Comment

The SPDM Evidence Manifest table of contents schema, a.k.a., tagged-spdm-toc SHALL be defined as follows:

```
tagged-spdm-toc = #6.IANA-TBA(spdm-toc-map)
spdm-toc = spdm-toc-map
spdm-toc-map = {
    &(tagged-evidence: 0) => [ + $tagged-evidence-type-choice]
  ? &(rim-locators: 1) =>  [ + corim-locator-map ] ;see corim
  ? &(profile: 2) => profile-type-choice ; see corim
  * $$spdm-toc-map-extension
}
$tagged-evidence-type-choice /= tagged-concise-evidence
```

The tagged-spdm-toc SHALL support CBOR [14] encoding and SHALL start with CBOR tag #6.IANA-TBA(spdm-toc-map).

The spdm-toc-map SHALL ccontain a tagged-evidence list, an optional rim-locators list, and an optional profile identifier. The spdm-toc-map SHALL be extensible.

The tagged-evidence list SHALL support one or more entries of $tagged-evidence-type-choice where at least one supported Evidence type SHALL be tagged-concise-evidence.

The spdm-toc-map SHALL support a Reference Values locator corim-locator-map as defined by CoRIM.

The spdm-toc-map SHALL support a profile identifier profile-type-choice as defined by CoRIM.

6.3.2 Concise Evidence Schema

This section defines the concise evidence schema, tagged-concise-evidence. A CBOR tag is used to indicate that a concise-evidence-map encoding follows the CBOR tag value.

The `concise-evidence-map` contains an `ev-triples-map`, which is an extensible, non-empty map of Evidence triples.

The `tagged-concise-evidence` statement SHALL be defined as follows:

```
tagged-concise-evidence = #6.IANA-TBA(concise-evidence-map)
concise-evidence = concise-evidence-map
concise-evidence-map = {
    &(ce.ev-triples: 0) => ev-triples-map
          ? &(ce.evidence-id: 1) => $evidence-id-type-choice
  * $$concise-evidence-map-extension
}
$evidence-id-type-choice /= tagged-uuid-type

ev-triples-map = non-empty<{
  ? &(ce.evidence-triples: 0) => [ + reference-triple-record ]
  ? &(ce.identity-triples: 1) => [ + identity-triple-record ]
  ? &(ce.dependency-triples: 2) =>
        [ + domain-dependency-triple-record ]
  ? &(ce.membership-triples: 3) =>
        [ + domain-membership-triple-record ]
  ? &(ce.coswid-triples: 4) => [ + ev-coswid-triple-record ]
  ? &(ce.attest-key-triples: 5) => [ + attest-key-triple-record ]
  * $$ev-triples-map-extension
}>

ev-coswid-triple-record = [
  environment-map,
  [ + ev-coswid-evidence-map ]
]

ev-coswid-evidence-map = {
  ? &(ce.coswid-tag-id: 0) => concise-swid-tag-id
    &(ce.coswid-evidence: 1) => evidence-entry ; see coswid
  ? &(ce.authorized-by: 2) => [ + $crypto-key-type-choice ]
}
```

If the `$tagged-evidence-type-choice` in `spdm-toc-map` is used, it will normally contain Evidence as defined by `tagged-concise-evidence`.

The `tagged-concise-evidence` element SHALL support CBOR [14] encoding and SHALL start with CBOR tag `#6.IANA-TBA(concise-evidence-map`.

Concise Evidence is an `ev-triples-map`, which is a map of Evidence triples records. The `concise-evidence-map` SHALL contain Evidence triple statements as defined by `ev-triples-map`.

The `concise-evidence-map` SHOULD contain an Evidence identifier as defined by `$evidence-id-type-choice`, that SHALL include, as a minimum, a CBOR tag `tagged-uuid-type` as defined by CoRIM.

The `concise-evidence-map` SHALL be extensible.

The `ev-triples-map` SHALL contain at least one triple record.

The `ev-triples-map` SHALL contain any of the following triples:

- `ce.evidence-triples` as an array of `reference-triple-record`, as defined by CoMID.

- `ce.identity-triples` as an array of `identity-triple-record`, as defined by CoMID.
- `ce.attest-key-triples` as an array of `attest-key-triple-record` , as defined by CoMID.
- `ce.coswid-triples` as an array of `ev-coswid-triple-record`, as defined by §7.3.
- `ce.dependency-triples` as an array of `domain-membership-triple-record`, as defined by §7.4.
- `ce.dependency-triples` – as an array of `domain-dependency-triple-record`, as defined by §7.5.

If a `domain-dependency-triple-record` or a `domain-membership-triple-record` is contained in Endorsements, identical triple statements in Evidence SHOULD be omitted.

The `ev-triples-map` SHALL support an extensible set of Evidence triple statements.

The `measurement-values-map` in `reference-triple-record` SHALL be extended to support indirect Evidence as defined by §7.1.

The `ev-coswid-triple-record` SHALL have a subject that consists of an `environment-map` and an object that consists of one or more `ev-coswid-evidence-map`.

The `ev-coswid-evidence-map` SHALL contain a CoSWID `evidence-entry`.

The `ev-coswid-evidence-map` MAY contain a `concise-swid-tag-id` and zero or more `$crypto-key-type-choice` values that identify entities authorized to provide Reference Measurements.

If CoSWID `ev-coswid-triples` is populated, then CoSWID Evidence SHALL be contained in `evidence-entry`.

If CoSWID `ev-coswid-triples` is populated, then `concise-swid-tag-id` SHOULD identify a SWID tag with Reference Values contained in `payload-entry`.

The `authorized-by` item SHOULD be included in Evidence if Reference Values are supplied by an entity other than the supplier of the Attester.

Start of Informative Comment

The concise evidence schema relates software, as defined by CoSWID [15], to a Target Environment, as defined by CoMID.

The `environment-map` value in each Evidence record identifies which Reference Values measurements are to be matched.

The CoSWID tag identifiers (`concise-swid-tag-id`) assist Verifiers in locating the CoSWID tags containing Reference Values.

The CoSWID schema distinguishes between Evidence and Reference Values. The `evidence-entry` map contains Evidence, while `payload-entry` contains Reference Values.

A layered Attesting Environment may depend on another layer's Attesting Environment for correct behavior. Evidence may capture dependency, such as a certificate chain. Alternatively, expected dependencies can be defined in Endorsements. The `domain-dependency-triple-record`, see §7.4, can be used in both Endorsements and Evidence. Verifiers ensure that dependencies found in Evidence match dependencies found in Endorsements.

End of Informative Comment

6.4 Considerations for Conveyance of Evidence

Start of Informative Comment

SPDM can convey Evidence in two ways: (a) inside certificates using Evidence extensions or, (b) inside the SPDM Measurement Manifest. When Evidence is conveyed using the SPDM Measurement Manifest, the measurement

block index will have the value 0xFD and the SPDM `MeasurementSpecification` defines how to interpret the measurement block contents.

Implementations may use indirect measurements in a statically configured SPDM Measurement Block for encoding efficiency purposes, see §6.5.1.

End of Informative Comment

If `spdm-toc-map` is conveyed, the `rim-locators` field in `spdm-toc-map` SHOULD identify locations where Endorsements useful to a Verifier may be retrieved.

