TRUSTED® COMPUTING GROUP

SPECIFICATION

DICE Endorsement Architecture for Devices

Version	1.0
Revision	0.38
November 15, 2022	

Contact: admin@trustedcomputinggroup.org

PUBLISHED

© TCG 2022

DISCLAIMERS, NOTICES, AND LICENSE TERMS

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

This document is copyrighted by Trusted Computing Group (TCG), and no license, express or implied, is granted herein other than as follows: You may not copy or reproduce the document or distribute it to others without written permission from TCG, except that you may freely do so for the purposes of (a) examining or implementing TCG specifications or (b) developing, testing, or promoting information technology standards and best practices, so long as you distribute the document with these disclaimers, notices, and license terms.

Contact the Trusted Computing Group at www.trustedcomputinggroup.org for information on specification licensing through membership agreements.

Any marks and brands contained herein are the property of their respective owners.

ACKNOWLEDGEMENT

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

NAME	COMPANY
Henk Birkholz	Fraunhofer SIT
Yogesh Deshpande	ARM Holdings
Thomas Fossati	ARM Holdings
Dennis Mattoon	Microsoft Corporation
Ned Smith	Intel Corporation

 © TCG 2020

CONTENTS

© TCG 2020

 © TCG 2020

FIGURES

 © TCG 2020

TABLES

 © TCG 2020

1 SCOPE

This specification is intended for software and hardware architects, developers, manufacturers, vendors, service providers, and data engineers seeking to build solutions that leverage DICE and integrate endorsement manifests. It describes the role of endorsement structures to attestation, the composition of an endorsement manifest schema that describes hardware (devices and components), how vendors might define relevant Claim sets, and how those Claim sets can be represented in an interoperable, machine-readable format. It further describes how to construct manifests that describe devices having multiple components and multiple component vendors that each might issue endorsement manifests.

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 comments and normative statements. Because most of the text in this specification will be of the kind normative statements, 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 can be considered a kind of 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).

End of informative comment

 © TCG 2020

2 REFERENCES

[1] Trusted Computing Group, "Attestation Framework Requirements Specification Version 1.0 Revision 0.10," 2021. [Online]. Available: https://trustedcomputinggroup.org/.

[2] Trusted Computing Group, "TCG Glossary," 2017. [Online]. Available: https://www.trustedcomputinggoup.org.

[3] Internet Engineering Task Force, "Remote Attestation Procedures Architecture," 23 April 2021. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-rats-architecture/.

[4] Internet Engineering Task Force, "Concise Binary Object Representation (CBOR) Tags for Object Identifiers," 3 July 2020. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-bormann-cbor-tags-oid-07.

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

[6] Internet Engineering Task Force, "CBOR Object Signing and Encryption (COSE)," July 2017. [Online]. Available: https://datatracker.ietf.org/doc/rfc8152/.

[7] Internet Engineering Task Force, "Concise Software Identification Tags," 2 July 2020. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-sacm-coswid/18/.

[8] Trusted Computing Group, "Hardware Requirements for Device Identifier Composition Engine Level 00, Revision 78," 22 March 2018. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/Hardware-Requirements-for-Device-Identifier-Composition-Engine-r78_For-Publication.pdf.

[9] IEEE, "802.1AR: Secure Device Identity," 2018. [Online]. Available: https://www.ieee.org/.

[10] Trusted Computing Group, "Reference Integrity Manifest Specification Version 1.04," 4 November 2020. [Online]. Available: https://trustedcomputinggroup.org/resource/tcg-pc-client-reference-integrity-manifest-specification/.

[11] ISO/IEC, "Information Technology - Software Asset management - Part 2: Software Identification Tag," 1 October 2015. [Online]. Available: Reference Number: ISO/IEC 19770-2:2015(E).

[12] Internet Engineering Task Force, "Uniform resource Identifier (URI) : Generic Syntax," January 2005. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3986/.

[13] Trusted Computing Group, "DICE Layering Architecture Version 1.0 Revision 0.19," 23 July 2020. [Online]. Available: https://trustedcomputinggroup.org/resource/dice-layering-architecture/.

[14] Trusted Computing Group, "DICE Attestation Architecture Version 1.0 Revision 0.23," 1 March 2021. [Online]. Available: https://trustedcomputinggroup.org/resource/dice-attestation-architecture/.

[15] Trusted Computing Group, "DICE Certificate Profiles Version 1.0 Revision 0.01," 23 July 2020. [Online]. Available: https://trustedcomputinggroup.org/resource/dice-certificate-profiles/.

[16] Trusted Computing Group, "Canonical Event Log Format Version 1.0 Revisition 0.30," 11 December 2020. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TCG_IWG_CEL_v1_r0p30_13feb2021.pdf.

 © TCG 2020

[17] Trusted Computing Group, "TCG PC Client Platform Firmware Integrity Measurement Version 1.0 Revision 43 Family 2.0," 7 May 2021. [Online]. Available: https://trustedcomputinggroup.org/resource/tcg-pc-client-platform-firmware-integrity-measurement/.

[18] Internet Engineering Task Force, "Concise Binary Object Representation (CBOR)," October 2013. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7049.

[19] Internet Engineering Task Force, "The JavaScript Object Notation (JSON) Data Interchange Format," December 2017. [Online]. Available: https://datatracker.ietf.org/doc/rfc8259/.

[20] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Fifth Edition)," 26 November 2008. [Online]. Available: https://www.w3.org/TR/xml/.

[21] NIST, "Guidelines for the Creation of Interoperable Software Identification (SWID) Tags," April 2016. [Online]. Available: https://csrc.nist.gov/publications/detail/nistir/8060/final.

[22] Internet Engineering Task Force, "Concise Reference Integrity Manifest (CoRIM)" 12 July 2021. [Online]. Available: https://datatracker.ietf.org/doc/draft-birkholz-rats-corim/.

[23] Internet Engineering Taskforce, "Use Cases and Requirements for JSON Object Signing and Encryption (JOSE)," April 2014. [Online]. Available: https://datatracker.ietf.org/doc/rfc7165/.

[24] Internet Engineering Task Force, "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages," August 2005. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4108.

[25] Internet Assigned Numbers Authority, "Named Information Hash Algorithm Registry," September 2016. [Online]. Available: https://www.iana.org/assignments/named-information/named-information.txt.

[26] FreeBSD, "tar(5) manual page," 20 May 2004. [Online]. Available: https://www.unix.com/man-page/freebsd/5/tar/.

[27] Internet Assigned Numbers Authority (IANA), "Concise Binary Object Representation (CBOR) Tags," 19 September 2013. [Online]. Available: https://www.iana.org/assignments/cbor-tags/cbor-tags.xhtml. [Accessed 1 November 2021].

[28] Internet Engineering Task Force, "Universal Resource Identifiers in WWW," June 1994. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1630.

[29] Internet Engineering Task Force, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)," June 2010. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5912.

[30] Internet Engineering Task Force, "INTERNET PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION," September 1981. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc791.

[31] Internet Engineering Task Force, "IP Version 6 Addressing Architecture," February 2006. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4291.

[32] IEEE, "Guidelines for Use of Extended Unique Identifier (EUI), Organizationally Unique Identifier (OUI), and Company ID (CID)," 3 August 2017. [Online]. Available: https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/tutorials/eui.pdf.

 © TCG 2020

[33] Internet Engineering Task Force, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," May 2008. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5280.

[34] Internet Engineering Task Force, "The Entity Attestation Token," 7 June 2021. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-rats-eat/.

[35] Internet Engineering Task Force, "A UUID URN Namespace," December 2004. [Online]. Available: https://datatracker.ietf.org/doc/html/RFC4122.

[36] Trusted Computing Group, "TCG Trusted Attestation Protocol Information Model for TPM families 1.2 and 2.0 and DICE Family 1.0 Version 1.0 Revision 0.36," 3 September 2019. [Online]. Available: https://trustedcomputinggroup.org/resource/tcg-tap-information-model/.

[37] ITU-T, "Data Networks and Open System Communications, OSI networking and system aspects-Abstract Syntax Notation One (ASN.1)," July 2002. [Online]. Available: https://www.itu.int/rec/t-rec-x.680/en.

[38] Internet Engineering Task Force, "Tags for Identifying Languages," September 2009. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5646.

[39] Trusted Computing Group, "TCG Glossary Version 1.1 Revision 1.0," 11 May 2017. [Online]. Available: https://trustedcomputinggroup.org/resource/tcg-glossary/.

 © TCG 2020

3 TERMS AND DEFINITIONS

For the purposes of this specification, the following terms and definitions apply. This specification uses attestation related terminology defined in the Trusted Computing Group Attestation Framework Requirements [1]. Additionally, some terms are defined in the Trusted Computing Group Glossary [2].

This specification assumes the reader is familiar with the TCG Attestation Framework Requirements [1] that defines attestation roles (i.e., Attester, Verifier, Relying Party, Endorser, Verifier Owner, and Relying Party Owner) and role messages (i.e., Evidence, Attestation Results, Endorsements, Appraisal Policy for Evidence, and Appraisal Policy for Attestation Results). Similar terminology definitions are found in the IETF Remote Attestation Procedures (RATS) architecture (see [3]).

This specification may use the terms *device* and *component* interchangeably or may use them together to express decomposition semantics. Additionally, *tag* and *module* may be used interchangeably.

3.1 Acronyms

ABBREVIATIONS	DESCRIPTION
CBOR	Constrained Binary Object Representation [4]
CDDL	Concise Data Definition Language [5]
CoRIM	Concise Reference Integrity Manifest (see Section 5.5.2)
CoMID	Concise Module Identifier (see Section 5.5.3)
COSE	CBOR Object Signing and Encryption [6]
CoSWID	Concise Software Identifier [7]
DICE	Device Identifier Composition Engine [8]
IDevID	Initial Device Identity [9]
JOSE	JSON Object Signing and Encryption
OCM	Original Component Manufacturer
ODM	Original Device Manufacturer
OEM	Original Equipment Manufacturer
RIM	Reference Integrity Manifest [10]
SWID	Software Identifier Tags [11]
TCG	Trusted Computing Group
UML	Unified Modeling Language
URI	Uniform Resource Identifier [12]

 © TCG 2020

4 INTRODUCTION

Start of informative comment

This specification defines an attestation endorsement architecture, schema, and manifest profile that informs and supports verification and appraisal of Evidence obtained from a DICE layered device [13] [14] [15]. A primary objective of this specification is the interoperation of Endorsements supplied by one or more Endorsers to one or more Verifiers.

This introductory section provides an overview of concepts related to attestation Endorsements. Endorsements are also discussed in the IETF Attestation Architecture [3] and by the TCG Attestation Framework Requirements Specification [1].

Section 5 describes an Endorsement Manifest Architecture. An information model for the manifest is presented in sections 5.2 and 5.3 followed by a data model definition in sections 5.5 and 5.6.

Section 6 provides guidance for attestation Verifiers with particular attention given to verification of Evidence from DICE certificates [14] [15].

End of informative comment

4.1 Endorsement

Start of informative comment

Endorsers are supply chain entities that supply Endorsements to Verifiers (see Figure 1). Some examples of supply chain entities are device manufacturers, original component manufacturers (OCMs), original device manufacturers (ODMs), original equipment manufacturers (OEMs), firmware vendors, software vendors, pre-release validators, post-release validators, test labs, and quality compliance labs. Some Endorsers define and create components and other ingredients that make up a device (or platform, see [2]). The device, as the Attester, supplies Evidence that a Verifier uses to appraise the Attester's trustworthiness. The Verifier compares the Claims in Evidence with Endorsements to determine whether the expected Reference Values and Endorsed Values differ from the actual values in Evidence.

Endorsements can include Reference Values and Endorsed Values. Reference Values are Claims that the Verifier can expect to find in Evidence from an Attester. Reference Values in Endorsements are compared directly to Claims in Evidence from the Attester. Endorsed Values describe trustworthiness properties that apply to the Attester but are not reported in Evidence from the Attester. Endorsed Values indicate intrinsic properties of a device that are asserted by the Endorser.

The Endorser that produces a module may be the most authoritative entity for making assertions about the module's intrinsic trustworthiness properties or to generate Reference Values for the module. Verifiers should have a clear policy for identifying which Endorsers are authorized to provide Endorsements.

Claims are data provided as Evidence by an Attester. However, the term Claim may also sometimes be used to refer to values in Endorsements. For example: a Verifier may compare a firmware measurement Claim in Evidence to a Reference Value Claim in Endorsements. In this context the term Claim refers to a statement in Evidence, but also to the Reference Value in Endorsements against which the Evidence Claim is compared.

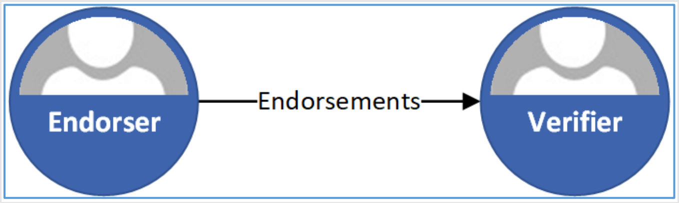

Figure 1: Endorsements message from Endorser to Verifier

Endorsements have the following objectives:

- Define a logical representation of the device and its sub-components.
- Define a logical representation of the device that is machine-readable.
- Define machine-readable Reference Values that are to be matched with Evidence.
- Define machine-readable values for Attester Claims that are not expected to be matched with Evidence.
- Construct Endorsements that can be authenticated by Verifiers.
- Construct device identities that can be authenticated (e.g., IDevIDs [9]).

Generally, Endorsements are information provided by manufacturers and suppliers about the Attester. The type of information provided generally fits into the following three categories:

(i) Class Endorsements

(ii) Instance Endorsements

(iii) Identity Endorsements

Class Endorsements apply to a class of device or component, of which there may be millions or even billions of mass-produced clones. Consequently, the same endorsement document (e.g., manifest) could be reused for all devices in the class.

Instance Endorsements are specific to a device instance. Each device instance requires a different Endorsement Manifest containing the information that is specific for each. Due to the potential for differential scalability and manageability properties, mixing class and instance information in the same Endorsement Manifest should be avoided.

Identity Endorsements describe the relationship between cryptographic keys and identifiers. For example, a device may have an embedded cryptographic key pair used to authenticate the device or to sign attestation Evidence.

For the purposes of this specification, Endorsements are contained in Endorsement Manifests. Endorsements may also be contained in other types of documents as well, for example, identity certificates [16], attribute certificates, or other manifests [10] [17].

End of informative comment

4.2 Attestation Verification

Start of informative comment

Endorsements and Evidence are used to determine which Claims accurately describe the Attester. If Endorsements do not accurately describe the Attester, or the Verifier is unable to process Endorsements, the Verifier could become confused, resulting in incorrect Attestation Results.

Verifiers accept Endorser-supplied Claims if the Endorser is trusted. Typically, this means the Verifier policy contains a trust anchor for the Endorser. Endorsements contain:

1. Reference Values – Claims that must match Evidence before Evidence is accepted as valid.
2. Endorsed Values – Assertions from the Endorser about the Attester that are accepted as valid if Attester Evidence is accepted as valid.

Verifiers obtain an Appraisal Policy for Evidence to determine which Endorsements are relevant for a particular Attester. Accepted Claims are those Claims received from an Attester that a Verifier can verify using Endorsements. By default, all accepted Claims are relevant. However, for a given Relying Party, application, or network deployment, certain Claims might insufficiently or extraneously describe the Attester. The Appraisal Policy for Evidence specifies which of the accepted Claims are required.

End of informative comment

 © TCG 2020

4.3 Endorsement Manifests

Start of informative comment

An Endorsement Manifest is a machine-readable (and possibly human-readable) document. The manifest may have a schema for expressing Claims and relating them to the Attester. An Endorsement Manifest also includes measurements that can be compared with Attester Evidence.

Manifests can be integrity protected and authenticated using cryptography. The Endorsement Manifest signer is the entity that asserts the Endorsement Claims.

In a complex supply chain, it is likely multiple Endorsers will produce Endorsement Manifests. For example, an OCM, being the most authoritative Endorser, may be required to sign a manifest describing the component it produces. Endorsers working independently may complete the manufacture of parts at different times making it impractical for one to become the signer of a single manifest for all components. OEMs often build systems from components supplied by OCMs and software vendors. Another aspect of Endorsement Manifests is that manifests provide links between various other manifests such that the combination of manifests describes a device class.

End of informative comment

4.4 Endorsement Manifest Schemas and Technology

Start of informative comment

Endorsement schemas facilitate interoperability between different Endorsers and between Endorsers and Verifiers. They specify metadata that provides an abstract representation of a class of attestable devices. This specification uses the term device template to refer to the human-readable, and possibly machine-readable, expression of the abstract device class. If multiple Endorsers cooperate to produce a device, each supplying different device components or Reference Values, the device template defines a profile of the Endorsements schema to provide a common understanding of the device class.

The CoRIM schema, expressed in CDDL, is used to generate the CoRIM file that typically is rendered in CBOR [18] or JSON [19] format. This is illustrated in Figure 2.

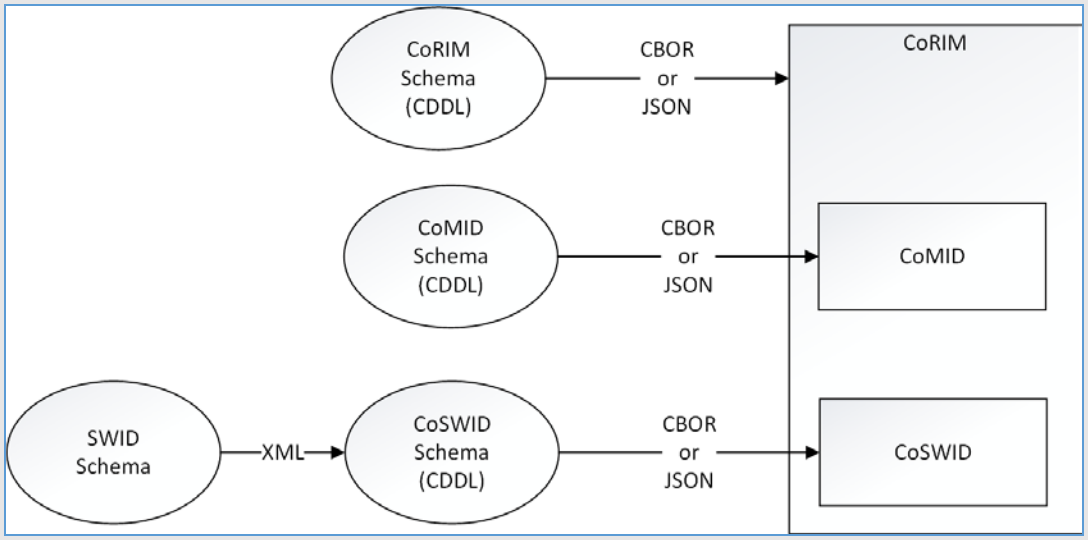

Figure 2: CoRIM Schema informs CoRIM creation.

A SWID Tag [11] is both a schema and an encoding (i.e., XML [20]) that models software asset management. A SWID Tag is adapted to attestation use cases by [21]. CoSWID [7] is a concise representation of a SWID Tag using CDDL [5] as the data definition language. CoSWID defines an extension to a SWID Tag and can be implemented in a memory efficient manner. CoSWID supports at least two encodings, CBOR and JSON. A SWID Tag and CoSWID define a software 'tag' structure that contains metadata and Claims about software packages.

 © TCG 2020

In contrast, Concise Module ID (CoMID) (see Section 5.5.3) tags contain metadata and Claims about hardware and firmware modules. CoMID can be used to describe platforms, devices, and components. Multiple CoMID tags can describe the composition of a device. Multiple vendors may cooperate to issue linked CoMID tags that help describe a supply chain.

CoRIM [22] is a manifest schema that envelopes CoSWID and CoMID expressions. A signed CoRIM may be revoked using a deny list defined by Xcorim (see Section 5.6).