6.5 Considerations when Comparing Evidence with Reference Values

Start of Informative Comment

A DICE device that implements SPDM can convey Evidence in two ways: certificates, and the SPDM Measurement Block. When using SPDM Measurement Block, the optional SPDM measurement manifest can be used to format Evidence aligned with the CoRIM schema and other DICE Evidence contained in certificates. The evaluation of both types of Evidence is essential to obtaining a complete understanding of the state of the Attester device.

Endorsement information about an Attester is conveyed to a Verifier from various entities. Endorsements may contain three types of information:

Reference Values – Claims containing measurement values that are known to be good or expected by the Endorser to match Attester supplied Evidence values. Claims contain measurement values that the Verifier can use to determine whether a set of Endorsements is appropriate for a device.

Endorsed Values – Claims that augment Evidence claims. They may describe quality properties, such as Common Criteria [17] and FIPS 140-3 [16] evaluation results. Endorsed Values claims may vouch for the integrity of the Attester that a Verifier may use to satisfy an appraisal policy, or to derive an appropriate Attestation Result.

Authenticated Values – Credentials that assist the Verifier in authenticating the Attester or an endpoint within the Attester. Authenticated Values might also be used during key exchange protocols to negotiate keys used for integrity and confidentiality protections.

Implementation of the Attesting Environment that collects and formats Evidence claims may rely on the Reference Values Claims (or vice versa) to ensure these values are easily compared by the Verifier.

End of informative comment

If an SPDM Measurement Manifest containing `tagged-concise-evidence` is used (see §6.3.2), an Attester device SHALL construct Evidence using `concise-evidence-map` such that Evidence can be matched with Reference Values using `concise-mid-tag`, see [8].

6.5.1 Indirect Measurements

Attestation primarily involves two workflows: (a) conveyance of Evidence from the Attester to the Verifier and (b) conveyance of Endorsements from the Endorser to the Verifier.

Measurements contained in an SPDM Measurement Manifest are referred to, in this specification, as *direct measurements*. Measurements contained in the SPDM Measurement Block at locations other than the manifest location (0xFD), when an SPDM Measurement Manifest is used, are referred to as *indirect measurements*. Indirect measurement values are copied into direct measurements according to a measurement manifest schema (e.g., `concise-evidence-map`).

If an `spdm-indirect` measurement is used, an indirect measurement value SHALL be placed in the SPDM Measurement Block by the implementation at an index location and the SPDM Measurement Manifest SHALL contain an `spdm-indirect-index` value with the index location.

Start of Informative Comment

An SPDM Responder may put measurements in the SPDM Measurement Block at various indexed locations. If an SPDM Measurement Manifest is used, it is possible that duplicate measurements could exist in both the SPDM Measurement Manifest and in SPDM Measurement Block locations other than 0xFD.

An SPDM Responder uses the `spdm-indirect-map` to identify indirect measurements rather than duplicate them in the SPDM Measurement Manifest.

Figure 6 shows a lead Attester translating indirect measurements to direct measurements. Measurements that are formatted according to the SPDM Measurement Manifest schema are already in a direct measurement format.

Indirect measurements are normally integrity protected by an SPDM transcript. The transcript may be included with translated indirect measurements or the lead Attester may protect them with a lead Attester key.

A lead Attester may check the CBOR tag (first 3 bytes) of the measurement manifest to determine whether it is capable of indirect measurement translation. If not, the lead Attester may identify a different lead Attester.

End of Informative Comment

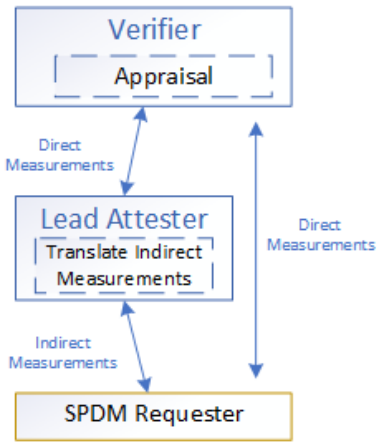

Figure 6: Indirect measurement translation

If a lead Attester supports `tagged-concise-evidence` and an `spdm-indirect` measurement is used, a lead Attester SHOULD translate indirect measurements to direct measurements.

Start of Informative Comment

If a `profile` is supplied, the mapping convention may differ from that described in Table 7.

If indirect measurements are supplied, the authentication status of the original attester contained in the SPDM transcript may be replaced by the authentication status of the lead Attester for direct measurements.

The `spdm-indirect` measurement need only appear in Evidence as this measurement type is meaningless when contained in Reference Values.

To process indirect measurements, a lead Attester replaces the `spdm-indirect` measurements obtained from the SPDM Measurement Block with measurements that comply with `concise-evidence` schema such that a Verifier using CoRIM formatted Reference Values can appraise direct measurements. The SPDM `DMTFSpecMeasurementValueType` field helps determine the CoMID `measurement-values-map` keys.

In cases where indirect measurements are in little-endian byte order, they are converted to the endian convention used by the SPDM Measurement Manifest. CBOR encoding is big-endian.

End of Informative Comment

If a lead Attester supports `tagged-concise-evidence` and an `spdm-indirect` measurement is used, the DMTFMeasurementValueType to CoMID measurement mapping SHOULD be applied according to Table 7.

DMTFSPEC-MEASUREMENTVALUE-TYPE	COMID MEASUREMENT-VALUES-MAP TYPE	COMMENTS
0x85	raw-value	The `raw-value` direct measurement SHALL be set such that the `$raw-value-type-choice` CBOR tag is `560(bytes)`, and `bytes` contains the SPDM debug mode measurement in big-endian format.
0x86	version	The `version-map` direct measurement SHALL be set such that `ver` contains the SPDM version bit stream. If the SPDM raw version is UTF-8 formatted characters, then the `ver` entry SHALL be UTF-8 formatted characters.
0x87	svn	The `svn-type-choice` direct measurement SHALL be set such that the `tagged-svn` contains the CBOR tag `552(svn)`, and `svn` contains the SPDM svn measurement[2].
0..0x7F	digests	The `digest` record `alg:` SHALL be set to the hash type negotiated in the SPDM ALGORITHMS response message and the `digest` record `val:` contains the SPDM digest measurement.
0x80, 0xFF	raw-value	If the SPDM measurement is not a debug mode, version, svn, or digest, then the `$raw-value-type-choice` CBOR tag is `560(bytes)`, and `bytes` contains the SPDM measurement.

Table 7: DMTFMeasurementValueType to CoMID measurement mapping

Start of Informative Comment

If there are multiple measurements that map to the same type (e.g., digest, bit stream), disparate measurements can be disambiguated using CoMID by assigning distinct measurement identifiers. For example, if the first measurement uses a `class-id` measurement identifier OID, “2.1.123.1.15.4.99.1”, as its class identifier, then a second measurement might use the OID, “2.1.123.1.15.4.99.2”, as its class identifier.

End of Informative Comment

If a lead Attester supports `tagged-concise-evidence` and the `spdm-indirect` measurement is used, and no SPDM Measurement Block measurement is found at an `spdm-indirect-map` index location then the `spdm-indirect-map` entry SHOULD be invalidated by the lead Attester.

If a lead Attester supports `tagged-concise-evidence` and the `spdm-indirect` measurement is used, and two or more `spdm-indirect-map` index entries point to the same SPDM Measurement Block measurement, then the `spdm-indirect-map` entry SHOULD be invalidated by the lead Attester.

[2] Note that the SPDM Measurement field holds a little endian SVN, while CBOR integers are big endian.

If a lead Attester supports `tagged-concise-evidence` and the `spdm-indirect` measurement is used, and two or more SPDM Measurement Block measurements collide with the same `measurement-values-map` key, then the `spdm-indirect-map` entry SHOULD then be invalidated by the lead Attester.

If a lead Attester supports `tagged-concise-evidence` and the `spdm-indirect` measurement is used, a lead Attester MAY use `profile` from `spdm-toc-map` to apply a profile-defined convention for mapping indirect measurements.

6.5.2 Appraisal of Evidence

Start of Informative Comment

An SPDM Responder may convey Evidence in two ways: (a) as extensions in a certificate and (b) as measurements in an SPDM measurement block. A Verifier collects the available Evidence from both sources to arrive at a complete set of Evidence. If Evidence contains duplicate measurements, redundant measurements are ignored.

The SPDM Responder may report runtime measurements over a dynamic Target Environment, in which case, use of SPDM v1.2 MEAS_CAP may be helpful.

If the `rim-locators` field or the `tcg-dice-endorsement-manifest-uri` certificate extension are present, the Verifier may use these URIs to locate Reference Values.

An appraisal process for Evidence is summarized in the DICE Endorsement Architecture for Devices [2] (see also [5]).

SPDM v1.2 defines *Device Mode* Evidence. Use of *Device Mode* is fully compatible with concise evidence, see §6.3.2 and DICE certificate extensions, see [1].

End of Informative Comment

Evidence measurements SHOULD NOT be redundantly supplied in an SPDM Measurement Block or in certificates.

7 CoMID SCHEMA EXTENSIONS

This section defines extensions to the CoMID schema [2] that support concise evidence schema definition.

7.1 Indirect Measurements Extension

The CoMID `measurement-values-map` SHALL be extended to define the `spdm-indirect-map` as follows:

```
$$measurement-values-map-extension //= (
  &(spdm-indirect: 12) => spdm-indirect-map
)
spdm-indirect-map = {
  ? &(index: 0) => [ + uint ]
  * $$spdm-indirect-map-extension
}
```

The `spdm-indirect-map` contains an array of unsigned integers corresponding to the SPDM Measurement Block indexes containing indirect measurement values.

The `spdm-indirect` measurement type SHOULD NOT be used as Reference Values.

7.2 Measurement Authorization Extension

The CoMID schema `measurement-map` is extended to include the `authorized-by` entry that identifies the entity authorized to supply reference values. The `authorized-by` entry applies to the measurement values found in `measurement-map`.

The `authorized-by` entry contains a public key identifier of the entity authorized to provide Reference Values. Normally, the supplier that produces the device is authorized to supply the Endorsements.

The `authorized-by` field can be included in Evidence to assert, to a Verifier, which Endorser is expected to supply Reference Values.

The `authorized-by` entry contains the public key identifier from a root or intermediate certificate in the certification path of the certificate that issues the DeviceID Key[3].

Start of Informative Comment

The consequence of not checking for authorization is that a Verifier might accept values that are incorrect.

End of Informative Comment

The CoMID `measurement-map` structure SHALL be extended with `authorized-by` as follows:

```
$$measurement-map-extension //= (
  ? &(authorized-by: 2) => [ + $crypto-key-type-choice ]
)
```

The `$crypto-key-type-choice` SHALL identify the entity authorized to supply Reference Values.

If `authorized-by` is not specified, the manifest signer SHALL identity the entity authorized to supply Reference Values.

[3] Note that the DeviceID Key and keys derived from subsequent DICE layers are device specific, so there is no point in including these keys in the authorized-by entry.

7.3 CoSWID Triple Extension

The CoMID schema `triples-map` is extended with triples that associate a Target Environment with one or more CoSWID [15] tags.

Start of Informative Comment

Reference Values used to match `ev-coswid-triple-record` Evidence require an extension to the CoMID schema so that matching Reference Values can be included in the manifest.

End of Informative Comment

The CoMID `triples-map` structure SHALL be extended to include `coswid-triples` as follows:

```
$$triples-map-extension //= (
&(coswid-triples: 6) => [ + coswid-triple-record ]
)
coswid-triple-record = [
   environment-map,
   [ + concise-swid-tag-id ]
]
concise-swid-tag-id = text / bstr .size 16 ; see CoSWID schema
```

The triple subject Target Environment SHALL be associated with one or more triple object CoSWID tags.

The `coswid-triple-record` is a record containing two elements: `environment-map`, and an array of `concise-swid-tag-id`. The environment map identifies which Target Environment is suitable for installing and loading the software. The array of `concise-swid-tag-id` identifies CoSWID tags.

Start of Informative Comment

Typically, CoSWID tags are found in a CoRIM identified by `concise-swid-tag-id`.

End of Informative Comment

The `coswid-triple-record` differs from the `ev-coswid-triple-record` in that Reference Values are found in a CoSWID tag instead of the CoSWID triple record.

7.4 Domain Membership Triple Extension

The CoMID schema `triples-map` extension point is used to define domain membership triples that associate a domain with one or more Target Environments.

The CoMID `triples-map` structure SHALL be extended to include `membership-triples` as follows:

```
$$triples-map-extension //= (
  ? &(membership-triples: 5) =>
     [ + domain-membership-triple-record ]
)
domain-membership-triple-record = [
  $domain-type-choice,
  [ + $environment-type-choice ]
]
$environment-type-choice /= environment-map
```

The domain members, as defined by `$environment-type-choice`, are indivisibly grouped by the target domain, as defined by `$domain-type-choice`.

The triple subject domain SHALL support having one or more members.

Member environment type is extensible, as defined by `$environment-type-choice`, and supports at least `environment-map`.

7.5 Domain Dependency Triple Extension

The CoMID schema `triples-map` extension point is used to define domain dependency triples that associate a domain with one or more dependent domains.

Start of Informative Comment

Dependency means the *subject* domain's correctness properties rely on the *object* domain(s). For example, the protection of a child domain's key is a prerequisite to a trustworthiness assertion made by the parent domain.

End of Informative Comment

The CoMID `triples-map` structure SHALL be extended to include `dependency-triples` as follows:

```
$$triples-map-extension //= (
  ? &(dependency-triples: 4) =>
     [ + domain-dependency-triple-record ]
)
domain-dependency-triple-record = [
  $domain-type-choice,
  [ + $domain-type-choice ]
]
$domain-type-choice /= uint
$domain-type-choice /= tstr
$domain-type-choice /= tagged-uuid-type
$domain-type-choice /= tagged-oid-type
```

The consistency of the triple subject domain, as defined by `$domain-type-choice`, depends on the consistency of the triple object subordinate domains, as defined by `$domain-type-choice`.

The triple subject domain SHALL support having one or more dependent domains.