CoRIM is intended to be Root of Trust agnostic but anticipates DICE layering architectures and compositions involving multiple DICE Roots of Trust and Trusted Computing Base (TCB) layers. The TCG Reference Integrity Manifest (RIM) specification [10] uses some of the extensibility features in SWID Tags to reference TPM-centric manifest schemas [10] [17].

For the purposes of this specification, the CoRIM schema is inclusive of both CoMID and CoSWID schemas.

CoRIM manifests may be authored using a variety of popular human-readable formats. Often there is rich tooling available as well. The CoRIM Schema is a CDDL representation that can be used with tooling to automate schema compliance checking as part of the CoRIM authoring process (see Figure 3). Authoring tools generate a CoRIM manifest file that is conveyed to Verifiers. CoRIM files are formatted in either CBOR or JSON.

When Verifiers receive the CoRIM file, the manifest schema (CoRIM CDDL) is used to ensure the manifest file complies with expected CoRIM Schema. Having schema compliant manifests ensures interoperability between Endorsers and Verifiers. Verifiers create an internal representation of the information contained in the manifest to complete Evidence appraisal.

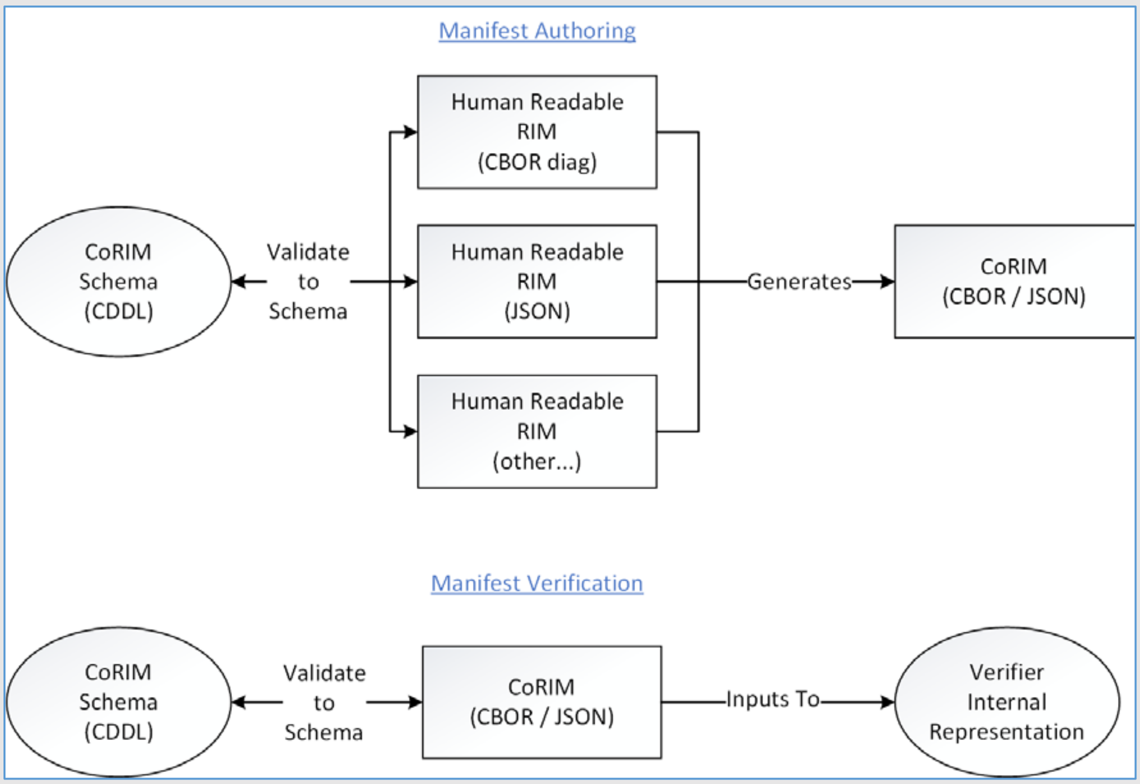

Figure 3: Manifest authoring and verification

End of informative comment

5 ENDORSEMENT MANIFEST ARCHITECTURE

Start of informative comment

The focus of this Endorsement Manifest architecture is to define the information, data models, and encapsulation for exchanging information between the Endorser and Verifier Roles [1]. Endorsements are information produced by Endorsers and consumed by Verifiers.

Endorsement Manifests contain tags that describe the composition and measurements of a platform, device, component, or software. This specification contains an information model for Endorsement Manifests (See Section 5.2). For brevity, the term *module* may be used instead of platform, device, component, or package. Modules have attributes that are described by *tag*s. Software tags describe attributes related to software and software lifecycle. Hardware tags describe attributes related to hardware and firmware. A tag may reference other tags that together model module architecture. Tags contain Claims that describe the trustworthiness properties of the module. Claims consist of a subject, object, and predicate. The predicate defines semantics for relating a subject to an object. Typical Claim expressions relate a Target Environment to a set of Reference Values (e.g., measurements).

Endorsement Manifest Claims can have varying properties, including:

- Claims may apply to a class of device or software where a reference measurement is matched to Evidence.
- Claims may be unique to a module instance.
- Claims may apply to a class of device or software where no matching Evidence is expected.
- Claims may assert a cryptographic key and identifier that can be used to authenticate the module.
- Claims may assert that a key can be used to sign Evidence.

This specification defines both an Endorsement Manifest information model and a data model. The data model may be realized using data definition language. There are several popular data definition languages such as CDDL and ASN.1. Data definition languages generate data encoded output suitable for conveyance over a wire protocol. Common data encoding formats may include JSON, CBOR, BER, and XML. This specification does not specify a data encoding format. Nor does it require a specific data definition language, although it uses CDDL.

Claims about software are described by the SWID Tag and CoSWID data models. Claims about hardware or firmware are described by the CoMID data model (see Section 5.5).

This specification combines CoSWID and CoMID with COSE [6] signatures (and alternatively JOSE [23] signatures) to form a CoRIM manifest.

End of informative comment

5.1 Endorsement Manifest Requirements

The Endorsement Manifest (CoRIM) and encapsulated tags (CoMID, CoSWID) have the following interoperability and security requirements:

1) Endorsement Manifests MUST be machine-readable.
2) Endorsement Manifests MUST be authenticated by an Endorser (entity).
3) Endorsement Manifest signatures MUST have a validity period.
4) Endorsement Manifests MUST support multiple tags.
5) Module Tags MUST support linked tags relationships where the relationship type is explicitly stated.
6) Module Tags MUST support reference and endorsed Claims. (See Section 5.2.3)
7) Module Tags SHOULD support module identity Claims.
8) Module Tags MUST support Reference Values Claims that match DiceTcbInfo Evidence.

 © TCG 2020

Start of informative comment

Designers of Endorsements infrastructure should consider scalability challenges given the potential for millions or billions of instance Endorsements.

Designers of Endorsements infrastructure should anticipate multiple Endorsers (i.e., supply chain entities) working independently to produce complex devices and Endorsements structures.

End of informative comment

5.2 Information Model for Endorsement Manifests

This section describes various aspects of the information model for Endorsement Manifests. Normative language in the information model applies to the Endorsement Manifest data model as defined in section 5.5.

5.2.1 Attester Device Composition

Start of informative comment

Endorsement Manifests rely on module vendors to define Claims in a manner that allows universal verification. An interoperable representation of the module definition can be provided via CoMID tags. Tag creation involves mapping the trustworthiness properties of real-world modules to a tag representation that contains trustworthiness Claims (see Figure 4).

Tags have a tag identifier (e.g., *tag-id*) that identifies tag instances. Tags have a lifecycle. They are created, replaced, updated, and deleted. As part of tag lifecycle management, one tag might link to another tag. For example, a tag that replaces another tag might link to the replaced tag for historical continuity. A linked tag relationship defines link semantics. Additionally, tags are linked when the complete set of Claims is divided across multiple tags.

Software can exist independently of the module it runs on. *Software ID* (SWID) Tags describe software packages. A software tag has a tag identifier (e.g., *tag-id*). The software package is identified by its *software name*.

CoRIM manifests contain both CoMID and CoSWID tags.

End of informative comment

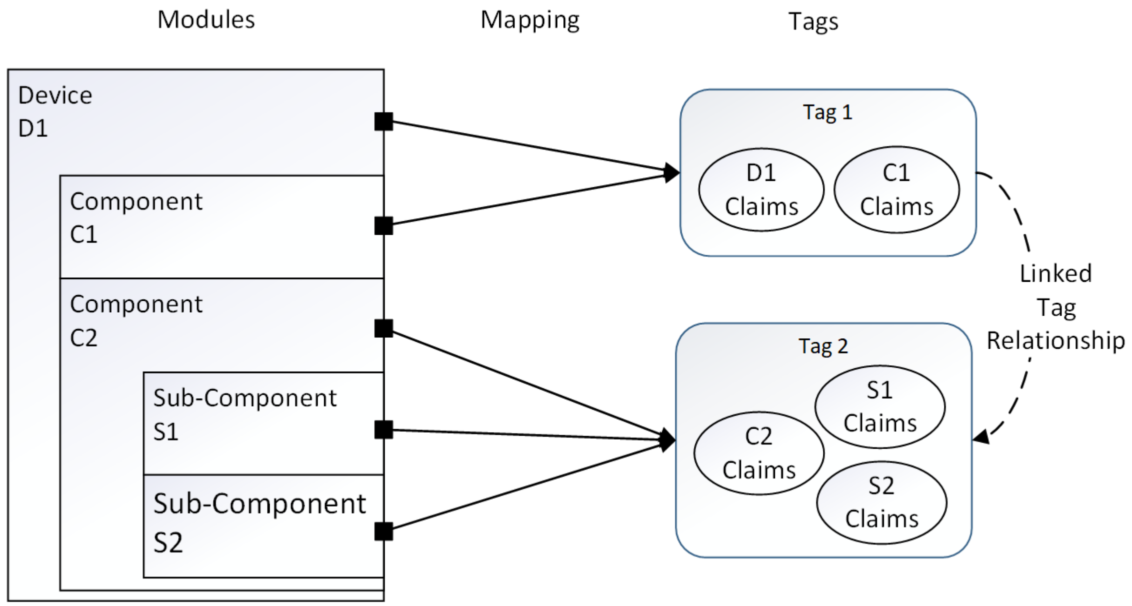

Figure 4: Module to Tag Mapping

 © TCG 2020

5.2.2 Manifest Information Model

Start of informative comment

This specification defines CoRIM, which is a concise representation of an Endorsement Manifest. The overall structure is illustrated in Figure 5. A CoRIM is identified by a universally unique manifest identifier. A CoRIM has a manifest locator. Verifiers may not have all the information needed to verify Evidence. A manifest locator contains Web references to services containing additional or updated manifests. A CoRIM can have both hardware and software tags. CoMID tags contain Claims about hardware and firmware. Claims are statements that relate measurements to Target Environments. CoSWID tags contain Claims about software.

Typically, a manifest is digitally signed. Signing signifies that the Claims contained in tags are valid according to the signer.

End of informative comment

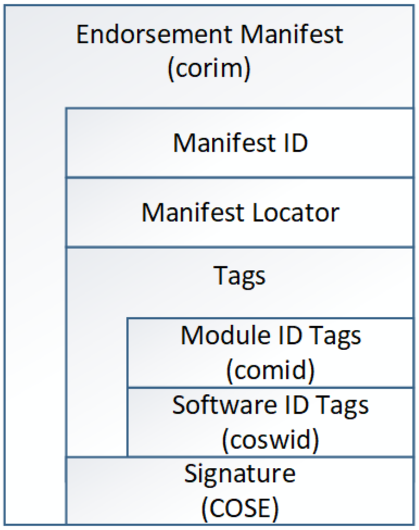

Figure 5: Concise Endorsement Manifest Information Model

A CoRIM manifest MUST contain a statistically unique Manifest ID.

5.2.3 CoMID Tag

Start of informative comment

The CoMID Tag information model (see Figure 6) includes a *Tag ID* that is universally unique. Tag metadata includes a tag version and a list of supply chain entities with their tag lifecycle role. Tags also contain Claims and links to other tags.

The Tag ID and tag version identify tags regardless of the entity that created them. Tag IDs are statistically unique: this property enables a tag lifecycle that may involve recycling previously issued tags. The tag version is part of the Tag metadata field in Figure 6, but is not shown explicitly.

CoMID defines several types of Claims:

- Reference Claims contain Reference Values that are expected to match attested Evidence. Evidence omitting Claims found in Reference Claims, or Evidence containing additional Claims not found in Reference Claims, could be an indication of Attester compromise.

 © TCG 2020

- Endorsed Claims contain assertions about the module that do not require Evidence matching to be accepted by the Verifier. Endorsed Claims may describe properties that are immutable or may reflect assurance properties derived from testing or other supply chain processes.
- Identity Claims contain cryptographic credentials used for authentication. The CoMID tag containing Identity Claims is used by Verifiers to cryptographically challenge the Attester. Credentials could identify a single device or component instance or could identify a group. Identity Claims include cryptographic keying material where the security property of the keying material determines group or singleton identity semantics.
- Attestation Key Claims contain cryptographic keys used to integrity protect Evidence. Verifiers use these Claims to verify signed Evidence.

CoMID tags that contain both class and instance Claims could have scalability and privacy implications. Class Claims can be applied to a large community of modules while instance Claims are unique to a specific module. Instance Claims may have privacy considerations if Claims are used to track individuals.

CoMID tags can link to other tags (i.e., linked tags). Linking involves identifying the linked tag that declares a linking relationship. Link tag relationships describe how to process the linked tag.

End of informative comment

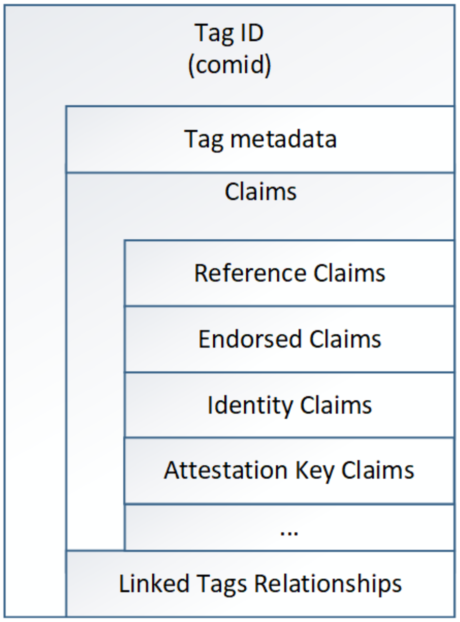

Figure 6: Concise Module ID Tag Information Model

5.2.3.1 Module Metadata

Each CoMID tag MUST contain a statistically unique Tag ID.

CoMID tag metadata MUST include a tag version.

5.2.3.2 Claims

CoMID tags MUST be able to include Reference Claims and Endorsed Claims.

CoMID tags MUST be able to include Claims with class measurements.

CoMID tags SHOULD be able to include Claims with cryptographic identities.

CoMID tags SHOULD support Claims with instance measurements.

 © TCG 2020

5.2.4 Claims

Start of informative comment

Claim triples (see Figure 7) are expressions that consist of a subject, object, and predicate, where the predicate relates the subject to the object. Note that Figure 7 illustrates the Claim triple information model and is labeled using English language terms (e.g., Endorsed Value Triple) in place of more technical structure names (e.g., ENDORSED-VALUES).

The Claim triple for Reference Values relates reference measurements to a Target Environment. For Reference Value Claims, the subject identifies a Target Environment, the object contains measurements, and the predicate asserts that these are the expected (a.k.a., reference) measurements for the Target Environment.

The Claim triple for Endorsed Values declares additional measurements that are valid when a Target Environment has been verified against reference measurements. For Endorsed Value Claims, the subject is either a Target or Attesting Environment, the object contains measurements, and the predicate defines semantics for how the object relates to the subject.

The Claim triple for Device Identity relates one or more cryptographic keys to a device. The subject of an Identity triple uses an instance or class identifier to refer to a device, and a cryptographic key is the object. The predicate asserts that the identity is authenticated by the key. A common application for this triple is device identity.

The Claim triple for Attestation Keys relates one or more cryptographic keys to an Attesting Environment. The Attestation Key triple subject is an Attesting Environment whose object is a key. The predicate asserts that the Attesting Environment signs Evidence that can be verified using the key.

Target Environments may be described by class attributes or by instance identifiers. Class attributes describe a collection of modules. Instance identifiers distinguish each instance of a module.

End of informative comment

This version of the specification defines four triples:

1) REFERENCE-VALUES triple:
    a. Subject: Target Environment ($TE) – identified by class name or instance identifier
    b. Object: One or more measurement values ($MV)
    c. Predicate: The $TE environment has these reference measurements $MV
2) ENDORSED-VALUES triple:
    a. Subject: A Target or Attesting Environment ($E) – identified by class name or instance identifier
    b. Object: One or more measurement values ($MV)
    c. Predicate: The $E environment has these endorsed measurements $MV
3) DEVICE-IDENTITY triple:
    a. Subject: The device ($D) - identified by class name or instance identifier
    b. Object: One or more device identity keys ($K)
    c. Predicate: The $D device is authenticated by the key ($K)
4) ATTESTATION-VERIFICATION-KEY triple:
    a. Subject: Attesting Environment ($AE) – identified by class name or instance identifier
    b. Object: One or more keys ($K)
    c. Predicate: The $AE signs Evidence that can be verified with key ($K)

 © TCG 2020

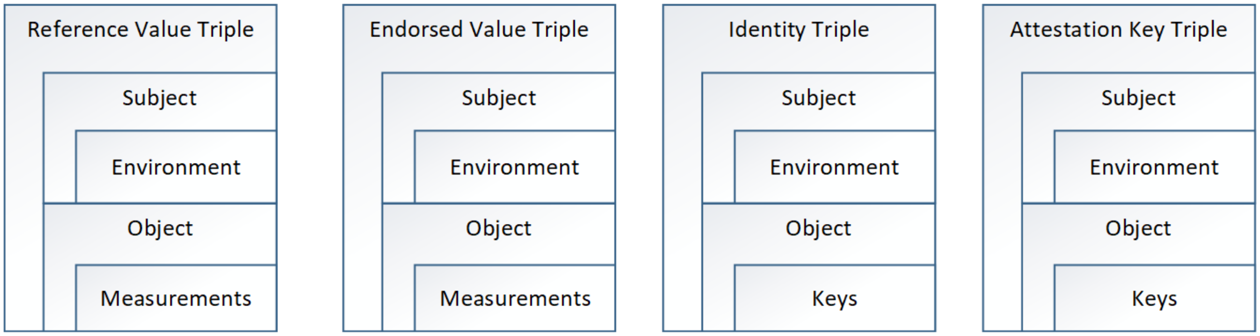

Figure 7: Claim Triple Information Model

Start of informative comment

This specification defines Reference Value Claims that correspond to the DICE `tcbinfo` Evidence extension (see [14]). The `tcbinfo` Evidence Claims include class attributes and measurements that easily match the Claims described by the REFERENCE-VALUES triple.

End of informative comment

A Claim MUST be a logical triple having a subject, object, and predicate, where the predicate relates the subject to the object.

A Claim triple MUST have at least one qualifying attribute as part of its predicate.

A set of Claims MUST be populated with data matching any of the defined triples.

The data definition for Claim triples created by the Endorser MUST be extensible.

5.2.5 Environments

Start of informative comment

The information model describing an *Environment* (see Figure 8) consists of a Class Name and/or an Instance Name. The Class Name consists of class attributes that distinguish the class of environment from other classes. The class attributes include *class-id*, *vendor*, *model*, *layer*, and *index*. The Endorser determines which attributes are needed.

- The *class-id* attribute identifies the environment via well-known identifier. Typically, the class-id is an object identifier (OID) or universally unique identifier (UUID). Use of this attribute is preferred.
- The v*endor* attribute identifies the entity responsible for choosing values for the other class attributes that do not already have naming authority.
- The *model* attribute describes a product, generation, and family.
- The *layer* attribute is used to capture where in a sequence the environment exists. For example, the order in which bootstrap code is executed may have security relevance.
- The *index* attribute is used when there are clones (i.e., multiple instances) of the same class of environment. Each clone is given a different *index* value to disambiguate it from the other clones. For example, given a chassis with several network interface controllers (NIC), each NIC can be given a different *index* value.

End of informative comment

The *Class Name* SHALL consist of at least one of five attributes as follows:

- `class-id` – An environment class identifier. An environment SHOULD have a class-id.

 © TCG 2020

Start of informative comment

For a class-id example, see `hwType` as defined in RFC4108 [24].

End of informative comment

- `vendor` – An identifier of the entity that defined, created, or manufactured the environment. The *vendor* SHOULD be the namespace authority for the other *class* attributes.
- `model` – A text string environment class identifier. If populated, *vendor* MUST also be populated.
- `layer` – A sequence number that relates this Target Environment's sequence of TCB layers to the sequence of other environments.
- `index` – An item number that distinguishes clone environments belonging to the same environment class.

The *Instance Name* SHALL consist of one of the following attributes:

- `uuid` – A universally unique identifier that is reliably bound to the environment.
- `Ueid` – A unique enough identifier that is reliably bound to the environment.

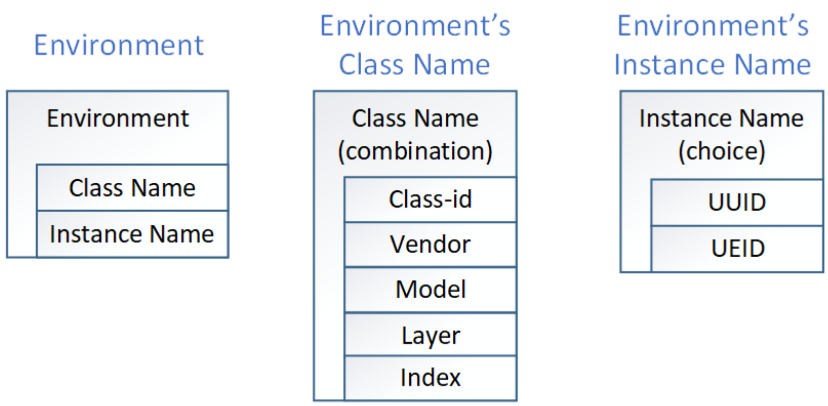

Figure 8: Environment Name Information Model

5.2.6 Measurements

Start of informative comment

Measurements can be of a variety of things including software, firmware, configuration files, read-only memory, fuses, IO ring configuration, partial reconfiguration regions etc. Measurements comprise raw values, digests, or status information.

An environment has one or more measurable elements. Each element can have a dedicated measurement or multiple elements could be combined into a single measurement. Measurements are identified by a class or instance identifier: this is the key in a key-value pair. Figure 9 illustrates structures containing class and instance measurement values. The left-most structure in Figure 9 illustrates a measurement key-value pair. Figure 9 shows both a class and instance key-value pair even though, in reality, a measurement key-value pair represents either a class measurement or an instance measurement, not both.

Class measurements apply generally to a class of something. Instance measurements apply to a specific instance of something. Environments identified by a class identifier have measurements that are common to the class. Environments identified by an instance identifier may have measurements that are specific to that instance. A cryptographic group key is an example of a class identifier, given a group size larger than one. A cryptographic key is an example of an instance identifier.

End of informative comment

 © TCG 2020

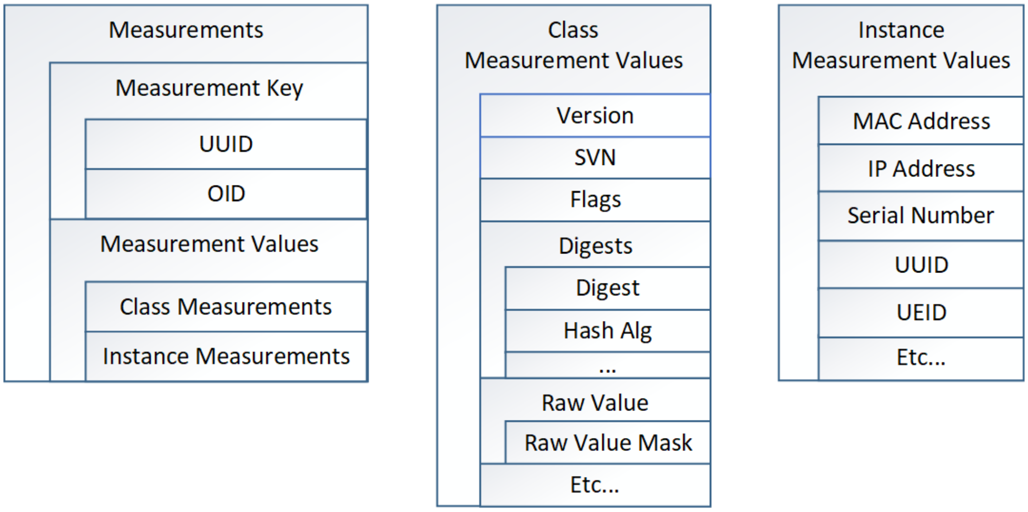

Figure 9: Class and Instance Measurements

5.2.6.1 Class Measurements

Start of informative comment

This specification defines a set of class measurements consisting of *version*, *security version number* (*SVN*), *flags*, *digests*, *raw value,* and *raw value mask*.

The *version* typically changes whenever an environment is updated.

The *svn* (security version number) typically changes only when a security relevant change is made to an environment.

The *flags* field describes security relevant operational modes. For example, security relevant operational modes include whether the environment is in a debug mode, recovery mode, not fully configured, not secure, not replay protected or not integrity protected. The flags field indicates which operational modes are relevant to the module.

The *digests* field contains the digest and hash algorithm (see [25]) used to generate a digest of an environment.

The *raw value* field contains the actual (not hashed) value of the element. The *raw value mask* indicates which bits in the raw value field are relevant for verification. A mask of all ones ("1") means all bits in the raw value field are relevant. Multiple values could be combined to create a single raw value attribute. The vendor determines how to pack multiple values into a single raw value structure. The same packing format is used when collecting Evidence so that Reference Values and collected values are bit-wise comparable. The *vendor* determines the encoding of raw value and the corresponding raw value mask.

The set of possible class measurements is not limited to those defined here.

End of informative comment

Class measurements SHALL include at least the following attributes:

- `version` – A revision control identifier.
- `svn` – A value that increases with each security relevant change.

Start of informative comment

A minimum SVN identifies the lowest possible SVN below which the Target Environment is determined to be unacceptable.

End of informative comment

 © TCG 2020

- `flags` – A value that identifies a security relevant operational mode. The operational mode SHOULD encompass several modes including debug, recovery, not-secure, not-configured, not-replay-protected, not-integrity-protected. A bit value of 1 means the mode that corresponds to that bit is operational. A bit value of 0 means the mode that corresponds to that bit is not operational. If the flags measurement is omitted the operational mode is unknown and not expected.
- `digests` – A set of tuples containing a digest and a hash algorithm identifier. The digest of a raw value is computed using the hash algorithm identifier.
- `raw value` – A bit array of the raw (actual) values.
- `raw value mask` – A bit array mask. A bit value corresponds to a bit value in the *raw value* at the same array position as the mask bit. A mask value of 1 exposes the corresponding bit in the raw value. A mask value of 0 hides the corresponding bit in raw value. The raw value mask SHOULD NOT be included in Evidence.

5.2.6.2 Instance Measurements

Start of informative comment

There are a variety of possible instance measurements. This specification defines five instance measurements that include: *MAC address*, *IP address*, *serial number*, *UUID*, and *UEID*. The measurements data structure (see Figure 9) could be extended to include additional instance measurements.

The Claim triple defines semantics for when it is appropriate (or inappropriate) to include instance measurements.

The set of possible instance measurements is not limited to those defined here.

End of informative comment

Instance measurements expressible in a measurement values map SHALL include at least the following attributes:

- `MAC address` – A EUI-48 or EUI-64 MAC address
- `IP address` – An IPv4 or IPv6 address
- `Serial Number` – A product serial number
- `UUID` – A universally unique identifier
- `UEID` – A unique enough identifier

The measurement values map created by the Endorser MUST be extensible.

5.2.7 CoSWID Information Model

Start of informative comment

The concise software ID tag (CoSWID) information model is formally documented in [7]. This section summarizes the CoSWID schema that consists of a software ID tag structure (see Figure 10) that contains three primary sub-structures:

(i) *package* and tag *metadata*,
(ii) *payload,* and
(iii) *links* to related CoSWID or CoMID tags.

The *payload* consists of a set of Claims called a *resource collection.* There are three types of Claims:

(i) *path elements*,
(ii) *resources*, and
(iii) *processes*.

Path elements consists of *directories* and *files*. Directories have a directory name. Files have a filename and measurement values. Path elements describes pathnames to the files that, when hashed, produce a digest that is contained in the *measurement* structure. A file may have multiple digests when multiple hash algorithms are permitted. This specification does not anticipate use of payloads to carry anything other than path elements.

CoSWID *resource collections* may also contain *resources*. A resource is anything that can be named that isn't otherwise described by path elements or processes. This specification does not anticipate use of CoSWID *resources*.

CoSWID *resource collections* may contain *process* Claims. A process Claim is a measurement over contents of memory. This specification does not anticipate use of CoSWID process.

CoSWID schema also defines *Evidence* which inherits the *payload* schema. However, because this specification defines Endorsement Manifests, CoSWID Evidence is out-of- scope.

End of informative comment

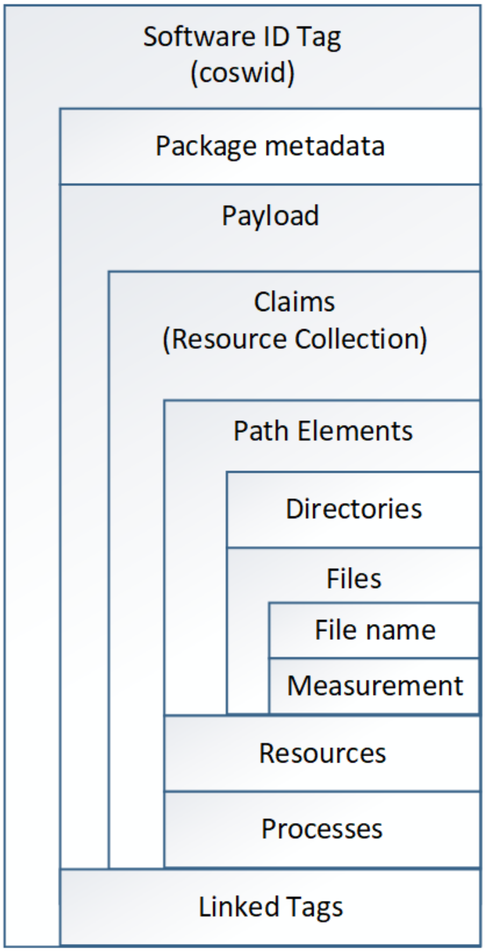

Figure 10: Software ID Tags Information Model

5.3 Device Composition Using Multiple Tags

Start of informative comment

A device can be partitioned into its component parts (e.g., components, modules, environments, elements etc.). Sub-components can be separately attested or reported. The various sub-components may have Endorsement Claims and may also have different suppliers that may create separate CoRIM tags.

 © TCG 2020

Recall that a tag is a collection of Claims about the device or its components. One or more tags could be used to describe the device or its components. Multiple manifests can be used to authenticate, and integrity protect the tags.

A CoRIM might contain multiple tags that describe different devices or components. Tags that describe the same device or component are associated through linking relationships. This section explains linked tags relationships. Tag linking semantics are the same whether working with multi-chip modules or across multiple different modules.

A multi-tag module refers to a device or component that employs multiple tag structures. A multi-tag manifest refers to a single manifest (e.g., CoRIM) that contains multiple tags.

End of informative comment

5.3.1 Tag Linking

Start of informative comment

Linked tags are used to create multi-tag compositions and manage tag lifecycles. Tag linking can occur between any tag type (e.g., CoMID, CoSWID) according to tag relationship semantics.

Linked tag relationships can express device compositions. A multi-tag module can describe a hierarchical composition where a root tag might link to second layer tags, that in turn link to third layer tags etc.

Linked tag relationships help describe multiple Endorsers, each asserting different Claims about a composition. Linked tag relationships can also help to correct malformed tags and update or patch previously minted tags.

End of informative comment

Module tags MUST support inter-tag linking.

Linked tags MUST contain a tag identifier and a link relationship identifier.

A child tag MUST have at least one parent tag.

Linking SHOULD be acyclic.

Linked tags MUST support CoMID-to-CoMID linking.

Linked tags SHOULD support CoMID-to-CoSWID, CoSWID-to-CoSWID and CoSWID-to-CoMID linking.

5.3.1.1 Linked Tag Relationship Types

Start of informative comment

Linked tag relationship types are specific to the type of tag originating the link and the type of tag that is the target of the link.

CoMID to CoMID and CoMID to CoSWID relationship types:

- '*replaces'* to correct erroneous tags
- '*supplements'* to augment existing Claims with additional Claims

CoSWID relationship types are documented in [7].

End of informative comment

5.3.1.2 Linked Tag Relationship Examples

Start of informative comment

This section describes examples of linked tags to highlight a particular relationship type. It uses the two relationship types (i.e., replaces, supplements) that apply to CoMID to CoMID and CoMID to CoSWID linking.

End of informative comment

 © TCG 2020

5.3.1.2.1 Replaces Relationship Example

Start of informative comment

In this example, Alice created a tag in error. Alice corrects the mistake by recreating the tag (tag-id: 11) and increments the tag version so the tag containing erroneous content can be ignored or deleted. Note that, in Figure 11, the position of the tag creator relative to the graphical representation of the tag itself is irrelevant.

End of informative comment

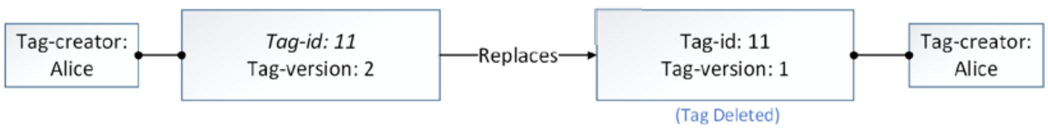

Figure 11: The 'REPLACES' relationship example

5.3.1.2.2 Supplements Relationship Example

Start of informative comment

In this example, Bob provides a firmware validation service for one of Alice's devices. Upon completion of the validation tests, Bob creates Claim-B to describe the validation test results. The device tested in Claim-B matches the device manufactured by Alice in Claim-A. Both tags may be archived for later inspection. Note that, in Figure 12, the position of a tag creator relative to the graphical representation of the tag is irrelevant.

End of informative comment

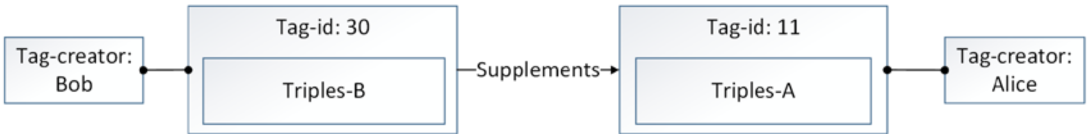

Figure 12: The 'SUPPLEMENTS' relationship example

5.3.1.3 Linked Tags Relationship Requirements

CoMID linked tag relationship types created by the Endorser MUST be extensible.

Start of informative comment

The extensibility provides an Endorser with the flexibility to define its own relationship between tags in the supply chain as per its own use case requirements.

End of informative comment

CoMID to CoMID linked tags MUST support *supplements* and *replaces* relationship types.

CoMID to CoSWID linked tags MUST support the *supplements* relationship type.

CoSWID tags MAY link to CoMID tags using the *supplemental* relationship type.

Given the *supplements* relationship type, the CoMID tag MUST contain Claims that augment the linked tag's Claim set.

Given the 'replaces' relationship type, the CoMID tag MUST contain corrections to the linked tag. The tag ID MUST remain the same. The tag version MUST be incremented.

See also Section 5.5.3.2 and Section 5.5.4.6.

 © TCG 2020

5.3.2 Multi-Tag Manifests

Start of informative comment

A multi-tag manifest contains multiple tags that may or may not be linked. Non-linked tags contain Claims about modules that are unrelated. Linked tags describe related modules. Linking semantics are at the tag level. Linked tags relationships describe the purpose for linking multiple tags. Often, tags are linked because they describe a common device. Tags contain Claims that describe module components and their measurements.

A device consisting of several components can be described using multiple tags where the Claims associated with different tags describe a device and its sub-components. For example, a top-level tag (e.g., the device tag) might link to tags that describe each subcomponent. A CoMID containing linked tags is known as a *multi-tag module*. Multi-tag modules typically describe the composition of modules because each supply chain entity creates a tag for the modules it produces, iteratively. Alternatively, a supply chain entity might merge the various Claim triples into a single tag that fully describes a multi-module device.

End of informative comment

A `corim-map` SHALL be linked if the tags describe the same device or related components.

`concise-tags` in a `corim-map` SHALL NOT be linked if the tags describe unrelated devices or components.

5.3.3 Multi-Endorser Manifests

Start of informative comment

A multi-tag manifest could be divided into multiple manifests signed by different Endorsers. Each of those multiple manifests is called a *multi-Endorser manifest*. Each multi-Endorser manifest could itself be a multi-tag manifest that might be divided into multi-Endorser manifests as well.

For example, a supply chain might consist of a device vendor that works with several component suppliers that produce printed circuit boards (PCBs) and discrete logic components for the PCBs. There may be add-in PCBs that attach to the main board that further consist of discrete logic components. Each supplier independently issues Endorsement Manifests containing tags for each module.

A tag creator might link to other tags created by other entities. The system of linked tags is a composition. The tag creator ensures the multi-tag module is partitioned correctly to ensure the correct supplier can assert the desired Claims when manifests are issued (i.e., digitally signed). Typically, the entity that issues a manifest should be the entity that provides the most credibility for the Claims in that manifest.

Additionally, a composer could collapse a multi-Endorser manifest into a single Endorser manifest by copying Claims from the various multi-Endorser manifests into a tag created by a singleton Endorser. It is assumed that the composer is trusted by the Verifier community to combine the tags and Claims. The Verifier Appraisal Policy for Evidence determines which Endorsers must issue which manifests.

An example multi-Endorser manifest (see Figure 13) describes a device (see Figure 4) that produces two tags and five Claim sets (D1, C1, C2, S1, S2). D1 and C1 are included in Tag-1 in manifest M1 that is issued by Endorser E1. Claim sets C2, S1 and S2 are included in Tag-2 in manifest M2 that is issued by Endorser E2. There is a *supplements* linked tag relationship between Tag-1 and Tag-2. The exact details of the relationship depend upon the exact nature of the Claims.

The third Endorser E3 adds valuable information (such as validation of device compliance) to the device described by M1 and M2 such as compliance validation. The results of compliance validation are expressed in E3 Claims in Tag-3. The Endorser E3 issues a third manifest M3 that links Tag-3 to Tag-1 where the E3 Claims identify C1 as a related module.