Dependent domain type is extensible, as defined by `$domain-type-choice`, and supports at least `uint, tstr, tagged-uuid-type, and tagged-oid-type`.

7.6 Target Environment Properties Extension

The Target Environments have properties that may be indicators of trust worthiness.

Target Environment properties can be expressed in the CoMID schema using `flags-map`.

The CoMID schema SHALL be extended as follows:

```
$$flags-map-extension //= (
  ? &(runtime-meas: 6) => bool
  ? &(immutable: 7) => bool
  ? &(tcb: 8) => bool
)
```

Target Environments properties MAY be TRUE, FALSE, or not specified.

Target Environments MAY have the following properties:

- `runtime-meas`: This property, if set to TRUE, indicates that the Target Environment is measured after being loaded into memory.
- `immutable`: This property, if set to TRUE, indicates that the measured Target Environment is immutable.

- `tcb`: This property, if set to TRUE, indicates that the Target Environment measurements are measurements of a Trusted Computing Base (TCB).

If a property is asserted for a Target Environment that is composed of a subordinate or nested Target Environment, the property SHALL NOT apply to the subordinate.

The `flags-map` properties MAY be asserted in Endorsed Values, Reference Values, or Evidence.

Start of Informative Comment

The `DiceTcbInfo OperationalFlags` is an Evidence format that is matches with Reference Values as defined by `flags-map`.

When `flags-map` is an Endorsed Value, it is not asserted until the Target Environment has been successfully appraised.

If a `flags-map` property is a Reference Value that matches its analog in Evidence, it is accepted. If the property has previously been accepted and it is accepted again, it can be ignored since acceptance of like Claims is idempotent.

When a `flags-map` property is asserted only in Evidence, (i.e., it is not asserted in either Endorsed Values or Reference Values) the property is asserted after all other Evidence Claims have been accepted.

End of Informative Comment

Appendix A - Use Cases and Evidence management

Table 8 summarizes some of the SPDM use cases. The table describes the use case, the type of Evidence conveyed, and the attestation roles that process the Evidence.

USE CASE	EVIDENCE	VERIFIER (REQUESTER)	ATTESTOR (RESPONDER)	NOTES
Asset Tracking	DeviceID Certificate	Request Certificate	Provide Certificate	When tracking hardware identity, a DeviceID Certificate could be part of the certificate chain containing the Alias Certificate or an end entity certificate. The DeviceID Key can be used as an instance specific hardware identity.
Firmware Measurement	Measurement Manifest	Request Measurement(s)	Provide Measurement(s)	An Alias Certificate can be used for change detection.
On Boarding	Alias Certificate	Provision Certificate	Add a Certificate Chain or add Certificates	This is used to provision a new certificate chain[4] or add an Alias Certificate to an existing certificate chain that contains a DeviceID Certificate. Adding an Alias Certificate to a certificate chain containing a DeviceID Certificate requires enabling an Embedded Certificate Authority on the device.
Firmware Update	Alias Certificate	Request Certificate	Provide a Certificate	This is used for firmware or software change detection.
Reprovision, Re-onboarding	Alias Certificate	Provision the Certificate Chain	Verify and store	Performed in a secure environment or at least with a secure session.
Remanufacturing	Device and Alias Certificates	Provision the Certificate chain	Verify and store	This is performed in a secure environment. The device identity will change, leading to a new DeviceID Key, and reissuance of the ECA and Alias certificates.
Decommissioning	Device and Alias Certificates	Provision	Update	This changes the DeviceID Key, thus invalidating any stored and generated certificates tied to the previous DeviceID Key, including

[4] The SPDM specification v1.2 requires that all SPDM certificate slots chain to the same Alias certificate.

				DeviceID Certificate, ECA, and Alias certificates.

Table 8 Use Cases and Evidence Mapping

Appendix B – Sample Concise Evidence in CBOR Diagnostic Format

B.1 SPDM Indirect Evidence in CBOR Diagnostic Format

The following pseudocode is an example of `spdm-indirect` and `authorized-by` in `measurement-map`.

```
/ spdm-toc / 570(  {
  / tagged-evidence / 0 : [
    571(  {
      / ce.ev-triples / 0 : {
        / ce.evidence-triples / 0 : [
          [ /** uses reference-triple-record schema **/
            / environment-map / {
              / class / 0 : {
                / class-id / 0 :
                  / tagged-oid-type /
111(h'6086480186F84D010F046301'), / 2.16.840.1.113741.1.15.4.99.1 /
                / vendor / 1 : "xyzinc.example"
              }
            },
            [
              / measurement-map / {
                / mval / 1 : {
                    / spdm-indirect / 12 : / extends measurement-
values-map / {
                      / index / 0 : [ 1, 2, 3, 4, 5 ]
                    },
                    / digests / 2 : [ [ 1,
h'FFFEFDFCFBFAF9F8F7F6F5F4F3F2F1F0'] ],
                    / raw-value / 4 : 560(h'0123456789')
                },
                / authorized-by / 2 : [
                  / tagged-pkix-base64-key-type / 554("base64_key_X")
                ]
              }
            ]
          ]
        ]
      }
    }  )
  ],
  / rim-locators / 1 : [
    / corim-locator-map / {
      / ** link to reference manifest ** /
      / href / 0 :
32("https://refs.example.com/path/to/reference.corim")
    },
    / corim-locator-map / {
      / ** link to manifest revocation list ** /
      / href / 0 : 32("https://rim-
revoke.example.com/path/to/revocation.xcorim")
    }
  ]
}  )
```

B.2 Domain Dependency Evidence in CBOR Diagnostic Format

The following pseudocode shows a hypothetical case where the SPDM Responder is aware of multiple layers and the trust dependencies between various modules. A real-world case might include only the layer or module that the SPDM Responder describes using SPDM indirect evidence.

```
/ spdm-toc / 570(  {
  / tagged-evidence / 0 : [
    571(  {
      / ce.ev-triples / 0 : {
        / ce.dependency-triples / 2 : [
          [ 3, [ 2 ] ],
          [ 2, [ 1, "L1-extension" ] ],
          [ "L1-extension", [ "XYZ_Root-of-trust" ]],
          [ 1, [ "XYZ_Root-of-trust" ]]
        ]
      }
    }  )
  ],
  / spdm-rim-locators / 1 : [
    / corim-locator-map / {
      / ** link to reference manifest ** /
      / href / 0 :
32("https://refs.example.com/path/to/reference.corim")
    },
    / corim-locator-map / {
      / ** link to manifest revocation list ** /
      / href / 0 : 32("https://rim-
revoke.example.com/path/to/revocation.xcorim")
    }
  ]
}  )
```

B.3 Domain Membership Evidence in CBOR Diagnostic Format

The following pseudocode shows a hypothetical case where the SPDM Responder is aware of multiple layers and domain dependencies. The various measurements, as described by environment-map, are listed for each domain.

```
/ spdm-toc / 570(  {
  / tagged-evidence / 0 : [
    571(  {
      / ce.ev-triples / 0 : {
        / ce.membership-triples / 3 : [
          [ / domain / "XYZ_Root-of-trust",
            [
              / environment-map / {
                / ** A Root of Trust module ** /
                / class / 0 : {
                    / class-id / 0 :
                      / tagged-oid-type / 111(h'0607517B010F6201'), /
2.1.123.1.15.98.1 /
                    / vendor / 1 : "XYZ.example"
                }
              }
            ]
```

```
            ],
            [ / domain / 1,
              [
                / environment-map / {
                  / ** Layer 1 loader module 1 ** /
                  / class / 0 : {
                      / class-id / 0 :
                        / tagged-oid-type / 111(h'0607517B010F0801'), /
2.1.123.1.15.8.1 /
                      / vendor / 1 : "LoadInc.example",
                      / layer / 3 : 1
                  }
                },
                / environment-map / {
                  / ** Layer 1 loader module 2 ** /
                  / class / 0 : {
                    / class-id / 0 :
                      / tagged-oid-type / 111(h'0607517B010F0802'), /
2.1.123.1.15.8.2 /
                    / vendor / 1 : "LoadInc.example",
                    / layer / 3 : 1
                  }
                }
              ]
            ],
            [ / domain / "L1-extension",
              [
                / environment-map / {
                  / ** L1 Extension module 1 ** /
                  / class / 0 : {
                    / class-id / 0 :
                      / tagged-oid-type / 111(h'0607517B010F0903'), /
2.1.123.1.15.9.3 /
                    / vendor / 1 : "LoadInc.example",
                    / layer / 3 : 1
                  }
                }
              ]
            ],
            [ / domain / 2,
              [
                / environment-map / {
                  / ** Layer 2 design module 1 ** /
                  / class / 0 : {
                      / class-id / 0 :
                        / tagged-oid-type / 111(h'0607517B010F0401'), /
2.1.123.1.15.4.1 /
                      / vendor / 1 : "FPGAsRuS.example",
                      / layer / 3 : 2
                  }
                },
                / environment-map / {
                  / ** Layer 2 design module 2 ** /
                  / class / 0 : {
                    / class-id / 0 :
```



```
                    / tagged-oid-type / 111(h'0607517B010F0402'), /
2.1.123.1.15.4.2 /
                  / vendor / 1 : "FPGAsRuS.example",
                  / layer / 3 : 2
                }
              },
              / environment-map / {
                / ** Layer 2 design module 3 ** /
                / class / 0 : {
                  / class-id / 0 :
                    / tagged-oid-type / 111(h'0607517B010F0403'), /
2.1.123.1.15.4.3 /
                  / vendor / 1 : "FPGAsRuS.example",
                  / layer / 3 : 2
                }
              }
            ]
          ],
          [ / domain / 3,
            [
              / environment-map / {
                / ** ISV App module 1 ** /
                / class / 0 : {
                  / class-id / 0 :
                    / tagged-oid-type / 111(h'0607517B010F046301'), /
2.1.123.1.15.4.99.1 /
                  / vendor / 1 : "ISV-App.example"
                }
              },
              / environment-map / {
                / ** ISV App module 2 ** /
                / class / 0 : {
                  / class-id / 0 :
                    / tagged-oid-type / 111(h'0607517B010F046302'), /
2.1.123.1.15.4.99.2 /
                  / vendor / 1 : "ISV-App.example"
                }
              }
            ]
          ]
        ]
      }
    }  )
  ],
  / spdm-rim-locators / 1 : [
    / corim-locator-map / {
      / ** link to reference manifest ** /
      / href / 0 :
32("https://refs.example.com/path/to/reference.corim")
    },
    / corim-locator-map / {
      / ** link to manifest revocation list ** /
      / href / 0 : 32("https://rim-
revoke.example.com/path/to/revocation.xcorim")
    }
```



```
    ]
}   )
```

B.3 CoSWID Evidence in CBOR Diagnostic Format

The following pseudocode is an example where Evidence is represented using the CoSWID schema. A fictitious software measurement is described using the CoSWID schema.

```
/ spdm-toc / 570(  {
  / tagged-evidence / 0 : [
    571(  {
      / ce.ev-triples / 0 : {
        / ce.coswid-triples / 4 : [
          [ /** ev-coswid-triple-record **/
            / environment-map / {
              / class / 0 : {
                / class-id / 0 :
                  / tagged-oid-type / 111(h'0607517B010F046308'), /
2.1.123.1.15.4.99.8 /
                / vendor / 1 : "xyzinc.example"
              }
            },
            [
              / ev-coswid-evidence-map / {
                / ce.coswid-tag-id / 0 : "com.acme.rrd2013-ce-sp1-v4-
1-5-0",
                / ce.coswid-evidence / 1 : { /evidence-entry /
                  / directory / 16: {
                    / fs-name / 24: "rrdetector",
                    / root / 25: "%programdata%",
                    / path-elements / 26: {
                      / file / 17: {
                        / fs-name / 24: "rrdetector.exe",
                        / size / 20: 532712,
                        / hash / 7: [
                            / hash-alg-id / 1,
                            / hash-value /
h'A314FC2DC663AE7A6B6BC6787594057396E6B3F569CD50FD5DDB4D1BBAFD2B6A'
                        ]
                      }
                    }
                  }
                },
                / ce.authorized-by / 2 : [
                  / tagged-pkix-base64-key-type/554("base64_key_X")
                ]
              }
            ]
          ]
        }
    }  )
  ],
  / rim-locators / 1 : [
    / corim-locator-map / {
      / ** link to reference coswids ** /
```



```
      / href / 0 : 32("https://ref-
sw.example.com/path/to/reference.corim")
    },
    / corim-locator-map / {
      / ** link to reference manifest ** /
      / href / 0 : 32("https://ref-
hw.example.com/path/to/reference.corim")
    },
    / corim-locator-map / {
      / ** link to manifest revocation list ** /
      / href / 0 : 32("https://rim-
revoke.example.com/path/to/revocation.xcorim")
    }
  ]
}  )
```

B.4 Properties Flags as Endorsements in CBOR Diagnostic Format

The following pseudocode is an example of properties flags represented as Endorsed Values.

```
/ concise-mid-tag / {
  / tag-identity / 1 : {
    / tag-id / 0 : h'1EACD596F4A34FB699BFAEB58E0A4E49'
  },
    / entities / 2 : [ {
    / entity-name / 0 : "OEM-A",
    / reg-id / 1 : 32("https://oem-a.example"),
    / role / 2 : [ 0 ] / tag-creator /
  } ],
  / linked-tags / 3 : [
    / linked-tag-map / {
        / linked-tag-id / 0 : h'1EACD596F4A34FB699BFAEB58E0A4E47',
        / tag-rel / 1 : / supplements / 0
      },
    / linked-tag-map / {
        / linked-tag-id / 0 : h'AF1CD895BE784ADBB7E9ADD44A65ABF3',
        / tag-rel / 1 : / supplements / 0
      }
    ],
  / triples / 4 : {
      / endorsed-triples / 1 : [
        [
          / environment-map / {
           / class / 0 : {
             / ** Firmware is valid (example) ** /
             / class-id / 0 :
              / tagged-oid-type / 111(h'6086480186F84D010F046301'), /
2.16.840.1.113741.1.15.4.99.1 /
              / vendor / 1 : "fwmfginc.example"
           }
          },
          [
            / measurement-map / {
              / mval / 1 : {
                / flags / 3 : {
```



```
                    / secure / 0 : true,
                    / runtime-meas / 6 : false,
                    / immutable / 7 : true,
                    / tcb / 8 : true
                  }
                },
              / authorized-by / 2 : [
                / tagged-pkix-base64-key-type / 554("base64_key_X")
              ]
            }
          ]
        ]
      ]
  }
}
```