End of informative comment

 © TCG 2020

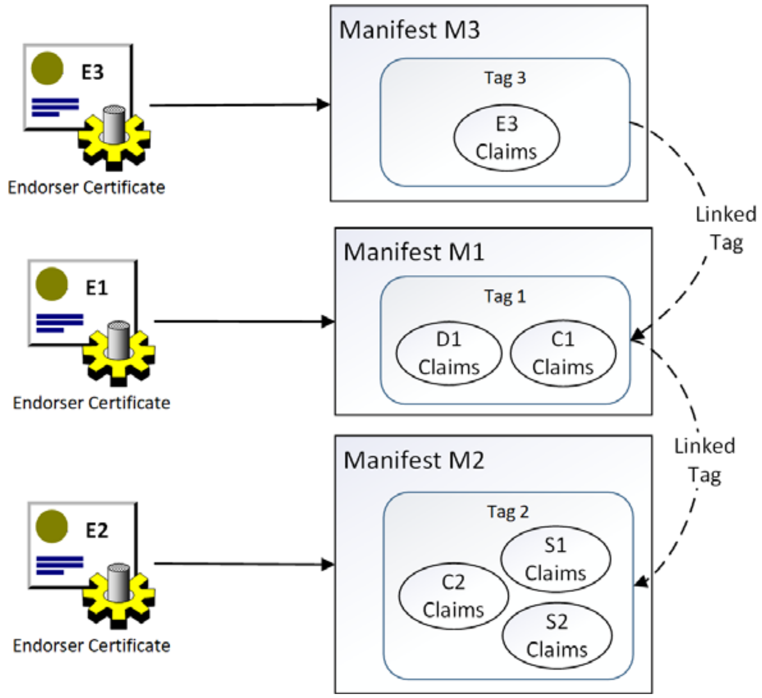

Figure 13: Example Multi-Endorser Manifest

5.4 Archive Files

Start of informative comment

Verifiers need to obtain the manifests, certificates, and trust anchors necessary for appraisal. CoRIM locator maps (e.g., `corim-locator-map` and `link-entry`) may facilitate the discovery and download of such files.

The collection of manifests and certificates used for verification can be made available to Verifiers by archiving them in a file format such as TAR [26] . Alternatively, the manifests that describe device composition and measurements may be provisioned to a Verifier by an Endorser.

Additionally, a Verifier may be provisioned with trust anchors to verify the manifest signature to ensure the integrity and authenticity of CoRIM's that are not directly provisioned.

The security of unsigned manifests and certificates in a file archive is out of scope for this specification.

Multiple CoRIM files can be packaged into a file archive format but note that specification of archive formats is outside the scope of this specification.

Archive files might contain related certificates or other files useful to Verifiers.

End of informative comment

5.5 Data Model for Concise Reference Integrity Manifest (CoRIM)

Start of informative comment

This section describes a data model definition for CoRIM, also known as 'CoRIM schema'. The Endorsement Manifest data model consists of a top-level map structure containing either a signed CoRIM or an unsigned CoRIM. A CoRIM describes one or more devices and their respective components, and sub-components (i.e., modules).

A single manifest could describe multiple modules.

 © TCG 2020

Multiple manifests could describe the same module.

A CoRIM signature indicates that the signer Endorses the Claims.

The entity that created the manifest could be different from the entity that signed it. Therefore, CoRIM can specify these entities and which actions each can perform.

An unsigned CoRIM endorses Claims by a method (such as a conveyance protocol or an enveloped data structure) that is not defined in this specification.

End of informative comment

5.5.1 Data Model Conventions

5.5.1.1 Data Model Notation

Start of informative comment

The Concise Reference Integrity Manifest (CoRIM) data model is defined in tabular notation (see Table 1) and in Unified Modeling Language (UML). The data model captures the semantics of the CoRIM schema while allowing adaptation to various data definition languages (DDL) and encoding formats. A CDDL definition of the CoRIM schema is found in [22]. An example CBOR encoding is found in APPENDIX A – ATTESTATION EXAMPLE.

Data model names use a dash character ('-') to separate names consisting of multiple strings. However, UML prohibits inclusion of the dash characters in class names. Therefore, an underscore ('_') is used instead.

The schema in tabular form in Table 1 captures several properties of a description grammar.

End of informative comment

NUMERIC KEY (NK)	STRING KEY (SK)	VALUE / TYPE	OPTIONAL (OPT)	MULTIPLICITY (MUL)	DESCRIPTION
Numeric identifier	String identifier	Type specifier or data	'T' is optional; otherwise, tuple is required.	'*' – Zero or more '+' – One or more	Additional information

Table 1: Notation for data model definition

Start of informative comment

Data model conventions follow a <*key*><*value*> tuple syntax.

The <*key*> portion of a tuple can be represented by a Numeric Key (NK) and/or String Key (SK). Depending on the data encoding, either the numeric key or the string key might be used in a <*key*><*value*> tuple.

The <*value*> portion of a tuple describes its value or type. Types may refer to other types that are recursively substituted to construct the <*value*>. Additionally, a <*value*> may refer to yet another <*key*><*value*> tuple, data type, boolean, or scalar.

The format for a statement in the schema (expressed here as a row in a table similar to Table 1) contains columns for keys (numeric and string), value, optionality, multiplicity, and description information.

The key columns are *Numeric* Key and *String Key*. They contain the <*key*> portion of the <*key*><*value*> pair. Binary encodings such as CBOR use the numeric keys and text encodings, such as JSON, use string keys.

The *Value Type* column describes the <*value*> portion of a given tuple. It contains a statement defined in another table in the schema or a type structure that is defined in the data types table (see 5.5.3.3). Note that braces ('{}') contain additional information to support correct encodings.

The *Optional* column in the table indicates whether a tuple is optional or required. The character 'T' in this column indicates the statement is optional, otherwise the statement is required.

 © TCG 2020

The *Multiplicity* column indicates whether a statement can occur multiple consecutive times. If a single instance of a statement is intended, then the multiplicity column is empty. If the statement can occur multiple times, this column contains either the '+' character (i.e., 1 or more) or the '*' character (i.e., zero or more).

The *Description* column contains normative statements corresponding to the *<key><value>* tuple.

End of informative comment

5.5.1.2 Typographical Notation

Start of informative comment

The tabular notation (see Table 1) uses a 'key' to 'value' (or 'key' to 'type') mapping convention where the key is expressed in both numeric and text representations. DDLs that define numeric code points, like CBOR, use the numeric key while those using text keys, like JSON use the string key. The 'type' describes the type or range of possible data allowed. Structure naming follows the typographical conventions contained in Table 2.

A 'type' is a data type or constant data value. A 'type choice' is an extensible data type where differently typed data can subsequently be added to the schema. A non-extensible 'type choice' is a data type definition that cannot be extended by the specification. A 'tagged type' is a 'type' that is prefaced with a well-known data type tag such as the Concise Binary Object Representation (CBOR) Tags in the IANA registry (See [27]). A 'map' is a collection of name-value pairs. A 'group' is set of name-value pairs that can be added to a map. A 'group choice' is an extensible group. A 'flags' structure enumerates a set of key names.

End of informative comment

TYPE TRAIT	CDDL EXAMPLE	FORMAT FOR VALUE
Type	`int`	*NAME*`-type`
Type choice (extensible)	`int / text`	`$`*NAME*`-type-choice`
Type choice (non-extensible)	`int / text`	*NAME*`-type-choice`
Tagged type	`#6.123(int)`	tagged-*NAME*-type
Map	`{ 1 => int, 2 => text }`	*NAME*`-map`
Group	`( 1 => int, 2 => text )`	*NAME*`-group`
Group choice	`( 1 => int // 2 => text )`	`$$`*NAME*`-group-choice`
Flags	`&(a: 1, b: 2)`	*NAME*`-flags`
Flags (extensible)	`&(a: 1, b: 2)`	*$NAME*`-flags`

Table 2: Typographical notation conventions

Start of informative comment

A typographical convention for key names prepends a namespace identifier to the string key with the '.' character as a separator. For example, given a namespace identifier of `corim` and a key name of device, a qualified key name would be `corim.device`.

End of informative comment

5.5.2 CoRIM Schema

Start of informative comment

The creator of a CoRIM constructs either an `unsigned-corim` or a `signed-corim` (see Figure 17). This section and its subsections describe the data model of the CoRIM schema. Later sections describe the data model for the manifest tags, either CoMID (see Section 5.5.3) or CoSWID (see Section 5.5.4).

An example of CDDL for CoRIM is in APPENDIX C – CORIM CDDL EXAMPLES, see also [22].

End of informative comment

5.5.2.1 concise-reference-integrity-manifest

The top-level CoRIM structure is described in Table 3.

NK	SK	Value Type	Opt	Mul	Description
0	corim	#6.500($concise-reference-integrity-manifest-type-choice)			Top-level CoRIM structure

Table 3: Format of concise-reference-integrity-manifest structure.

5.5.2.2 corim-map

The structure of an unsigned CoRIM is described in Table 4.

Start of informative comment

An unsigned CoRIM contains a manifest identifier, CoMID or CoSWID tags, and optionally references to services where additional related CoRIMs, certificates or other Endorsements can be obtained. The unsigned CoRIM also may identify profiles that this CoRIM satisfies. A profile specifies which of the optional parts of a CoRIM are required, which are prohibited, and which extension points [5] are exercised and how.

An unsigned CoRIM payload isn't integrity protected or authenticated. Integrity protection and authentication can be applied in a variety of ways. This specification defines one possibility using COSE signing (see Section 5.5.2.3).

Note that a CoRIM may contain tags that describe multiple unrelated modules.

End of informative comment

An unsigned CoRIM SHOULD NOT be trusted unless contained in a signed CoRIM or its integrity and authenticity has been established by some other means.

NK	SK	Value Type	Opt	Mul	Description
0	corim.id	$corim-id-type-choice			MUST identify a manifest instance
1	corim.tags	concise-tag-type-choice		+	One or more CoMID or CoSWID tag
2	corim.dependent-rims	corim-locator-map	T	+	An Internet service that supplies additional, possibly dependent, manifests or other related files
3	corim.profile	profile-type-choice	T	+	Profiles that if specified MUST be processed before processing any tags. If a profile isn't recognized the tags MUST NOT be processed.
4	corim.rim-validity	validity-map	T		The validity period of the RIM contents.
5	corim.entities	corim-entity-map	T	+	A list of entities involved in the CoRIM lifecycle.
{int}	{text}	$$corim-map-extension		*	Extension point for future expansion

Table 4: Format of corim-map structure.

5.5.2.2.1 corim-locator-map

The CoRIM locator structure is described in Table 5.

Start of informative comment

The CoRIM locator map contains pointers to repositories where dependent manifests, certificates or other information can be retrieved by a Verifier. The Verifier may have additional resources it checks to obtain the Endorsements it needs.

End of informative comment

The CoRIM SHOULD reference other manifests, certificates, or Endorsements upon which this CoRIM depends.

NK	SK	Value Type	Opt	Mul	Description
0	corim.href	uri-type			Pointer to a service that supplies dependent files or records
1	corim.thumbprint	hash-entry	T		Digest of the file or record referenced by `corim.href`.

Table 5: Format of corim-locator-map structure.

 © TCG 2020

5.5.2.3 signed-corim

The structure of a signed CoRIM is described in Table 6.

NK	SK	Value Type	Opt	Mul	Description
0	corim.cose-signature	#6.18(COSE-Sign1-corim)			An IANA global content tag is defined for COSE-Sign1-corim

Table 6: Format of signed-corim structure.

5.5.2.3.1 COSE-Sign1-corim

The CoRIM signature structure is described in Table 7.

NK	SK	Value Type	Opt	Mul	Description
N/A	protected	Bstr .cbor protected-corim-header-map			A CBOR encoded COSE header that is protected by the COSE signature
N/A	unprotected	unprotected-corim-header-map			A COSE header that is not protected by the COSE signature.
N/A	payload	bstr .cbor tagged-corim-map			A CBOR encoded tagged CoRIM
N/A	signature	bstr			A COSE signature block

Table 7: Format of COSE-Sign1-corim structure.

5.5.2.3.2 protected-corim-header-map

The protected CoRIM header structure is described in Table 8.

NK	SK	Value Type	Opt	Mul	Description
1	corim.alg-id	integer			Signature algorithm identifier (see [25])
3	corim.content-type	“application/rim+cbor”			MIMIE content type identifier
4	corim.issuer-key-id	bstr			Issuer key identifier
8	corim.meta-key	bstr .cbor corim-meta-map			CBOR encoded metadata associated with a signed CoRIM
{int}	cose-label {text}	cose-values {any}		*	Additional data in the COSE signature block

Table 8: Format of protected-corim-header-map structure.

5.5.2.3.3 unprotected-corim-header-map

The unprotected CoRIM header structure is described in Table 9.

NK	SK	Value Type	Opt	Mul	Description
{int}	cose-label {text}	cose-values {any}		*	Additional data in the COSE signature block

Table 9: Format of unprotected-corim-header-map structure.

5.5.2.3.4 corim-meta-map

The CoRIM meta map structure is described in Table 10.

Start of informative comment

The CoRIM meta map identifies the entity (or entities) that create and sign the CoRIM. This ensures the consumer of the CoRIM is able to identify credentials used to authenticate its creator / signer.

End of informative comment

 © TCG 2020

NK	SK	Value Type	Opt	Mul	Description
0	`corim.signer`	`corim-signer-map`		+	The entity (or entities) that creates and signs the CoRIM. The entity name SHOULD reference the manifest issuer's entity name as contained in the manifest issuer's signing credential (e.g., if the issuer credential is an X.509 certificate, it is the `subjectName`, `subjectAltName`, or `SubjectPublicKeyInfo`).
1	`corim.signature-validity`	`validity-map`	T		The validity period of a signed manifest

Table 10: Format of corim-meta-map structure.

5.5.2.3.5 validity-map

The CoRIM validity structure is described in Table 11.

NK	SK	Value Type	Opt	Mul	Description
0	`corim.not-before`	`time-type`	T		Beginning of the validity period
1	`corim.not-after`	`time-type`			End of the validity period

Table 11: Format of valitity-map structure.

5.5.2.3.6 corim-signer-map

The CoRIM signer structure is described in Table 12.

NK	SK	Value Type	Opt	Mul	Description
0	`corim.signer-name`	`$entity-name-type-choice`			The name of an organization that performs the signer role.
1	`corim.signer-uri`	`uri-type`	T		The registration identifier for the organization that manages the namespace for `corim.signer-name`.
{int}	{text}	`$$corim-signer-map-extension`		*	Extension point for future expansion

Table 12: Format of corim-signer-map structure.

5.5.2.3.7 corim-entity-map

The CoRIM entity structure is described in Table 13.

NK	SK	Value Type	Opt	Mul	Description
0	`corim.entity-name`	`$entity-name-type-choice`			The name of an organization that performs the signer role.
1	`corim.reg-id`	`uri-type`	T		The registration identifier for the organization that manages the namespace for `corim.entity-name`.
2	`corim.role`	`$corim-role-type-choice`			Manifest lifecycle role.
{int}	{text}	`$$corim-signer-map-extension`		*	Extension point for future expansion

Table 13: Format of corim-entity-map structure.

5.5.2.3.8 Signed CoRIM Requirements

The following requirements apply to a signed CoRIM structure:

1) A `signed-corim` SHALL identify its signer with a CoRIM meta map structure that has an optional validity period.
2) A `corim.signer` SHOULD identify the Endorser that creates the manifest.
3) A `corim.signer` MUST identify the Endorser that issues (signs) the manifest.
4) The `corim.signer` MUST be identified and SHOULD use a name that is statistically unique.
5) A `signed-corim` or `corim-map` SHOULD include `corim-locator-map` references to the CoRIMs that the current manifest depends upon.
6) A `corim-meta-map` SHOULD contain a `validity-map` that defines the manifest validity period.

 © TCG 2020

7) A `signed-corim` SHALL be invalid after the `corim.not-after` time is exceeded.
8) If the `validity-map` is omitted, the `signed-corim` SHALL be immediately valid.
9) If a CoRIM signing key is compromised, the signing certificate MUST be revoked.
10) A `signed-corim` SHALL be invalid if the key used to sign the CoRIM is revoked.

5.5.2.4 CoRIM Manifest Content Type and File Extension

The content type for a CoRIM manifest SHALL be "`application/rim+cbor`".

The filename extension for a file containing a CoRIM manifest SHALL be "`.corim`".

5.5.2.5 CoRIM Manifest URI/URL Fragments

Start of informative comment

A URI [28] or URL uses the following conventions to reference a tag in CoRIM manifests:

- The URI *authority* refers to a service where the manifest file is stored.
- The URI *path* identifies the manifest file `xyz.corim` where `xyz` is the name of the manifest filename and the filename extension is "`.corim`".
- The URI *fragment* is delineated by #*<tag-id>*, where *<tag-id>* identifies a tag within the manifest CoRIM and has the form `tag-id=`*<tag-id-value>,* and *<tag-id-value>* is an ASCII representation of the CoMID or CoSWID `tag-id` field.

An example of a `tag-id` field, the string "1234" is the value of the `tag-id`:

{ HYPERLINK https://my.manifest-server.com/my-manifests/manifestfilefile.corim#tag-id=1234 }

End of informative comment

5.5.3 A Schema for CoMID Tags

Start of informative comment

The CoRIM schema consists of two sub-schemas: CoMID and CoSWID. Parsers familiar with CoRIM are assumed to recognize CoMID and CoSWID schemas. However, some parsers may recognize only CoSWID or only CoMID schemas. Global CBOR code point tags are defined to allow pipelining to standalone parsers.

This specification reserves three global code point names: #6.500 for CoRIM, #6.501 for CoMID and #6.502 for signed CoMID. The CoSWID specification defines a global code point name for the CoSWID schema.

Numeric code points within a namespace restart at zero '0' for each defined map.

A CoMID tag is extensible in two ways: either a map structure is defined with an extension point or a value is an extensible type (e.g., `-type-choice`). Extensions to the CoMID schema may require an accompanying specification that defines code points, data types, and related information model semantics.

Vendor-specific code points may be used when extending the CoRIM or CoMid schema. Numeric code points may be globally assigned via IANA [27]. Vendor-specific schema extensions require a published information model that is generally available to the community of Verifier implementors.

An example of CDDL for CoMID is in APPENDIX C – CORIM CDDL EXAMPLES.

End of informative comment

5.5.3.1 concise-mid-tag-map

The CoMID tag structure is described in Table 14.

 © TCG 2020

NK	SK	Value Type	Opt	Mul	Description
0	comid.language	language-type	T		A textual language tag that conforms with IANA Language Subtag Registry [27]
1	comid.tag-identity	tag-identity-map			Tag identity attributes
2	comid.entities	entity-map	T	+	List of entities that contribute to the CoMID tag lifecycle (e.g., tag creation, registration, update)
3	comid.linked-tags	linked-tag-map	T	+	List of linked tags
4	comid.claims	triples-map	T		CoMID Claim set
{int}	{text}	$$concise-mid-tag-map-extension		*	Extension point for future expansion

Table 14: Format of concise-mid-tag-map structure.

5.5.3.1.1 tag-identity-map

The CoMID tag identity structure is described in Table 15.

NK	SK	Value Type	Opt	Mul	Description
0	comid.tag-id	$tag-id-type-choice			CoMID tag ID MUST be statistically unique
1	comid.tag-version	tag-version-type			CoMID tag version

Table 15: Format of tag-identity-map structure.

5.5.3.1.2 linked-tag-map

The CoMID linked tag structure is described in Table 16.

NK	SK	Value Type	Opt	Mul	Description
0	comid.linked-tag-id	$tag-id-type-choice			Tag ID of the linked tag; MAY be a CoMID or a CoSWID tag
1	comid.tag-rel	$tag-rel-type-choice			CoMID linked tag relationship (see Section 5.3.1)

Table 16: Format of linked-tag-map structure.

5.5.3.1.3 triples-map

The structure for a CoMID triples map is described in Table 17.

NK	SK	Value Type	Opt	Mul	Description
0	comid.reference-triples	reference-triple-record	T	+	If supplied, MUST contain one or more reference Claims
1	comid.endorsed-triples	endorsed-triple-record	T	+	If supplied, MUST contain one or more endorsed Claims
2	comid.attest-key-triples	attest-key-triple-record	T	+	If supplied, MUST contain one or more attestation key Claims
3	comid.identity-triples	identity-triple-record	T	+	If supplied, MUST contain one or more identity Claims
{int}	{text}	$$triples-map-extension		*	Extension point for future expansion

Table 17: Format of triples-map structure.

The `triples-map` MUST contain at least one entry.