Appendix C – Sample Pretty CBOR

C.1 SPDM Indirect Evidence in Pretty CBOR Format

```
d9 023a                                 # tag(570)
   a2                                   # map(2)
      00                                # unsigned(0)
      81                                # array(1)
         d9 023b                        # tag(571)
            a1                          # map(1)
               00                       # unsigned(0)
               a1                       # map(1)
                  00                    # unsigned(0)
                  81                    # array(1)
                     82                 # array(2)
                        a1              # map(1)
                           00           # unsigned(0)
                           a2           # map(2)
                              00        # unsigned(0)
                              d8 6f     # tag(111)
                                 4c     # bytes(12)
                                    6086480186f84d010f046301 #
"`\x86H\x01\x86\xF8M\x01\x0F\x04c\x01"
                              01        # unsigned(1)
                              6e        # text(14)
                                 78797a696e632e6578616d706c65 #
"xyzinc.example"
                        81              # array(1)
                           a2           # map(2)
                              01        # unsigned(1)
                              a1        # map(1)
                                 0c     # unsigned(12)
                                 85     # array(5)
                                    01  # unsigned(1)
                                    02  # unsigned(2)
                                    03  # unsigned(3)
                                    04  # unsigned(4)
                                    05  # unsigned(5)
                              02        # unsigned(2)
                              81        # array(1)
                                 d9 022a # tag(554)
                                    6c  # text(12)
                                       6261736536345f6b65795f58 #
"base64_key_X"
      01                                # unsigned(1)
      82                                # array(2)
         a1                             # map(1)
            00                          # unsigned(0)
            d8 20                       # tag(32)
               78 30                    # text(48)

68747470733a2f2f726566732e6578616d706c652e636f6d2f706174682f746f2f7265
666572656e63652e636f72696d #
"https://refs.example.com/path/to/reference.corim"
```



```
         a1                                  # map(1)
            00                               # unsigned(0)
            d8 20                            # tag(32)
               78 38                         # text(56)

68747470733a2f2f72696d2d7265766f6b652e6578616d706c652e636f6d2f70617468
2f746f2f7265766f636174696f6e2e78636f72696d # https://rim-
revoke.example.com/path/to/revocation.xcorim
```

C.2 Domain Dependency Evidence in Pretty CBOR Format

```
d9 023a                                    # tag(570)
   a2                                      # map(2)
      00                                   # unsigned(0)
      81                                   # array(1)
         d9 023b                           # tag(571)
            a1                             # map(1)
               00                          # unsigned(0)
               a1                          # map(1)
                  00                       # unsigned(0)
                  81                       # array(1)
                     82                    # array(2)
                        a1                 # map(1)
                           00              # unsigned(0)
                           a2              # map(2)
                              00           # unsigned(0)
                              d8 6f        # tag(111)
                                 4c        # bytes(12)
                                    6086480186f84d010f046301 #
"`\x86H\u0001\x86\xF8M\u0001\u000F\u0004c\u0001"
                              01           # unsigned(1)
                              6e           # text(14)
                                 78797a696e632e6578616d706c65 #
"xyzinc.example"
                        81                 # array(1)
                           a2              # map(2)
                              01           # unsigned(1)
                              a3           # map(3)
                                 0c        # unsigned(12)
                                 a1        # map(1)
                                    00     # unsigned(0)
                                    85     # array(5)
                                       01 # unsigned(1)
                                       02 # unsigned(2)
                                       03 # unsigned(3)
                                       04 # unsigned(4)
                                       05 # unsigned(5)
                                 02        # unsigned(2)
                                 81        # array(1)
                                    82     # array(2)
                                       01 # unsigned(1)
                                       50 # bytes(16)
```



```
fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0 #
"\xFF\xFE\xFD\xFC\xFB\xFA\xF9\xF8\xF7\xF6\xF5\xF4\xF3\xF2\xF1\xF0"
                                 04     # unsigned(4)
                                 d9 0230 # tag(560)
                                    45  # bytes(5)
                                       0123456789 # "\u0001#Eg\x89"
                              02         # unsigned(2)
                              81         # array(1)
                                 d9 022a # tag(554)
                                    6c   # text(12)
                                       6261736536345f6b65795f58 #
"base64_key_X"
      01                                # unsigned(1)
      82                                # array(2)
         a1                             # map(1)
            00                          # unsigned(0)
            d8 20                       # tag(32)
               78 30                    # text(48)

68747470733a2f2f726566732e6578616d706c652e636f6d2f706174682f746f2f7265
666572656e63652e636f72696d #
"https://refs.example.com/path/to/reference.corim"
         a1                             # map(1)
            00                          # unsigned(0)
            d8 20                       # tag(32)
               78 38                    # text(56)

68747470733a2f2f72696d2d7265766f6b652e6578616d706c652e636f6d2f70617468
2f746f2f7265766f636174696f6e2e78636f72696d # "https://rim-
revoke.example.com/path/to/revocation.xcorim"
```

C.3 Domain Membership Evidence in Pretty CBOR Format

```
d9 023a                             # tag(570)
   a2                               # map(2)
      00                            # unsigned(0)
      81                            # array(1)
         d9 023b                    # tag(571)
            a1                      # map(1)
               00                   # unsigned(0)
               a1                   # map(1)
                  03                # unsigned(3)
                  85                # array(5)
                     82             # array(2)
                        71          # text(17)
                           58595a5f526f6f742d6f662d7472757374 #
"XYZ_Root-of-trust"
                        81          # array(1)
                           a1       # map(1)
                              00    # unsigned(0)
                              a2    # map(2)
                                 00 # unsigned(0)
                                 d8 6f  # tag(111)
```



```
                                    48  # bytes(8)
                                       0607517b010f6201 #
"\x06\x06Q{\x01\x0Fb\x01"
                                 01     # unsigned(1)
                                 6b     # text(11)
                                    58595a2e6578616d706c65 #
"XYZ.example"
                        82              # array(2)
                           01           # unsigned(1)
                           82           # array(2)
                              a1        # map(1)
                                 00     # unsigned(0)
                                 a3     # map(3)
                                    00  # unsigned(0)
                                    d8 6f  # tag(111)
                                    48  # bytes(8)
                                       0607517b010f0801 #
"\x06\x06Q{\x01\x0F\b\x01"
                                    01  # unsigned(1)
                                    6f  # text(15)
                                       4c6f6164496e632e6578616d706c65 #
"LoadInc.example"
                                    03  # unsigned(3)
                                    01  # unsigned(1)
                              a1        # map(1)
                                 00     # unsigned(0)
                                 a3     # map(3)
                                    00  # unsigned(0)
                                    d8 6f  # tag(111)
                                    48  # bytes(8)
                                       0607517b010f0802 #
"\x06\x06Q{\x01\x0F\b\x02"
                                    01  # unsigned(1)
                                    6f  # text(15)
                                       4c6f6164496e632e6578616d706c65 #
"LoadInc.example"
                                    03  # unsigned(3)
                                    01  # unsigned(1)
                        82              # array(2)
                           6c           # text(12)
                              4c312d657874656e73696f6e # "L1-extension"
                           81           # array(1)
                              a1        # map(1)
                                 00     # unsigned(0)
                                 a3     # map(3)
                                    00  # unsigned(0)
                                    d8 6f  # tag(111)
                                    48  # bytes(8)
                                       0607517b010f0903 #
"\x06\x06Q{\x01\x0F\t\x03"
                                    01  # unsigned(1)
                                    6f  # text(15)
                                       4c6f6164496e632e6578616d706c65 #
"LoadInc.example"
                                    03  # unsigned(3)
```



```
                                01      # unsigned(1)
                     82                 # array(2)
                        02              # unsigned(2)
                        83              # array(3)
                           a1           # map(1)
                              00        # unsigned(0)
                              a3        # map(3)
                                 00     # unsigned(0)
                                 d8 6f  # tag(111)
                                    48  # bytes(8)
                                       0607517b010f0401 #
"\x06\x06Q{\x01\x0F\x04\x01"
                                 01     # unsigned(1)
                                 70     # text(16)
                                    46504741735275532e6578616d706c65 #
"FPGAsRuS.example"
                                 03     # unsigned(3)
                                 02     # unsigned(2)
                              a1        # map(1)
                                 00     # unsigned(0)
                                 a3     # map(3)
                                    00  # unsigned(0)
                                    d8 6f  # tag(111)
                                       48  # bytes(8)
                                          0607517b010f0402 #
"\x06\x06Q{\x01\x0F\x04\x02"
                                    01  # unsigned(1)
                                    70  # text(16)
                                       46504741735275532e6578616d706c65 #
"FPGAsRuS.example"
                                    03  # unsigned(3)
                                    02  # unsigned(2)
                              a1        # map(1)
                                 00     # unsigned(0)
                                 a3     # map(3)
                                    00  # unsigned(0)
                                    d8 6f  # tag(111)
                                       48  # bytes(8)
                                          0607517b010f0403 #
"\x06\x06Q{\x01\x0F\x04\x03"
                                    01  # unsigned(1)
                                    70  # text(16)
                                       46504741735275532e6578616d706c65 #
"FPGAsRuS.example"
                                    03  # unsigned(3)
                                    02  # unsigned(2)
                     82                 # array(2)
                        03              # unsigned(3)
                        82              # array(2)
                           a1           # map(1)
                              00        # unsigned(0)
                              a2        # map(2)
                                 00     # unsigned(0)
                                 d8 6f  # tag(111)
                                    49  # bytes(9)
```



```
                                        0607517b010f046301 #
"\x06\x06Q{\x01\x0F\x04c\x01"
                                     01     # unsigned(1)
                                     6f     # text(15)
                                        4953562d4170702e6578616d706c65 #
"ISV-App.example"
                               a1           # map(1)
                                  00        # unsigned(0)
                                  a2        # map(2)
                                     00     # unsigned(0)
                                     d8 6f  # tag(111)
                                        49  # bytes(9)
                                           0607517b010f046302 #
"\x06\x06Q{\x01\x0F\x04c\x02"
                                     01     # unsigned(1)
                                     6f     # text(15)
                                        4953562d4170702e6578616d706c65 #
"ISV-App.example"
      01                                    # unsigned(1)
      82                                    # array(2)
         a1                                 # map(1)
            00                              # unsigned(0)
            d8 20                           # tag(32)
               78 30                        # text(48)

68747470733a2f2f726566732e6578616d706c652e636f6d2f706174682f746f2f7265
666572656e63652e636f72696d #
"https://refs.example.com/path/to/reference.corim"
         a1                                 # map(1)
            00                              # unsigned(0)
            d8 20                           # tag(32)
               78 38                        # text(56)

68747470733a2f2f72696d2d7265766f6b652e6578616d706c652e636f6d2f70617468
2f746f2f7265766f636174696f6e2e78636f72696d # https://rim-
revoke.example.com/path/to/revocation.xcorim
```

C.4 CoSWID Evidence in Pretty CBOR Format

```
d9 023a                                 # tag(570)
   a2                                   # map(2)
      00                                # unsigned(0)
      81                                # array(1)
         d9 023b                        # tag(571)
            a1                          # map(1)
               00                       # unsigned(0)
               a1                       # map(1)
                  04                    # unsigned(4)
                  81                    # array(1)
                     82                 # array(2)
                        a1              # map(1)
                           00           # unsigned(0)
                           a2           # map(2)
                              00        # unsigned(0)
```



```
                                d8 6f     # tag(111)
                                   49     # bytes(9)
                                      0607517b010f046308 #
"\u0006\u0007Q{\u0001\u000f\u0004c\b"
                                01        # unsigned(1)
                                6e        # text(14)
                                   78797a696e632e6578616d706c65 #
"xyzinc.example"
                             81           # array(1)
                                a3        # map(3)
                                   00     # unsigned(0)
                                   78 20  # text(32)

636f6d2e61636d652e727264323031332d63652d7370312d76342d312d352d30 #
"com.acme.rrd2013-ce-sp1-v4-1-5-0"
                                01        # unsigned(1)
                                a1        # map(1)
                                   10     # unsigned(16)
                                   a3     # map(3)
                                      18 18 # unsigned(24)
                                      6a  # text(10)
                                         72726465746563746f72 #
"rrdetector"
                                      18 19 # unsigned(25)
                                      6d  # text(13)
                                         2570726f6772616d6461746125 #
"%programdata%"
                                      18 1a # unsigned(26)
                                      a1  # map(1)
                                         11 # unsigned(17)
                                         a3 # map(3)
                                            18 18 # unsigned(24)
                                            6e # text(14)

72726465746563746f722e657865 # "rrdetector.exe"
                                               14 # unsigned(20)
                                               1a 000820e8 #
unsigned(532712)
                                               07 # unsigned(7)
                                               82 # array(2)
                                                  01 # unsigned(1)
                                                  58 20 # bytes(32)

a314fc2dc663ae7a6b6bc6787594057396e6b3f569cd50fd5ddb4d1bbafd2b6a #
"\xA3\u0014\xFC-
\xC6c\xAEzkk\xC6xu\x94\u0005s\x96\xE6\xB3\xF5i\xCDP\xFD]\xDBM\e\xBA\xF
D+j"
                                02        # unsigned(2)
                                81        # array(1)
                                   d9 022a # tag(554)
                                      6c  # text(12)
                                         6261736536345f6b65795f58 #
"base64_key_X"
      01                                  # unsigned(1)
      83                                  # array(3)
```



```
         a1                                # map(1)
            00                             # unsigned(0)
            d8 20                          # tag(32)
               78 32                       # text(50)

68747470733a2f2f7265662d73772e6578616d706c652e636f6d2f706174682f746f2f
7265666572656e63652e636f72696d # "https://ref-
sw.example.com/path/to/reference.corim"
         a1                                # map(1)
            00                             # unsigned(0)
            d8 20                          # tag(32)
               78 32                       # text(50)

68747470733a2f2f7265662d68772e6578616d706c652e636f6d2f706174682f746f2f
7265666572656e63652e636f72696d # "https://ref-
hw.example.com/path/to/reference.corim"
         a1                                # map(1)
            00                             # unsigned(0)
            d8 20                          # tag(32)
               78 38                       # text(56)

68747470733a2f2f72696d2d7265766f6b652e6578616d706c652e636f6d2f70617468
2f746f2f7265766f636174696f6e2e78636f72696d # "https://rim-
revoke.example.com/path/to/revocation.xcorim"
```