5.5.3.1.4 version-map

The CoMID version structure is described in Table 18.

NK	SK	Value Type	Opt	Mul	Description
0	comid.version	version-type			Dot separated version string
1	comid.version-scheme	$version-scheme-type-choice	T		If supplied, MUST contain version string formatting rules. See [7].

Table 18: Format of version-map structure.

5.5.3.1.5 environment-map

The CoMID environment structure is described in Table 19.

NK	SK	Value Type	Opt	Mul	Description
0	comid.class	class-map	T		If supplied, MUST identify an environment class attributes
1	comid.instance	$instance-id-type-choice	T		If supplied, MUST identify an environment by instance id
2	comid.group	$group-id-type-choice	T		If supplied, MUST identify an environment by group id

Table 19: Format of environment-map structure.

5.5.3.1.6 reference-triple-record

The CoMID reference triple is described in Table 20.

NK	SK	Value Type	Opt	Mul	Description
0	comid.environment	environment-map			Identifies an Attester Target Environment. Attester Evidence MUST match the class or instance values of `environment-map`.
1	comid.measurement	measurement-map		+	Contains one or more reference measurements. Attester Evidence MUST match all measurements in `measurement-map`.

Table 20: Format of reference-triple-record structure.

5.5.3.1.7 endorsed-triple-record

The CoMID endorsed triple is described in Table 21.

NK	SK	Value Type	Opt	Mul	Description
0	comid.environment	environment-map			Identifies a Target Environment or Attesting Environment. The endorsed value Claims for `environment-map` MUST match an `environment-map` from a `reference-triple-record` of the Attester before it can be associated with the Attester. See Section 6.
1	comid.measurement	measurement-map		+	Contains one or more endorsed measurements.

Table 21: Format of endorsed-triple-record structure.

5.5.3.1.8 identity-triple-record

The CoMID identity triple is described in Table 22.

NK	SK	Value Type	Opt	Mul	Description
0	comid.environment	environment-map			Identifies an environment. The environment MUST prove it is the `environment-map` by authenticating with the key identified by comid.verification-key (e.g., by signing a challenge).
1	comid.verification-key	$crypto-key-type-choice		+	A set of keys where the entity identified by environment-map possesses the key material identified by $crypto-key-type-choice. See [29].

Table 22: Format of identity-triple-record structure.

5.5.3.1.9 attest-key-triple-record

The CoMID attestation key triple is described in Table 23.

NK	SK	Value Type	Opt	Mul	Description
0	comid.environment	environment-map			Identifies an Attester's Attesting Environment. The Attesting Environment MUST sign Evidence using the key identified by comid.verification-key.
1	comid.verification-key	$crypto-key-type-choice		+	A set of keys where the entity identified by environment-map possesses the key material identified by $crypto-key-type-choice.

Table 23: Format of attest-key-triple-record structure.

 © TCG 2020

5.5.3.1.10 class-map

The CoMID class map structure is described in Table 24.

The `class-map` MUST contain either the `comid.class-id` or both `comid.vendor` and `comid.model`.

The `class-map` SHOULD contain the `comid.class-id`.

NK	S Key	Value Type	Opt	Mul	Description (normative)
0	comid.class-id	$class-id-type-choice	T		The environment's class identifier.
1	comid.vendor	vendor-type	T		The environment's vendor, manufacturer, creator, etc.
2	comid.model	model-type	T		The environment's class identifier.
3	comid.layer	layer-type	T		Identifies the environment's sequence relative to other environments' layer value.
4	comid.index	index-type	T		Identifies clones of the same environment class.

Table 24: Format of class-map structure.

5.5.3.1.11 measurement-map

The CoMID measurement map structure is described in Table 25.

NK	SK	Value Type	Opt	Mul	Description
0	comid.mkey	$measured-element-type-choice	T		Key that identifies the type of element `measurement-values-map` represents: can be used to describe multiple elements that are measured as a single block.
1	comid.mval	measurement-values-map			Measurement values

Table 25: Format of measurements-map structure.

5.5.3.1.12 measurement-values-map

The CoMID measurement values map is described in Table 26.

The `measurement-values-map` MUST contain at least one measurement value.

NK	SK	Value Type	Opt	Mul	Description
0	comid.ver	version-map	T		A version number measurement value
1	comid.svn	$svn-type-choice	T		A security related version number measurement value
2	comid.digests	digests-type	T		A digest of the raw value measurement
3	comid.flags	flags-map	T		Operational modes that are made permanent at manufacturing time and do not change during normal operation of the Attester.
N/A	N/A	raw-value-group	T		A raw value measurement (See Section 5.5.3.1.14)
6	comid.mac-addr	$mac-addr-type-choice	T		EUI-48 or EUI-64 MAC address see [30]
7	comid.ip-addr	$ip-addr-type-choice	T		IPv4 or IPv6 address
8	comid.serial-number	serial-number-type	T		Serial number in ASCII
9	comid.ueid	ueid-type	T		Unique Enough ID
10	comid.uuid	uuid-type	T		Universally Unique ID
11	comid.name	text	T		Human readable information (e.g., model name)
{int}	{text}	$$measurement-values-map-extension		*	Extension point for future expansion

Table 26: Format of measurement-values-map structure.

5.5.3.1.13 Flags-map

Start of informative comment

The `flags-map` measurement describes Boolean operational modes.

If a `flags-map` value is not specified, then the operational mode is unknown.

End of informative comment

NK	SK	Value Type	Opt	Mul	Description
0	`comid.operational-flag-configured`	`boolean`	T		The Target Environment is fully configured for normal operation if the flag is true.
1	`comid.operational-flag-secure`	`boolean`	T		The Target Environment's configurable security settings are fully enabled if the flag is true.
2	`comid.operational-flag-recovery`	`boolean`	T		The Target Environment is NOT in a recovery state if the flag is true.
3	`comid.operational-flag-debug`	`boolean`	T		The Target Environment is in a debug enabled state if the flag is true.
4	`comid.operational-flag-replay-protected`	`boolean`	T		The Target Environment is protected from replay by a previous image that differs from the current image if the flag is true.
5	`comid.operational-flag-integrity-protected`	`boolean`	T		The Target Environment is protected from unauthorized update if the flag is true.
{int}	{text}	`$$flags-map-extension`		*	An extension point for future expansion.

Table 27: Format of flags-map structure.

5.5.3.1.14 raw-value-group

The raw value CoMID structure is described in Table 28.

NK	SK	Value Type	Opt	Mul	Description
4	`comid.raw-value`	`$raw-value-type-choice`			Bit positions in `raw-value-type` correspond to bit positions in `raw-value-mask-type`.
5	`comid.raw-value-mask`	`raw-value-mask-type`	T		The `raw-value-mask-type` bit MUST be `1` to evaluate the corresponding `raw-value-type` bit.

Table 28: Format of raw-value-group structure.

5.5.3.1.15 digests-type

The CoMID digest structure is described in Table *29*.

NK	SK	Value Type	Opt	Mul	Description
0	`comid.hash-entry`	`hash-entry`		+	A list of hash-entry values: used when multiple hash algorithms generate different digests for the same input data.

Table 29: Format of digests-type structure.

5.5.3.1.16 entity-map

The CoMID entity map structure is described in Table 30.

NK	SK	Value Type	Opt	Mul	Description
0	`comid.entity-name`	`$entity-name-type-choice`			The name of an organization that performs the role as defined in `$comid-role-type-choice`.
1	`comid.reg-id`	`uri-type`	T		The registration identifier for the organization that manages the namespace for `comid.entity-name`.
2	`comid.role`	`$comid-role-type-choice`		+	The list of CoMID entity roles. The entity that creates the `concise-mid-tag` SHOULD include a `comid.$module-role-type-choice` value of `comid.tag-creator`.
{int}	{text}	`$$entity-map-extension`		*	extension point for future expansion

Table 30: Format of entity-map structure.

 © TCG 2020

5.5.3.2 CoMID Linked Tag Relationships

Start of informative comment

This specification defines two linked tags relationships. Other relationships may be added by other specifications as needed. The `comid.supplements` relationship is especially useful for multi-tag modules and manifests. The `comid.replaces` relationship is useful for managing tag lifecycle.

CoMID tags can link to other CoMID tags or to CoSWID tags. The linking relationships differ for each.

Relationship types are defined by `$tag-rel-type-choice`.

End of informative comment

Relationship	Description
supplements	CoMID: The current CoMID tag contains Claims that augment the linked module's Claim set, identified by the linked tag. The linked tag module name MUST match the subject of the Claim in the current tag before the new Claims can be combined with the linked tag. CoSWID: N/A
replaces	CoMID: The current CoMID tag contains corrections to a previously issued tag. The current tag replaces the linked tag. The linked tag MAY be deleted. The tag-id of the current tag MUST remain unchanged from the tag-id of the linked tag. The tag-version of the current tag MUST be greater than the tag-version of the linked tag. CoSWID: N/A

Table 31: CoMID linked tag relationships.

5.5.3.3 CoRIM and CoMID Type Definitions

Start of informative comment

Type definitions describe the type of data expected. Types may also be constant data values.

The type definitions for CoRIM and CoMID are in Table 32. The types provided in Table 32 are provided as guidance; additional types may also be defined.

Table 32 substitutes *Type Name* with *Type Value* recursively until it reaches a termination point. In some cases, there are multiple potential terminating Type Values. The *Choice* column indicates that a choice of multiple Type Values is expected. Only one of the possible choices is selected when Choice is true. The 'C' character in the Choice column indicates Choice is true. Additionally, the Type Value column may contain hints for data models that focus on data encoding and representation.

End of informative comment

Type Name	Type Value	Choice	Description

`$class-id-type-choice`	`tagged-oid-type`
	`tagged-uuid-type`
	`tagged-int-type`
		C	The value selected MUST be used in Evidence.

`$comid.module-role-type-choice`	`comid.tag-creator`
	`comid.creator`
	`comid.maintainer`
		C	CoMID entity role types

`comid.tag-creator`	`Constant : 1`		Creator of the CoMID tag
`comid.creator`	`Constant : 2`		Creator of the module
`comid.maintainer`	`Constant : 3`		Person or organization making changes to a module, used when different from the module creator.

 © TCG 2020

Type Name	Type Value	Choice	Description
`$version-scheme-type-choice`	`$coswid.version-scheme`		Note that CoMID reuses the CoSWID defined version-scheme; See [7]
`$concise-reference-integrity-manifest-type-choice`	`tagged-corim-map` `#6.502(signed-corim)`	C	An IANA global content tag that identifies one of `corim-map` or `signed-corim`
`$concise-tag-type-choice`	`corim.concise-mid-tag` `corim.concise-swid-tag`	C	CoRIM tag types
`corim.concise-mid-tag`	`#6.506(concise-mid-tag-map)`		`concise-mid-tag-map` with IANA global identifier code point
`corim.concise-swid-tag`	`#6.505(concise-swid-tag-map)`		`concise-swid-tag-map` with IANA global identifier code point
`$corim-id-type-choice`	`text` `uuid-type`	C	A manifest identifier; SHOULD use UUID
`$corim-role-type-choice`	`corim.manifest-creator`	C	Roles related to manifest lifecycle
`corim.manifest-creator`	`1`		Creator of the CoRIM
`$entity-name-type-choice`	`text`	C	Entities, such as supply chain entities
`$group-id-type-choice`	`tagged-uuid-type`	C	A group identifier
`index-type`	`integer`		A clone environment item number
`$instance-id-type-choice`	`tagged-ueid-type` `tagged-uuid-type`	C	The value selected MUST be used to construct Evidence.
`$ip-addr-type-choice`	`ip4-addr-type` `ip6-addr-type`	C	IPv4 or IPv6 address types
`ip4-addr-type`	`bytes .size 4`		IPv4 address, see [30]
`ip6-addr-type`	`bytes .size 6`		IPv6 address, see [31]
`layer-type`	`integer`		An environment layering sequence number
`$mac-addr-type-choice`	`mac48-addr-type` `mac64-addr-type`	C	MAC address types
`mac48-addr-type`	`bytes .size 6`		EUI-48 MAC address, see [32]
`mac64-addr-type`	`bytes .size 8`		EUI-64 MAC address, see [32]
`min-svn`	`svn-type`		A minimum security version number
`model-type`	`text`		An environment class identifier string

`$measured-element-type-choice`	`tagged-oid-type`
	`tagged-uuid-type`
	`uint`
		C	The type of identifier used to identify the measurement contained in comid.mval.

`oid-type`	`bytes`		Object Identifier, see [4]
`$crypto-key-type-choice`	`tagged-pkix-base64-key-type`		A tagged structure containing a base64 encoded pkix public key.
	`tagged-pkix-base64-cert-type`		A tagged structure containing a base64 encoded pkix certificate that contains a public key.
	`tagged-pkix-base64-cert-path-type`		A tagged structure containing a base64 encoded pkix certificate path that contains an end entity public key certificate and other certificates in the certificate path.
`tagged-pkix-base64-key-type`	`#6.554(tstr)`		A raw key in DER format base64 encoded. See [29]

 © TCG 2020

Type Name	Type Value	Choice	Description
tagged-pkix-base64-cert-type	#6.555(tstr)		An X.509 certificate in DER format base64 encoded. See [33]
tagged-pkix-base64-cert-path-type	#6.556(tstr)		An X.509 certificate path in DER format base64 encoded.
profile-type-choice	uri-type tagged-oid-type	C	CoRIM profile identifier: choice of URI or OID
$raw-value-type-choice	raw-value-type	C	The extensible type choice that can be bytes, text or yet to be defined tagged values
raw-value-type	#6.560(bytes)		Bit array
raw-value-mask-type	bytes		Bit array
serial-number-type	text		A product serial number
svn	svn-type		A security version number
svn-type	unsigned integer		A monotonically increasing value
$svn-type-choice	tagged-svn tagged-min-svn	C	A tagged security version number
$tag-id-type-choice	text uuid-type	C	SHOULD use a UUID
$tag-rel-type-choice	comid-supplements comid-replaces	C	See also Section 5.3.1
comid.supplements	Constant : 0		A CoMID linked tags relationship type
comid.replaces	Constant : 1		A CoMID linked tags relationship type
tag-version-type	integer .default 0		A version string for CoMID tags
tagged-corim-map	#6.501(corim-map)		A corim-map with IANA global identifier code point
tagged-int-type	#6.551(int)		Numeric identifier with IANA global identifier code point
tagged-min-svn	#6.553(min-svn)		An IANA global content tag that identifies a security version number that is evaluated with greater than or equals semantics
tagged-oid-type	#6.111(oid-type)		OID with IANA global code point
tagged-svn	#6.552(svn)		An IANA global content tag that identifies a security version number that is evaluated with equivalence semantics
tagged-euid-type	#6.550(ueid-type)		UEID with IANA global identifier code point
tagged-uuid-type	#6.37(uuid-type)		UUID with IANA global identifier code point
tagged-xcorim-map	#6.526(xcorim-map)		A xcorim-map with IANA global identifier code point
time-type	time		Coordinated Universal Time
ueid-type	ueid		Unique Enough ID, see [34]
uri-type	URI		Universal Resource Identifier, see [12]
uuid-type	bytes .size 16		Universally Unique ID, see [35]
vendor-type	text		Vendor name or identifier
version-type	text .default ‘0.0.0’		A version string that MAY follow a version string encoding scheme (see coswid.$version-scheme).

Table 32: CoRIM and CoMID type definitions.

 © TCG 2020

5.5.4 A Schema for CoSWID Tags

Start of informative comment

This section describes normative requirements and extensions to CoSWID that are unique to CoRIM. The Reader is directed to existing specifications that describe normative SWID Tag [11] and CoSWID [7] syntax and semantics. Additionally, Figure 20 contains a simplified UML description of CoSWID.

End of informative comment

5.5.4.1 concise-swid-tag

The CoSWID tag structure is described in Table 33.

NK	SK	Value Type	Opt	Mul	Description
	`coswid.tag-id`	`$tag-id-type-choice`			A value that identifies a tag.
	`coswid.software-name`	`text`			A value that identifies a software component.
	`coswid.software-version`	`text`			A version string that identifies the software version.
	`coswid.version-scheme`	`$version-scheme`			A version string formatting convention.
	`coswid.software-meta`	`software-meta-entry`			A map containing additional information about the software tag.

Table 33: Format of concice-swid-tag CoSWID structure.

5.5.4.2 software-meta-entry

The software meta map structure is described in Table 34.

NK	SK	Value Type	Opt	Mul	Description
	`coswid.persistent-id`	`text`	T		MUST be a statistically unique identifier that identifies the software.
	`coswid.product`	`text`	T		A software component product name.

Table 34: Format of software-meta-entry CoSWID structure.

5.5.4.3 entity-entry

The CoSWID entity name structure is described in Table 35.

NK	SK	Value Type	Opt	Mul	Description
	`coswid.entity-name`	`text`			The name of an organization that performs the role as defined in `$role-type-choice`.
	`coswid.role`	`$role-type-choice`		+	The name of an entity role. The entity that creates the `concise-swid-tag` SHOULD include a role value of `coswid.tag-creator`.

Table 35: Format of entity-entry CoSWID structure.

5.5.4.4 link-entry

The CoSWID link tag relationship type structure is described in Table 36.

NK	SK	Value Type	Opt	Mul	Description
	`coswid.rel`	`$rel`			Linked tag relationship types for both CoSWID-to-CoSWID and CoSWID-to-CoMID linked tags.

Table 36: Format of link-entry CoSWID structure.

5.5.4.5 Tag Identification

The `coswid.tag-id` field in `concise-swid-tag` uniquely identifies a tag. The `tag-id` SHOULD be statistically unique. The `corim.signer` is the entity that vouches for the process that produces a unique `coswid.tag-id` value.

CoSWID `concise-swid-tag.tag-id` SHOULD use the same `tag-id` format as is used by CoMID `concise-mid-tag-map.tag-identity-map.tag-id`.

 © TCG 2020

5.5.4.6 CoSWID Linked Tags Relationships

Start of informative comment

CoSWID tags can link to other CoSWID tags or to CoMID tags. The linking relationships differ for each. CoSWID to CoSWID linked tags relationships are defined using the `link-entry` map. CoSWID to CoMID linked tags relationships use the CoSWID relationships in the `$rel` type choice.

End of informative comment

5.5.4.7 CoSWID Claims

The `concise-swid-tag` map contains Endorsements and Evidence. This specification defines Endorsements only. Therefore, only `payload-entry` maps are recognized by this schema.

CoSWID payloads (`payload-entry` map) are normally interpreted as Reference Value Claims.

CoSWID payloads might contain additional information that does not have matching Reference Value Claims. This information is interpreted as Endorsed Value Claims. Endorsed Value Claims are accepted as definitive because of the Verifier's trust in the Endorser. Nevertheless, Verifiers MUST establish that Endorsed Value Claims pertain to the Attester. Typically, this is accomplished by validating Evidence according to the Reference Value Claims, then processing the Endorsed Value Claims.

Evidence encoding formats need to align with Reference Claims formats to ensure proper verification. CoSWID can be used as an Evidence encoding format when software about which the Verifier requires Evidence is installed on the Attester.

A CoSWID `evidence-entry` map MUST NOT be used to describe CoRIM Claims.

A CoSWID `payload-entry` map SHOULD be used to construct Endorsements Claims and MAY contain both Endorsed and Reference Values Claims.

CoMID links to CoSWID tags MUST use a link relationship of `comid.supplements`.

5.5.4.8 CoRIM Locator using CoSWID

Start of informative comment

The CoSWID `link-entry` map contains URI references to providers of CoRIM and possibly other manifests. The `link-entry` map contains an href to the entity supplying manifests for linked tags. The href identifies both the manifest provider and the `tag-id` of the linked tag. The href requestor verifies that the `tag-id` in the returned manifest matches the `tag-id` in the request. A tag `thumbprint` digest contained in the `link-entry` map may be used to verify that the expected tag contents have not been manipulated.

A Verifier uses CoRIM to reconstruct the composition of a multi-tag device. Attesters can provide URI references to CoRIM providers for use by Verifiers to aid in the reconstruction.

End of informative comment

Manifests containing linked CoSWID tags SHOULD have a `corim-locator-map` that contains a `corim.href` to a provider of the manifest containing the linked CoSWID tag.

Manifests containing CoSWID tags with linked CoMID tags SHOULD have a link-entry map that contains a `corim.href` to a provider of the manifest containing the linked CoMID tag. The `link-entry` relationship (`$rel`) SHOULD be `supplemental`.

A `link-entry` map SHOULD include a `thumbprint` digest of the linked tag contents.

 © TCG 2020

5.6 Xcorim Data Model for Deny Lists

Start of informative comment

This section contains normative requirements for a CoRIM deny list called *Xcorim.* A UML description is found in Figure 19 and Figure 21. If a CoRIM manifest is determined to be invalid and the `validity-entry` has not yet expired, a deny list can be used to invalidate that CoRIM manifest. An Xcorim deny list consists of the entity that originally issued the manifest and a list of CoRIM identifiers (e.g., `corim.id`) to be invalidated.

Typically, the same entity that issues the CoRIM will issue the Xcorim.

The integrity of the deny list is established via digital signature.

A service entity maintains the set of deny lists for use by the Verifier community.

Once a `deny-id` is placed on a deny list and distributed, the denial is considered permanent.

End of informative comment

5.6.1 Xcorim Signature and Map Structures

5.6.1.1 Xcorim

The top-level Xcorim structure is described in Table 37.

NK	SK	Value Type	Opt	Mul	Description
`0`	`xcorim`	`#6.525($corim-revocation-type-choice)`			The top-level Xcorim structure that is either a signed-xcorim or an xcorim-map

Table 37: Format of a corim revocation structure.

5.6.1.2 signed-xcorim

The signed Xcorim structure is described in Table 38.

NK	SK	Value Type	Opt	Mul	Description
`0`	`xcorim.cose-signature`	`#6.18(COSE-Sign1-xcorim)`			IANA defines a global content tag for `COSE-Sign1-xcorim`

Table 38: Format of signed-xcorim Xcorim structure.

5.6.1.3 COSE-Sign1-xcorim

The Xcorim signature structure is described in Table 39.

NK	SK	Value Type	Opt	Mul	Description
`N/A`	`protected`	`bstr .cbor protected-xcorim-header-map`			A CBOR encoded COSE header that is protected by the COSE signature
`N/A`	`unprotected`	`unprotected-xcorim-header-map`			A COSE header that is not protected by the COSE signature
`N/A`	`payload`	`bstr .cbor tagged-xcorim-map`			A CBOR encoded tagged Xcorim
`N/A`	`signature`	`bstr`			A COSE signature block

Table 39: Format of COSE-Sign1-xcorim structure.

5.6.1.4 protected-xcorim-header-map

The Xcorim protected header structure is described in Table 40.

NK	SK	Value Type	Opt	Mul	Description
`1`	`xcorim.alg-id`	`integer`			signature algorithm identifier
`3`	`xcorim.content-type`	`“application/xrim+cbor”`			MIMIE content type identifier
`4`	`xcorim.issuer-key-id`	`bstr`			Issuer key identifier

 © TCG 2020

NK	SK	Value Type	Opt	Mul	Description
9	`xcorim.meta-key`	`bstr .cbor xcorim-meta-map`			Metadata associated with a signed Xcorim
{int}	`cose-label` {text}	`cose-values` {any}		*	Additional information contained in the COSE signature block

Table 40: Format of protected-xcorim-header-map structure.

5.6.1.5 unprotected-xcorim-header-map

The Xcorim unprotected header structure is described in Table 41.

NK	SK	Value Type	Opt	Mul	Description
{ int }	`cose-label` { text }	`cose-values` { any }		*	Additional information contained in the COSE signature block

Table 41: Format of unprotected-xcorim-header-map structure.

5.6.1.6 xcorim-meta-map

The Xcorim meta map structure is described in Table 42.

NK	SK	Value Type	Opt	Mul	Description
0	`xcorim.signer`	`xcorim-signer-map`			The entity that signs the Xcorim. The entity name SHOULD reference the issuer's entity name as contained in the issuer's signing credential, e.g. the subjectName, subjectAltName, or SubjectPublicKeyInfo field of a X.509 certificate. The entity that issues the `signed-xcorim` MUST be identified by `xcorim.signer-name` in `xcorim-signer-map`.
1	`xcorim.timestamp`	`corim.time-type`			The date and time the deny list was issued.

Table 42: Format of xcorim-meta-map structure.

5.6.1.7 xcorim-map

The Xcorim entity map structure is described in Table 43.

NK	SK	Value Type	Opt	Mul	Description
0	`xcorim.entity`	`xcorim-entity-map`	T	+	The entity that constructs the unsigned Xcorim. The entity in `xcorim-entity-map` MUST have a type of `xcorim.deny-list-creator` for CoRIM revocation.
1	`xcorim.deny-list`	`$corim-id-type-choice`		+	An array of `$corim-id-type-choice` values.
{ int }	{ text }	`xcorim.deny-list-map-extension`		*	extension point for future expansion

Table 43: Format of xcorim-map structure.

5.6.1.8 xcorim-entity-map

The Xcorim entity name structure is described in Table 44.

NK	SK	Value Type	Opt	Mul	Description
0	`xcorim.entity-name`	`$entity-name-type-choice`			The name of an organization that performs the role defined in `$corim-role-type-choice`.
1	`xcorim.reg-id`	`uri-type`	T		The registration entity that manages the namespace for `xcorim.entity-name`.
2	`xcorim.role`	`$xcorim-role-type-choice`			The Xcorim entity role
{ int }	{ text }	`$$xcorim-entity-map-extension`		*	Extension point for future expansion

Table 44: Format of xcorim-entity-map structure.

5.6.1.9 xcorim-signer-map

The Xcorim signer structure is described in Table 45.

© TCG 2020

NK	SK	Value Type	Opt	Mul	Description
0	xcorim.signer-name	$entity-name-type-choice			The name of an organization that performs the role defined in `$corim-role-type-choice`.
1	xcorim.signer-uri	uri-type	T		URI reference to the entity that manages the namespace for `xcorim.signer-name`.
{ int }	{ text }	$$xcorim-signer-map-extension		*	Extension point for future expansion

Table 45: Format of xcorim-signer-map structure.

5.6.2 Xcorim Data Types

Start of informative comment

This section contains Xcorim type definitions in Table 46. These types are not already defined by CoRIM type definitions (see Table 32). See also Figure 19.

End of informative comment

Type Name	Type Value	Choice	Description
$corim-revocation-type-choice	tagged-xcorim-map #6.527(signed-xcorim)	C	An IANA global content tag identifying either: `xcorim-map`, or `signed-xcorim`
$entity-name-type-choice	text	C	Entities, such as supply chain entities, that de-assert the trustworthiness of claims contained in this Xcorim.
$xcorim-role-type-choice	xcorim.deny-list-creator	C	Xcorim roles
xcorim.deny-list-creator	1		The entity that created the xcorim.

Table 46: Xcorim data types.

5.6.3 Xcorim Content Type and File Extension

The content type for a CoRIM deny list SHALL be "`application/xrim+cbor`".

The filename extension for a file containing a CoRIM deny list SHALL be "`.xcorim`".

5.7 Certificate Considerations

Start of informative comment

The creator of CoRIM and Xcorim documents may use X.509v3 certificates to certify the signing key. Certificates are another form of Endorsements message that can be conveyed from an Endorser to a Verifier. An Endorser might issue manifests and device certificates. Manifests and device identities may be signed by an Endorser with a certificate. The certificate path may also be included in the Endorsements. A Verifier might trust certificates based on a trust anchor policy that recognizes the root CA as trusted. The Verifier needs to be able to determine which certificate is authorized for signing Endorsement Manifests vs. other types of data.

The X.509v3 certificate contains an extended key usage extension that indicates the key is used for signing manifests and manifest revocations (see [14]). The certificate key usage extension is expected to specify a *digitalSignature* usage.

Additional extended key usage extension object identifiers for manifest signing are defined below.

End of informative comment

```
tcg-dice-kp-manifestSign OBJECT IDENTIFIER ::= {tcg-dice-kp 13}
```

The `tcg-dice-kp-manifestSign` object identifier MUST be included as a KeyPurposeId in the Extended Key Usage extension (see [33]) in certificates used to sign CoRIM documents.

 © TCG 2020

```
tcg-dice-kp-manifestRevoke OBJECT IDENTIFIER ::= {tcg-dice-kp 14}
```

The `tcg-dice-kp-manifestRevoke` object identifier MUST be included as a KeyPurposeId in the Extended Key Usage extension in certificates used to sign Xcorim documents.

 © TCG 2020

6 PROCESSING ENDORSEMENT CLAIMS

Start of informative comment

This section describes the steps needed to process Endorsement Manifest Claims. Processing is divided into five steps (See Figure 14):

1) Provisioning – Installs appropriate default and Verifier Owner (represented in Figure 14 as “Owner”) authorized trust anchors used to verify Endorsement Manifests, certificates, or other information obtained from supply chain entities. Provisioning also installs other policies used during the appraisal of attestation Evidence.
2) Endorsements Discovery – Locates, downloads, and caches manifests, certificates, or other information used during Evidence appraisal.
3) Authentication – Authenticates the Attester endpoint which may include establishing a communication channel and a security context. Typically, authentication requires use of module identity credentials such as a device identity certificate.
4) Evidence collection and appraisal – Obtain fresh attestation Evidence from the attester. The Attester may be a multi-module device and may have multiple sets of Evidence that involve repeated matching and appraisal. Evidence Claims are matched against Reference Claims and associated with Endorsed Claims that may be necessary to properly apply an Appraisal Policy for Evidence. Evidence may also contain instance data (i.e., un-authenticated unique identifiers) that could be matched with instance Claims as directed by policy.
5) Attestation results propagation (not shown in Figure 14) – Attestation results are created and distributed to interested relying parties. Relying parties show interest in Attestation Results in a variety of ways that are out of scope for this specification.

In this example, (see Figure 14) each attestation role is performed by a separate entity. Actual deployments may combine multiple roles on the same entity or distribute a role across multiple entities. Although there will likely be a variety of attestation Verifiers ranging from special purpose, embedded, and generalized, this example assumes a generalized Verifier. Verifiers might, for example, operate as a service or as an installed application.

End of informative comment

 © TCG 2020

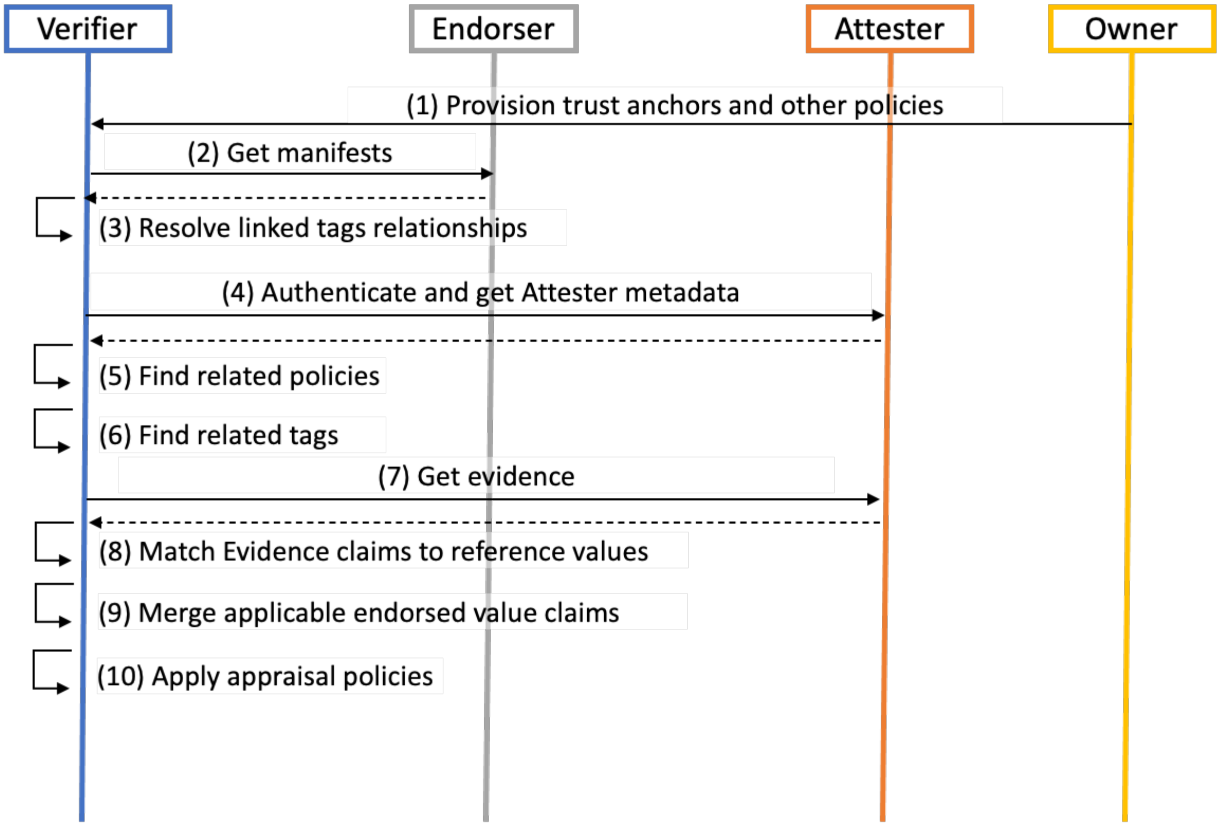

Figure 14: Processing Endorsements and Reference Values flow

6.1 Verifier Provisioning

Start of informative comment

This section describes steps (1), (2), and (3) in Figure 14.

A generalized Verifier can interoperate with many supply chain Endorsers by being configured or provisioned with multiple trust anchors as needed to reflect supply chain dynamics. Endorsement Manifests, manifest deny lists, certificates or other information are presumed to be digitally signed using asymmetric keys that are either certified by public key infrastructure with certificates that chain to one of the configured trust anchors or have individual signing keys that anchor signature verification operations. Manifest deny lists should be checked in addition to checking manifest signatures.

Verifiers may benefit from other policies that facilitate Evidence appraisal. Endorsements are assertions from supply chain entities that vouch for the integrity of modules as manufactured, tested and supplied. However, a Verifier Owner could apply admission control, asset tracking, auditing or other management function that affects appraisal results. This specification doesn't define an Appraisal Policy for Evidence structure or semantics. This specification presumes that an Appraisal Policy for Evidence is provisioned at the start of Endorsements and Evidence evaluation: therefore, the potential impact of updated policy is not considered. The provisioning step is performed by the Verifier Owner.

The supply chain ecosystem may be multi-faceted and dynamic such that multiple manifests and certificates are needed to bring together a complete picture of the device or platform. As such, supply chain services support generalized verifiers. Supply chain services can facilitate discovery, delivery and caching of manifests and certificates. Generalized Verifiers may need the provisioning or configuration of support services as a prerequisite to appraisal. This specification does not define standardized supply chain support services.

 © TCG 2020

Supply chain Endorsers might support attestation throughout a device's lifecycle resulting in changes to the manifest or certificates after original manufacture. Additionally, manifest and certificate creation and issuance during manufacturing may occur at different times and in different locations making it challenging to coordinate a monolithic approach to creation and issuance.

End of informative comment

6.2 Attester Authentication

Start of informative comment

This section describes steps (4) and (5) in Figure 14.

The Verifier authenticates the Attester and may establish a secure channel to ensure attestation Evidence comes from the Attester. The Attester may consist of multiple components that each might consist of multiple Attesting and/or Target Environments. The device authentication credentials will have layer or device scope, while a composition of modules may have multiple Attesting Environments, each with distinct attestation keys. Consequently, the Verifier expects the device authentication credentials to be securely bound to the attestation credentials. In a DICE layering endpoint, at least one of the layers contains a device authentication credential. The other layers may contain additional attestation credentials. DICE layers are cryptographically related using Compound Device Identifier (CDI) values that are layer specific. In a composite device model, there may be multiple Target Environments tied to a common Attesting Environment. Each Target Environment might report distinct Evidence protected using a common attestation key or might use a different attestation key for each Target Environment.

Attester policy determines Attester endpoint composition and key hierarchy. The Verifier depends on the Endorsement manifest for endpoint composition details that it uses to validate composition details supplied with attestation Evidence. In some cases, attestation Evidence may include endpoint composition details that might assist the Verifier in obtaining the correct Endorsement manifest and tags. However, all information obtained from the Attester is by default presumed to be suspicious until proven otherwise.

Device authentication credentials may contain device information that helps narrow the field of possible related Endorsement manifests, tags and policies. A Verifier might realize scalability improvements by focusing its search for related attestation Endorsements to the current attesting device.

End of informative comment

6.3 Resolution of Linked Tags

Start of informative comment

This section describes step (6) in Figure 14.

Attester devices might require multiple Endorsement tags to comprehensively describe endpoint composition. An Endorsement tag can describe a top-level device that corresponds to the device authentication credential. The device credential can be correlated with the device Endorsement tag using the following common techniques:

- Include device class identifiers in the device ID credential, such as vendor, model, and class-id that are already included in the top-level endorsement tag.
- Include a reference to the top-level Endorsement tag in the device ID credential.
- Include a reference to the device ID credential in the top-level Endorsement tag.
- Include a device identity Claim in the top-level Endorsement tag.
- Include Endorsements in the linked tag that matches a reference measurement that is the expected source of Claims in the linked tag.

Multi-module device compositions rely on multiple tags each containing module specific Claims. Different types of Claims may pertain to a common module, for example, different supply chain Endorsers may mint separate tags

 © TCG 2020

for reference, endorsed, instance and identity Claims. The union of these represents the complete set of Claims about the module.

Multiple Endorsers could cooperate to assert multiple Claims of the same type. For example, two suppliers may develop reference Claims derived from a common bill of materials. The ‘supplements’ link relationship can be used to model both situations.

Multi-module device compositions rely on linked tags to model device composition involving either singleton or multiple Endorsers.

Device lifecycle considerations could result in tags that either update, patch or replace existing tags.

The above considerations may require resolving linked tags in order to identify the most relevant Claims when matched with Evidence and when applying appraisal policies.

End of informative comment

6.3.1 Algorithm for Linked Tag Resolution

Start of informative comment

This section describes a possible algorithm for resolving linked tags that results in an accepted set of Claims for a given device or component. Claims are “accepted” in that they are received by the Verifier and determined to be relevant to the Appraisal process. The set of accepted Claims is a subset of the set of all Claims known to the Verifier. The set of all known Claims is referred to as the `CLAIMSET`. The algorithm could be applied once for a large set of possible compositions, or it could be integrated with Evidence matching to better optimize algorithm performance.

Step 0:

The algorithm presumes the top-level Endorsement tag `TCURR` has been successfully identified and placed into the `CLAIMSET`. If `TCURR` has no linked tags entries then processing stops, otherwise the next unprocessed tag is selected. The selected tag is named `TLINK`. `TCURR` has a link relationship with `TLINK` that determines how `TCURR` processes `TLINK`.

Step 1:

IF link relationship is ‘replaces’ AND

`TCURR.tag-id` = `TLINK.tag-id` *AND*

`TCURR.tag-version` > `TLINK.tag-version`

THEN

`TCURR` *replaces* `TLINK` *in* `CLAIMSET`

`TLINK` *may be audited*

`TLINK` *may be deleted*

Step 2:

IF link relationship is `supplements` *AND* `TLINK.tag-id != TCURR.tag-id`

THEN

Add `TLINK` *to* `CLAIMSET`

End of informative comment

 © TCG 2020

6.4 Evidence Processing

Start of informative comment

This section describes steps (7), (8), (9), and (10) in Figure 14.Section

Section 6.4.1 contains an algorithm for matching a set of related modules and their associated Evidence Claims, i.e., the `CLAIMSET`. CoMID Claims can describe compositional relationships that are security relevant, for example a device with DICE layering may have an upper layer that depends on a lower layer. Evidence could describe a particular layering, but the Endorser might describe an expected layering. A Verifier might rely on Endorsements for composition detail rather than inferring composition through Evidence as a strategy for avoiding replacement attacks.