C.5 Properties Flags as Endorsements in Pretty CBOR Format

```
a4                                     # map(4)
   01                                  # unsigned(1)
   a1                                  # map(1)
      00                               # unsigned(0)
      50                               # bytes(16)
         1eacd596f4a34fb699bfaeb58e0a4e49 #
"\x1E\xAC\xD5\x96\xF4\xA3O\xB6\x99\xBF\xAE\xB5\x8E\nNI"
   02                                  # unsigned(2)
   81                                  # array(1)
      a3                               # map(3)
         00                            # unsigned(0)
         65                            # text(5)
            4f454d2d41                 # "OEM-A"
         01                            # unsigned(1)
         d8 20                         # tag(32)
            75                         # text(21)
               68747470733a2f2f6f656d2d612e6578616d706c65 #
"https://oem-a.example"
         02                            # unsigned(2)
         81                            # array(1)
            00                         # unsigned(0)
   03                                  # unsigned(3)
   82                                  # array(2)
      a2                               # map(2)
         00                            # unsigned(0)
         50                            # bytes(16)
```



```
            1eacd596f4a34fb699bfaeb58e0a4e47 #
"\x1E\xAC\xD5\x96\xF4\xA3O\xB6\x99\xBF\xAE\xB5\x8E\nNG"
         01                              # unsigned(1)
         00                              # unsigned(0)
      a2                                 # map(2)
         00                              # unsigned(0)
         50                              # bytes(16)
            af1cd895be784adbb7e9add44a65abf3 #
"\xAF\x1C\xD8\x95\xBExJ\xDB\xB7\xE9\xAD\xD4Je\xAB\xF3"
         01                              # unsigned(1)
         00                              # unsigned(0)
   04                                    # unsigned(4)
   a1                                    # map(1)
      01                                 # unsigned(1)
      81                                 # array(1)
         82                              # array(2)
            a1                           # map(1)
               00                        # unsigned(0)
               a2                        # map(2)
                  00                     # unsigned(0)
                  d8 6f                  # tag(111)
                     4c                  # bytes(12)
                        6086480186f84d010f046301 #
"`\x86H\x01\x86\xF8M\x01\x0F\x04c\x01"
                  01                     # unsigned(1)
                  70                     # text(16)
                     66776d6667696e632e6578616d706c65 #
"fwmfginc.example"
            81                           # array(1)
               a2                        # map(2)
                  01                     # unsigned(1)
                  a1                     # map(1)
                     03                  # unsigned(3)
                     a4                  # map(4)
                        00               # unsigned(0)
                        f5               # primitive(21)
                        06               # unsigned(6)
                        f4               # primitive(20)
                        07               # unsigned(7)
                        f5               # primitive(21)
                        08               # unsigned(8)
                        f5               # primitive(21)
                  02                     # unsigned(2)
                  81                     # array(1)
                     d9 022a             # tag(554)
                        6c               # text(12)
                           6261736536345f6b65795f58 # "base64_key_X"
```


Appendix D – CDDL

This section contains Concise Data Definition Language (CDDL) [11] expressions of the CDDL normative contained in this specification.

D.1 CoMID Schema Extensions

This section contains extensions to the CoMID schema that support the `concise-evidence` schema.

```
$$measurement-values-map-extension //= (
  ? &(spdm-indirect: 12) => spdm-indirect-map
)
spdm-indirect-map = {
  &(index: 0) => [ + uint ]
  * $$spdm-indirect-map-extension
}

$$measurement-map-extension //= (
  ? &(authorized-by: 2) => [ + $crypto-key-type-choice ]
)

$$triples-map-extension //= (
  ? &(dependency-triples: 4) =>
     [ + domain-dependency-triple-record ]
)
domain-dependency-triple-record = [
  $domain-type-choice,
  [ + $domain-type-choice ]
]
$domain-type-choice /= uint
$domain-type-choice /= tstr
$domain-type-choice /= tagged-uuid-type
$domain-type-choice /= tagged-oid-type

$$triples-map-extension //= (
  ? &(membership-triples: 5) =>
     [ + domain-membership-triple-record ]
)
domain-membership-triple-record = [
  $domain-type-choice,
  [ + $environment-type-choice ]
]
$environment-type-choice /= environment-map

$$triples-map-extension //= (
  ? &(coswid-triples: 6) => [ + coswid-triple-record ]
)
coswid-triple-record = [
   environment-map,
   [ + concise-swid-tag-id ]
]
concise-swid-tag-id = text / bstr .size 16 ; see coswid

$$flags-map-extension //= (
  ? &(runtime-meas: 6) => bool
```



```
  ? &(immutable: 7) => bool
  ? &(tcb: 8) => bool
)
```

D.2 SPDM Table of Contents and Concise Evidence

This section contains the complete `spdm-toc` and `concise-evidence` CDDL.

```
tagged-spdm-toc = #6.IANA-TBA(spdm-toc-map)
spdm-toc = spdm-toc-map
spdm-toc-map = {
    &(tagged-evidence: 0) => [ + $tagged-evidence-type-choice]
  ? &(rim-locators: 1) =>  [ + corim-locator-map ] ;see comid
  ? &(profile: 2) => profile-type-choice ; see corim
  * $$toc-map-extension
}
$tagged-evidence-type-choice /= tagged-concise-evidence

tagged-concise-evidence = #6.IANA-TBA(concise-evidence-map)
concise-evidence = concise-evidence-map
concise-evidence-map = {
  &(ce.ev-triples: 0) => ev-triples-map
  ? &(ce.evidence-id: 1) => $evidence-id-type-choice
  * $$concise-evidence-map-extension
}
$evidence-id-type-choice /= tagged-uuid-type

ev-triples-map = non-empty<{
  ? &(ce.evidence-triples: 0) => [ + reference-triple-record ]
  ? &(ce.identity-triples: 1) => [ + identity-triple-record ]
  ? &(ce.dependency-triples: 2) =>
     [ + domain-dependency-triple-record ]
  ? &(ce.membership-triples: 3) =>
     [ + domain-membership-triple-record ]
  ? &(ce.coswid-triples: 4) => [ + ev-coswid-triple-record ]
  ? &(ce.attest-key-triples: 5) => [ + attest-key-triple-record ]
  * $$ev-triples-map-extension
}>
ev-coswid-triple-record = [
  environment-map,
  [ + ev-coswid-evidence-map ]
]
ev-coswid-evidence-map = {
  ? &(ce.coswid-tag-id: 0) => concise-swid-tag-id,
    &(ce.coswid-evidence: 1) => evidence-entry ; see coswid
  ? &(ce.authorized-by: 2) => [ + $crypto-key-type-choice ]
}
```