The Verifier starts with a set of accepted Claims that are initialized from Evidence, referred to as `ACCEPTED_CLAIMS`. Accepted Claims are a subset of the full Claim set and include Reference and Endorsed Values, and device identity Claims that the Verifier expects to find in Evidence from the Attester. The Verifier processes one or more tags from the Claim set that might match the Evidence. For each tag, the Verifier compares Reference Claims in `CLAIMSET` against Evidence in `ACCEPTED_CLAIMS`. If all Reference Claims match `ACCEPTED_CLAIMS`, then the Evidence matches, and all values in the `CLAIMSET` are added to the `ACCEPTED_CLAIMS`.

Evidence is obtained from an authenticated Attester and evaluated to find matching endorsed Claims contained in the `CLAIMSET`. `ACCEPTED_CLAIMS` are appraised according to an Appraisal Policy for Evidence. Other Claims, such as Endorsed Claims, are included in the `ACCEPTED_CLAIMS` if they share a common `claims-map` subject.

The `ACCEPTED_CLAIMS` are appraised by an Appraisal Policy for Evidence to determine Attestation Results. For example, if an Appraisal Policy for Evidence expects a set of Claims that is not a subset of the `ACCEPTED_CLAIMS`, then the Attestation Results might be the empty set.

Compare operations are applied at the attribute level for each Claim (e.g., ‘vendor’, ‘class-id’, ‘raw-value’ etc…). The endianness for CoRIM data is big endian. The endianness for attribute data is expected to be big endian unless endianness is specified by a related standard. CBOR encoded data is either big endian or does not have endianness constraints.

End of informative comment

6.4.1 Evidence Matching Algorithm

Start of informative comment

This section contains an algorithm (See also Figure 15) for matching Claims contained in Evidence to Endorsed Claims contained in `ACCEPTED_CLAIMS`. Evidence is expected to be cryptographically bound to the currently authenticated Attester. A cryptographic binding could be in the form of a digital signature by an attestation key that is certified by the device identity key.

Step 0:

The `CLAIMSET` is a collection of structures (consisting of *environment-measurement* associations).

There may be four maps within `CLAIMSET` that correspond to the four types of Claims triples (reference, endorsed, identity, and attestation key).

The *environment* fields that name a module class include `vendor`, `class-id`, `model`, `layer`, and `index`.

Note that `class-id` in CoMID `class-map` is matched with `type` in the `tcbinfo` Evidence extension in [14]. If `class-id` is a `$class-id-type-choice` of the form `tagged-uuid-type` or `tagged-oid-type` then the

 © TCG 2020

IANA global code point is not included. If the Evidence creator uses both forms such that there is a possibility of namespace collisions, the Vendor or model fields can be used to avoid namespace collisions.

If `comid.mkey` is used, then match `comid.mkey` in Evidence with `comid.mkey` in Reference Values.

The *measurement* fields include `digest`, `raw-value`, `version`, `svn`, `flags` and other measurements.

The `raw-value` in CoMID `class-map` is synonymous with `vendorinfo` in the `tcbinfo` Evidence extension in [14].

If the *measurement* fields contain instance measurements, the *environment* fields may be described by an instance identifier `uuid` or `ueid`.

For `uuid` or `ueid` in CoMID `$instance-id-type-choice`, the `type` field in the `tcbinfo` Evidence extension contains the UUID. Otherwise, `ueid` is contained in the `TcbUeid` Evidence extension. When comparing Evidence with CoMID, the IANA global code point is ignored.

The `ACCEPTED_CLAIMS` is initialized to empty.

This algorithm assumes the Verifier has a method for walking the Evidence and correlating it with the `CLAIMSET`.

The Claims in Evidence are called C_{EV}.

The Claims in `CLAIMSET` may contain reference Claims called C_{REF}, endorsed Claims called C_{END}, and identity Claims called C_{ID}.

Step 1:

Check that, for a given Claim C_X in Evidence C_{EV}, there isn't another Claim C_Y in Evidence with the same environment but different measurements, otherwise fail.

If evidence contains an element key then check that the `comid.mkey` in evidence has the same value as the corresponding `comid.mkey` in reference values (endorsements).

Step 2:

The Verifier has Evidence that it has obtained from the Attester device. Each piece of Evidence C_{EV} contains a collection of Claims.

For each Claim C_X in C_{REF} and C_Y in C_{EV}:

If the environment of C_Y matches any C_X environment in C_{REF} then assign C_Y to C_Z.

If there is no matching Claim (C_Z) then add C_Y to `PENDING_CLAIMS` and get the next C_Z.

If C_Z has a `layer` value that is greater than the `layer` value for any already `ACCEPTED_CLAIMS`, and it has the same environment (i.e., `class-map` or `instance-id`) and, if used, the same element (i.e., `comid.mkey`), and different or down-revision measurements (i.e., `measurement-values-map`), then fail Claims verification.

Compare the measurement values of C_X and C_Z.

If C_X contains `raw-value` and C_Z contains `raw-value` with a different binary value after applying the `raw-value-mask` then fail Claims verification. If the `raw-value-mask` length is different from the `raw-value` length, then fail Claims verification.

If C_X contains `digest` and C_Z contains `digest` with a different binary value, then fail Claims verification. Note that there should be at least one intersecting algorithm for digests. For every intersecting algorithm the digests must be the same.

 © TCG 2020

If C_X contains `version` and C_Z contains `version` with a different or down-revision binary value, then fail Claims verification.

If C_X contains `svn` and C_Z contains `svn` with a different integer value, then fail Claims verification. Note that `svn` refers to a `tagged-svn` that is stripped of the IANA global tag.

If C_X contains `min-svn` and C_Z contains `svn` that is less than `min-svn` then fail Claims verification. Note that `min-svn` refers to a `tagged-min-svn` that is stripped of the IANA global tag.

If C_X contains `flags` and C_Z contains `flags` with a different binary value, then fail Claims verification. When ref doesn't contain `flags` then Evidence must not contain flags.

Etc.… continue checking for other values in C_X and C_Z.

If all value fields match as described above, then add C_X Claim to the `ACCEPTED_CLAIMS`.

Note that the current attesting device may have Evidence Claims that were not anticipated by the Reference Claims. If the device is found to be trustworthy based on the Reference Claims, then it can assert additional Claims that are immediately accepted as definitive. These Claims are held in `PENDING_CLAIMS` until the device is found to be trustworthy.

Step 3:

If the currently attesting device has associated endorsed Claims.

For each Claim C_X in C_{END} and C_Y in `ACCEPTED_CLAIMS`:

If the C_X environment matches the C_Y environment, then add C_X to `ACCEPTED_CLAIMS`

Step 4:

If the current attesting device has identity Claims.

For each Claim C_X in C_{ID} and C_Y in `ACCEPTED_CLAIMS`:

If the C_X environment instance id matches the C_Y environment instance id then add C_X to `ACCEPTED_CLAIMS`

Step 5:

Apply the Appraisal Policy for Evidence to the `ACCEPTED_CLAIMS`

Step 6:

If the `ACCEPTED_CLAIMS` describes a valid trusted Attester, then apply the Appraisal Policy for Evidence to the `PENDING_CLAIMS`

Produce Attestation Results.

End of informative comment

 © TCG 2020

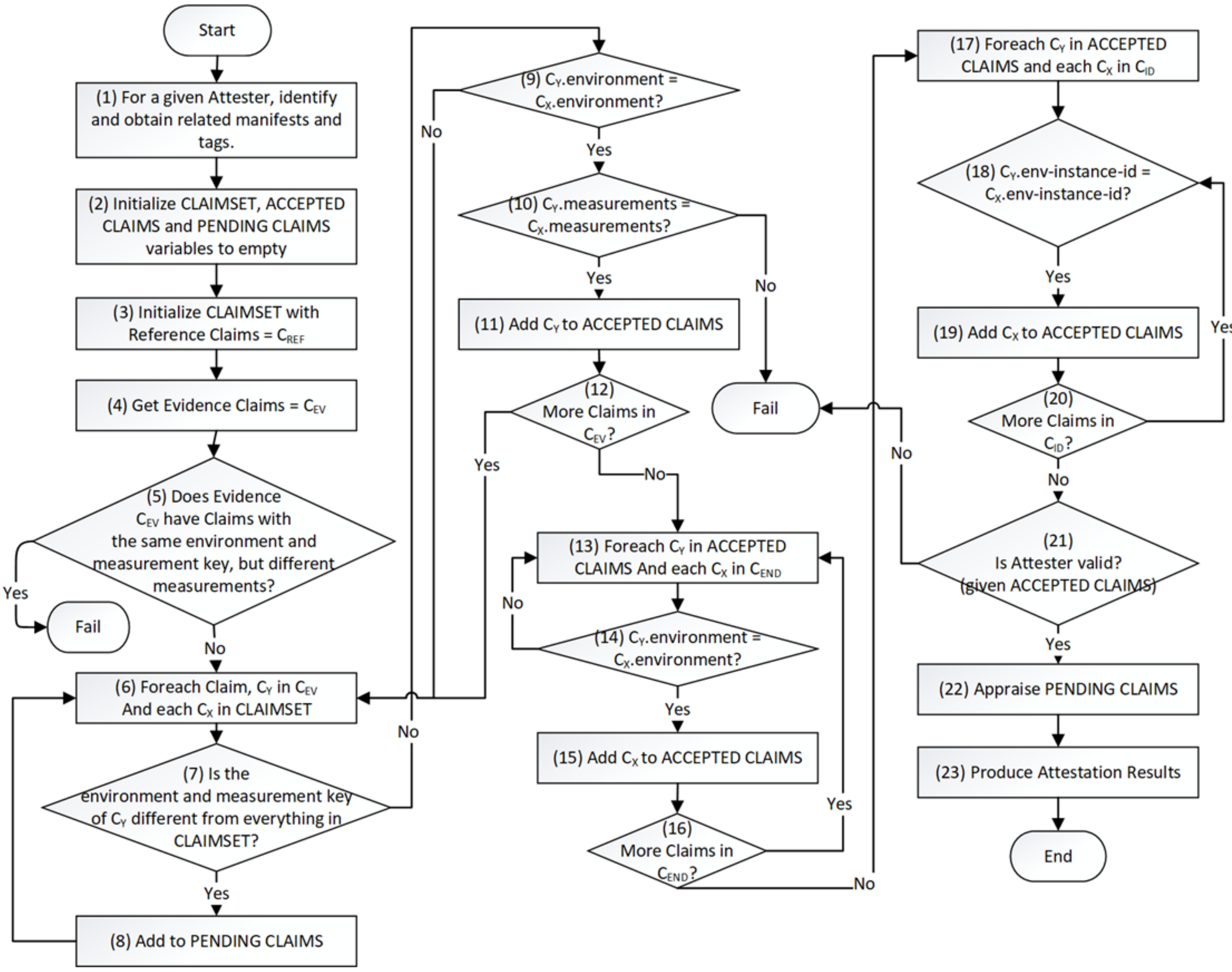

Figure 15: Evidence matching algorithm flowchart

APPENDIX A – ATTESTATION EXAMPLE

The attestation example (see Figure 16) shows an Attester consisting of four environments: a Root of Trust, an Attesting Environment, and two Target Environments. The Root of Trust environment collects Claims about environment E1, constructs Evidence for E1, creates an attestation key for E1 and issues a certificate with E1 Evidence and attestation public key E1. The Attesting Environment E1 collects Claims for two Target Environments E2 and E3 and creates Evidence for both. Evidence might take the form of a signed document or an attestation protocol payload.

The Attester conveys the certificate E1 and Evidence E2 and E3 to a Verifier for appraisal.

There are two Endorsers which issue manifests. Manifest M0 contains endorsed value Claims that describe the Root of Trust environment and a certificate that certifies the Root of Trust attestation key. Manifest M1 contains reference Claims for the Attesting Environment E1. Manifest M2 contains reference Claims for the Target Environments E2 and E3. The Endorsers deposit their manifests containing the various Claim sets in a repository for easy access by Verifiers.

The Verifier is configured to discover manifests, or the Attester could provide a pointer to a different repository. The Verifier inspects the Evidence and Claims for information about which manifests are needed. For example, the E1 certificate might reference the RoT certificate or an environment class identifier from one of the Claim sets could be used to formulate a query to the repository. The manifest M0 might have a locator for M1 and M1 might have a locator for M2.

Once the Verifier has the manifests, it constructs a working set of Claims, (i.e., RoT Endorsed Value Claims, E1, E2, E3 Reference Claims), for the related (linked) tags. To construct a Claim set, a Verifier might need to request Claims from multiple sources. The schema provides a mechanism to find related Claims via linked tags (see Section 5.3.1). The working set (i.e., `CLAIMSET`) determines whether the measurements from each source have corresponding (matching) reference measurements.

The Verifier policy determines whether to trust the Endorser keys that signed the manifests and issued the Root of Trust certificate. If the Root of Trust certificate is trusted and the certificate path associated with the signed Evidence is verified, the Root of Trust endorsed Claims are appraised using Verifier policy. If the Root of Trust and Attester attestation keys are valid and all the Evidence is accepted by the reference Claims, the Verifier generates an Attestation Result.

 © TCG 2020

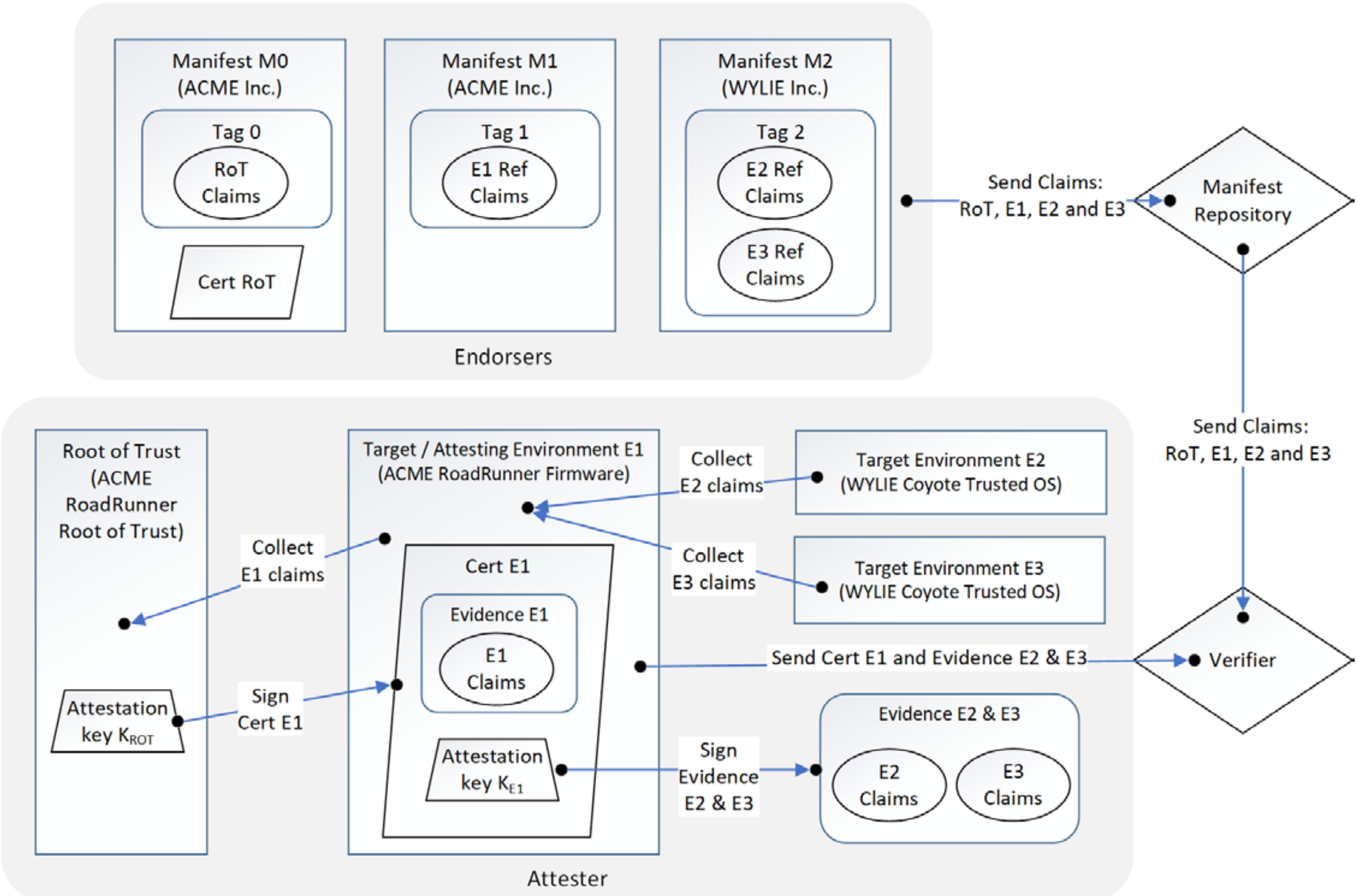

Figure 16: Attestation example

APPENDIX B – ENDORSMENT MANIFEST EXAMPLE

The example CoRIM in 'CBOR diagnostic format' depicts the example from Appendix A where ACME creates a manifest containing endorsed value Claims about the ACME RoadRunner Root of Trust and Reference Values Claims for the ACME RoadRunner Firmware, and two instances of the ABC Trusted OS. The single unsigned manifest containing a single CoMID tag with all four Claim sets is presumed to be issued by ACME Inc.

The manifest is represented in CBOR diagnostic format that is JSON-like with the exception that the tag name and tag numeric are provided to the left of the colon that separates tag information from data values. Tag name values are separated by beginning and ending slash characters (e.g., / name /).

Example CoRIM in CBOR Diagnostic format:

```
/ corim / 500(
  / corim-map / 501({
    / corim.id / 0 : h'284e6c3e5d9f4f6b851f5a4247f243a7',
    / corim.tags / 1 :
      / concise-mid-tag / 506( <<
        / concise-mid-tag / {
          / comid.tag-identity / 1 : {
            / comid.tag-id / 0 : h'3f06af63a93c11e4979700505690773f'
          },
          / comid.entities / 2 : {
            / comid.entity-name / 0 : "ACME Inc.",
            / comid.reg-id / 1 : 32("https://acme.example"),
            / comid.role / 2 : 0 / tag-creator /
          },
          / comid.triples / 4 : {
            / comid.reference-triples / 0 : [
              [
                / environment-map / {
                  / comid.class / 0 : {
                    / comid.class-id / 0 :
                      / tagged-uuid-type / 37(
                        h'a71b3e388d454a0581f352e58c832c5c '
                      ),
                    / comid.vendor / 1 : "ACME Inc.",
                    / comid.model / 2 : "ACME RoadRunner Firmware",
                    / comid.layer / 3 : 1
                  }
                },
                / measurement-map / {
                  / comid.mval / 1 : {
                    / comid.digests / 2 : [
                      / hash-alg-id / 1, / sha256 /
                      / hash-value / h'bb71198ed60a95dc3c619e555c2c0b8d7564a38031b034a195892591c65365b0'
                    ]
                  }
                }
              ],
              [
                / environment-map / {
                  / comid.class / 0 : {
                    / comid.class-id / 0 :
                      / tagged-uuid-type / 37(
                        h'a71b3e388d454a0581f352e58c832c5c'
                      ),
                    / comid.vendor / 1 : "ABC Inc.",
                    / comid.model / 2 : " ABC Trusted OS",
                    / comid.layer / 3 : 2,
                    / comid.index / 4 : 0
                  }
                },
                / measurement-map / {
                  / comid.mval / 1 : {
                    / comid.digests / 2 : [
                      / hash-alg-id / 1, / sha256 /
                      / hash-value / h'bb71198ed60a95dc3c619e555c2c0b8d7564a38031b034a195892591c65365b0'
```

```
                    ]
                  }
                }
              ],
              [
                / environment-map / {
                  / comid.class / 0 : {
                    / comid.class-id / 0 :
                      / tagged-uuid-type / 37(
                        h'5bf4c7c6d8ff4276948834b3706eb2c3'
                        h'a71b3e388d454a0581f352e58c832c5c'
                      ),
                    / comid.vendor / 1 : " ABC Inc.",
                    / comid.model / 2 : " ABC Trusted OS",
                    / comid.layer / 3 : 2,
                    / comid.index / 4 : 1
                  }
                },
                / measurement-map / {
                  / comid.mval / 1 : {
                    / comid.digests / 2 : [
                      / hash-alg-id / 1, / sha256 /
                      / hash-value / h'bb71198ed60a95dc3c619e555c2c0b8d7564a38031b034a195892591c65365b0'
                    ]
                  }
                }
              ]
            ],
            / comid.endorsed-triples / 1 : [
              / environment-map / {
                / comid.class / 0 : {
                  / comid.class-id / 0 :
                    / tagged-uuid-type / 37(
                      h'67b28b6c34cc40a19117ab5b05911e37'
                    ),
                  / comid.vendor / 1 : "ACME Inc.",
                  / comid.model / 2 : "ACME RoadRunner Root of Trust",
                  / comid.layer / 3 : 0
                }
              },
              / measurement-map / {
                / comid.mval / 1 : {
                  / comid.svn / 1 : 552(1)
                }
              }
            ]
          }
        }
      >> )
    }
  )
)
```

Example CoRIM in ASCII CBOR format:

```
d9 1f4                                             # tag(500)
   d9 1f5                                          # tag(501)
      a2                                           # map(2)
         00                                        # unsigned(0)
         50                                        # bytes(16)
            284e6c3e5d9f4f6b851f5a4247f243a7       # "(Nl>]\x9FOk\x85\x1FZBG\xF2C\xA7"
         01                                        # unsigned(1)
         d9 1fa                                    # tag(506)
            a3                                     # map(3)
               01                                  # unsigned(1)
               a1                                  # map(1)
                  00                               # unsigned(0)
                  50                               # bytes(16)
                     3f06af63a93c11e4979700505690773f
# "?\x06\xAFc\xA9<\x11\xE4\x97\x97\x00PV\x90w?"
               02                                  # unsigned(2)
```

```
              a3                                 # map(3)
                 00                              # unsigned(0)
                 69                              # text(9)
                    41434d4520496e632e           # "ACME Inc."
                 01                              # unsigned(1)
                 d8 20                           # tag(32)
                    74                           # text(20)
                       68747470733a2f2f61636d652e6578616d706c65
# "https://acme.example"
                 02                              # unsigned(2)
                 00                              # unsigned(0)
              04                                 # unsigned(4)
              a2                                 # map(2)
                 00                              # unsigned(0)
                 83                              # array(3)
                    82                           # array(2)
                       a1                        # map(1)
                          00                     # unsigned(0)
                          a4                     # map(4)
                             00                  # unsigned(0)
                             d8 25               # tag(37)
                                50               # bytes(16)
                                   67b28b6c34cc40a19117ab5b05911e37
# "g\xB2\x8Bl4\xCC@\xA1\x91\x17\xAB[\x05\x91\x1E7"
                             01                  # unsigned(1)
                             69                  # text(9)
                                41434d4520496e632e # "ACME Inc."
                             02                  # unsigned(2)
                             78 18               # text(24)
                                41434d4520526f616452756e6e6572204669726d77617265
# "ACME RoadRunner Firmware"
                             03                  # unsigned(3)
                             01                  # unsigned(1)
                       a1                        # map(1)
                          01                     # unsigned(1)
                          a1                     # map(1)
                             02                  # unsigned(2)
                             82                  # array(2)
                                01               # unsigned(1)
                                58 20            # bytes(32)
                                   44aa336af4cb14a879432e53dd6571c7fa9bccafb75f488259262d6ea3a4d91b
# "D\xAA3j\xF4\xCB\x14\xA8yC.S\xDDeq\xC7\xFA\x9B\xCC\xAF\xB7_H\x82Y&-n\xA3\xA4\xD9\e"
                    82                           # array(2)
                       a1                        # map(1)
                          00                     # unsigned(0)
                          a5                     # map(5)
                             00                  # unsigned(0)
                             d8 25               # tag(37)
                                50               # bytes(16)
                                   a71b3e388d454a0581f352e58c832c5c
# "\xA7\e>8\x8DEJ\x05\x81\xF3R\xE5\x8C\x83,\\"
                             01                  # unsigned(1)
                             6a                  # text(10)
                                57594c494520496e632e
# "ABC Inc."
                             02                  # unsigned(2)
                             77                  # text(23)
                                57594c494520436f796f74652054727573746564204f53
# "ABC Trusted OS"
                             03                  # unsigned(3)
                             02                  # unsigned(2)
                             04                  # unsigned(4)
                             00                  # unsigned(0)
                       a1                        # map(1)
                          01                     # unsigned(1)
                          a1                     # map(1)
                             02                  # unsigned(2)
                             82                  # array(2)
                                01               # unsigned(1)
                                58 20            # bytes(32)
                                   bb71198ed60a95dc3c619e555c2c0b8d7564a38031b034a195892591c65365b0
```

```
# "\xBBq\x19\x8E\xD6\n\x95\xDC<a\x9EU\\,\v\x8Dud\xA3\x801\xB04\xA1\x95\x89%\x91\xC6Se\xB0"
                     82                             # array(2)
                        a1                          # map(1)
                           00                       # unsigned(0)
                           a5                       # map(5)
                              00                    # unsigned(0)
                              d8 25                 # tag(37)
                                 50                 # bytes(16)
                                    a71b3e388d454a0581f352e58c832c5c
# "\xA7\e>8\x8DEJ\x05\x81\xF3R\xE5\x8C\x83,\\"
                              01                    # unsigned(1)
                              6a                    # text(10)
                                 57594c494520496e632e
# "ABC Inc."
                              02                    # unsigned(2)
                              77                    # text(23)
                                 57594c494520436f796f746520547275737465642204f53
# "ABC Trusted OS"
                              03                    # unsigned(3)
                              02                    # unsigned(2)
                              04                    # unsigned(4)
                              01                    # unsigned(1)
                        a1                          # map(1)
                           01                       # unsigned(1)
                           a1                       # map(1)
                              02                    # unsigned(2)
                              82                    # array(2)
                                 01                 # unsigned(1)
                                 58 20              # bytes(32)
                                    bb71198ed60a95dc3c619e555c2c0b8d7564a38031b034a195892591c65365b0
# "\xBBq\x19\x8E\xD6\n\x95\xDC<a\x9EU\\,\v\x8Dud\xA3\x801\xB04\xA1\x95\x89%\x91\xC6Se\xB0"
                  01                                # unsigned(1)
                  82                                # array(2)
                     a1                             # map(1)
                        00                          # unsigned(0)
                        a4                          # map(4)
                           00                       # unsigned(0)
                           d8 25                    # tag(37)
                              50                    # bytes(16)
                                 67b28b6c34cc40a19117ab5b05911e37
# "g\xB2\x8Bl4\xCC@\xA1\x91\x17\xAB[\x05\x91\x1E7"
                           01                       # unsigned(1)
                           69                       # text(9)
                              41434d4520496e632e    # "ACME Inc."
                           02                       # unsigned(2)
                           72                       # text(18)
                              41434d4520526f6f74206f66205472757374
# "ACME Root of Trust"
                           03                       # unsigned(3)
                           00                       # unsigned(0)
                     a1                             # map(1)
                        01                          # unsigned(1)
                        a1                          # map(1)
                           01                       # unsigned(1)
                           d9 228                   # tag(552)
                              01                    # unsigned(1)
```


© TCG 2020

APPENDIX C – CORIM CDDL EXAMPLES

This section contains sample CDDL for selected CoRIM structures.

```
corim = #6.500($concise-reference-integrity-manifest-type-choice)
tagged-corim-map = #6.501(corim-map)
$concise-reference-integrity-manifest-type-choice /= tagged-corim-map
$concise-reference-integrity-manifest-type-choice /= #6.502(signed-corim)

corim-map = {
  corim.id => $corim-id-type-choice
  corim.tags => [ + $concise-tag-type-choice ]
  ? corim.dependent-rims => [ + corim-locator-map ]
  ? corim.profile => [ + profile-type-choice ]
  ? corim.rim-validity => validity-map
  ? corim.entities => [ + corim-entity-map ]
  * $$corim-map-extension
}

profile-type-choice = uri / tagged-oid-type

corim-locator-map = {
  corim.href => uri
  ? corim.thumbprint => hash-entry
}

$concise-tag-type-choice /= #6.505(bytes .cbor concise-swid-tag)
$concise-tag-type-choice /= #6.506(bytes .cbor concise-mid-tag)

corim-entity-map = {
  corim.entity-name => $entity-name-type-choice
  ? corim.reg-id => uri
  corim.role => $corim-role-type-choice
  * $$corim-entity-map-extension
}

$corim-role-type-choice /= corim.manifest-creator

signed-corim = #6.18(COSE-Sign1-corim)

protected-corim-header-map = {
  corim.alg-id => int
  corim.content-type => "application/rim+cbor"
  corim.issuer-key-id => bstr
  corim.meta => bstr .cbor corim-meta-map
  * cose-label => cose-values
}

corim-meta-map = {
  corim.signer => corim-signer-map
  ? corim.signature-validity => validity-map
}

corim-signer-map = {
```

 © TCG 2020

```
  corim.signer-name => $entity-name-type-choice
  ? corim.signer-uri => uri
  * $$corim-signer-map-extension
}

validity-map = {
  ? corim.not-before => time
  corim.not-after => time
}

unprotected-corim-header-map = {
  * cose-label => cose-values
}

COSE-Sign1-corim = [
  protected: bstr .cbor protected-corim-header-map
  unprotected: unprotected-corim-header-map
  payload: bstr .cbor tagged-corim-map
  signature: bstr
]

concise-mid-tag = {
  ? comid.language => language-type
  comid.tag-identity => tag-identity-map
  ? comid.entities => [ + entity-map ]
  ? comid.linked-tags => [ + linked-tag-map ]
  comid.triples => triples-map
  * $$concise-mid-tag-extension
}

language-type = text

tag-identity-map = {
  comid.tag-id => $tag-id-type-choice
  ? comid.tag-version => tag-version-type
}

$tag-id-type-choice /= tstr
$tag-id-type-choice /= uuid-type

tag-version-type = uint .default 0

entity-map = {
  comid.entity-name => $entity-name-type-choice
  ? comid.reg-id => uri
  comid.role => [ + $comid-role-type-choice ]
  * $$entity-map-extension
}

$comid-role-type-choice /= comid.tag-creator
$comid-role-type-choice /= comid.creator
$comid-role-type-choice /= comid.maintainer

linked-tag-map = {
```

 © TCG 2020

```
  comid.linked-tag-id => $tag-id-type-choice
  comid.tag-rel => $tag-rel-type-choice
}

$tag-rel-type-choice /= comid.supplements
$tag-rel-type-choice /= comid.replaces

triples-map = non-empty<{
  ? comid.reference-triples => [ + reference-triple-record ]
  ? comid.endorsed-triples => [ + endorsed-triple-record ]
  ? comid.attest-key-triples => [ + attest-key-triple-record ]
  ? comid.identity-triples => [ + identity-triple-record ]
  * $$triples-map-extension
}>

; REFERENCE-VALUE triple
; "target environment $TE" "has reference measurements" "$RV"
reference-triple-record = [
  environment-map ; target environment
  [ + measurement-map ] ; reference measurements
]

; ENDORSED-VALUE triple
; "environment $E" "has endorsed measurements" "$EV"
endorsed-triple-record = [
  environment-map ; (target or attesting) environment
  [ + measurement-map ] ; endorsed measurements
]

; ATTESTATION-VERIFICATION-KEY triple
; "attesting environment $AE" "signs Evidence that can be verified with key" "$K"
attest-key-triple-record = [
  environment-map ; attesting environment
  [ + $crypto-key-type-choice ] ; attestation verification key(s)
]

; DEVICE-IDENTITY triple
; "device $D" "is identified by key" "$K"
identity-triple-record = [
  environment-map ; device identifier (instance or class)
  [ + $crypto-key-type-choice ] ; DevID, or semantically equivalent
]

; Public key in DER format base64-encoded.
$key-type-choice /= tstr

; Optional X.509 certificate chain corresponding to the public key
; in comid.key, encoded as an array of one or more base64-encoded
; DER PKIX certificates.
$cert-type-choice /= tstr

environment-map = non-empty<{
  ? comid.class => class-map
  ? comid.instance => $instance-id-type-choice
```

 © TCG 2020

```
  ? comid.group => $group-id-type-choice
}>

class-map = non-empty<{
  ? comid.class-id => $class-id-type-choice
  ? comid.vendor => tstr
  ? comid.model => tstr
  ? comid.layer => uint
  ? comid.index => uint
}>

$class-id-type-choice /= tagged-oid-type
$class-id-type-choice /= tagged-uuid-type

$instance-id-type-choice /= tagged-ueid-type
$instance-id-type-choice /= tagged-uuid-type

$group-id-type-choice /= tagged-uuid-type

oid-type = bytes
tagged-oid-type = #6.111(oid-type)

;
; github.com/lucas-clemente/cbor-specs/blob/master/uuid.md
;
tagged-uuid-type = #6.37(uuid-type)

;
; From draft-ietf-rats-eat
;
ueid-type = bytes .size 33
tagged-ueid-type = #6.550(ueid-type)

$measured-element-type-choice /= tagged-oid-type
$measured-element-type-choice /= tagged-uuid-type
$measured-element-type-choice /= uint

measurement-map = {
  ? comid.mkey => $measured-element-type-choice
  comid.mval => measurement-values-map
}

measurement-values-map = non-empty<{
  ? comid.ver => version-map
  ? comid.svn => svn-type-choice
  ? comid.digests => digests-type
  ? comid.flags => flags-map
  ? raw-value-group
  ? comid.mac-addr => mac-addr-type-choice
  ? comid.ip-addr =>  ip-addr-type-choice
  ? comid.serial-number => serial-number-type
  ? comid.ueid => ueid-type
  ? comid.uuid => uuid-type
  ? comid.name => tstr
```

 © TCG 2020

```
  * $$measurement-values-map-extension
}>

version-map = {
  comid.version => version-type
  ? comid.version-scheme => $version-scheme
}
version-type = text .default '0.0.0'
; version-scheme is defined in CoSWID

svn-type = uint
svn = svn-type
min-svn = svn-type
tagged-svn = #6.552(svn)
tagged-min-svn = #6.553(min-svn)
svn-type-choice = tagged-svn / tagged-min-svn


; operational flags maps to DiceTcbInfo.flags
flags-map = {
  ? comid.operational-flag-configured => bool
  ? comid.operational-flag-secure => bool
  ? comid.operational-flag-recovery => bool
  ? comid.operational-flag-debug => bool
  ? comid.operational-flag-replay-protected => bool
  ? comid.operational-flag-integrity-protected => bool
  * $$flags-map-extension
}

raw-value-group = (
  comid.raw-value => $raw-value-type-choice
  ? comid.raw-value-mask => raw-value-mask-type
)

$raw-value-type-choice /= #6.560(bytes)

raw-value-mask-type = bytes

ip-addr-type-choice = ip4-addr-type / ip6-addr-type
ip4-addr-type = bytes .size 4
ip6-addr-type = bytes .size 16

mac-addr-type-choice = eui48-addr-type / eui64-addr-type
eui48-addr-type = bytes .size 6
eui64-addr-type = bytes .size 8

serial-number-type = text

; Notes:
; - hash-entry is defined in CoSWID schema
digests-type = [ + hash-entry ]
```

APPENDIX D – UML DIAGRAMS

This section contains Unified Modeling Language (UML) diagrams for CoRIM, CoMID, CoSWID (simplified), and Xcorim.

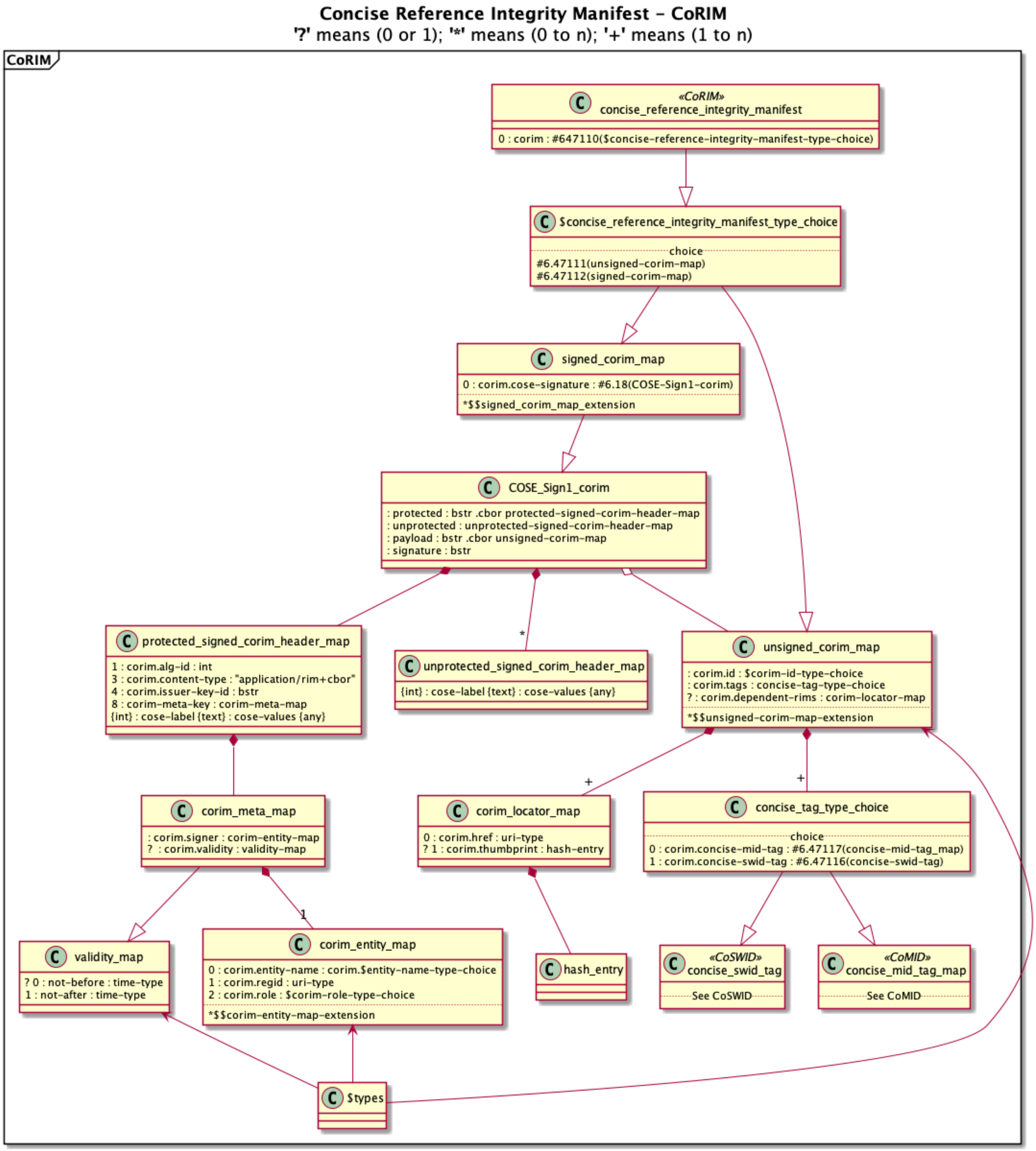

Figure 17: CoRIM data model in UML

 © TCG 2020

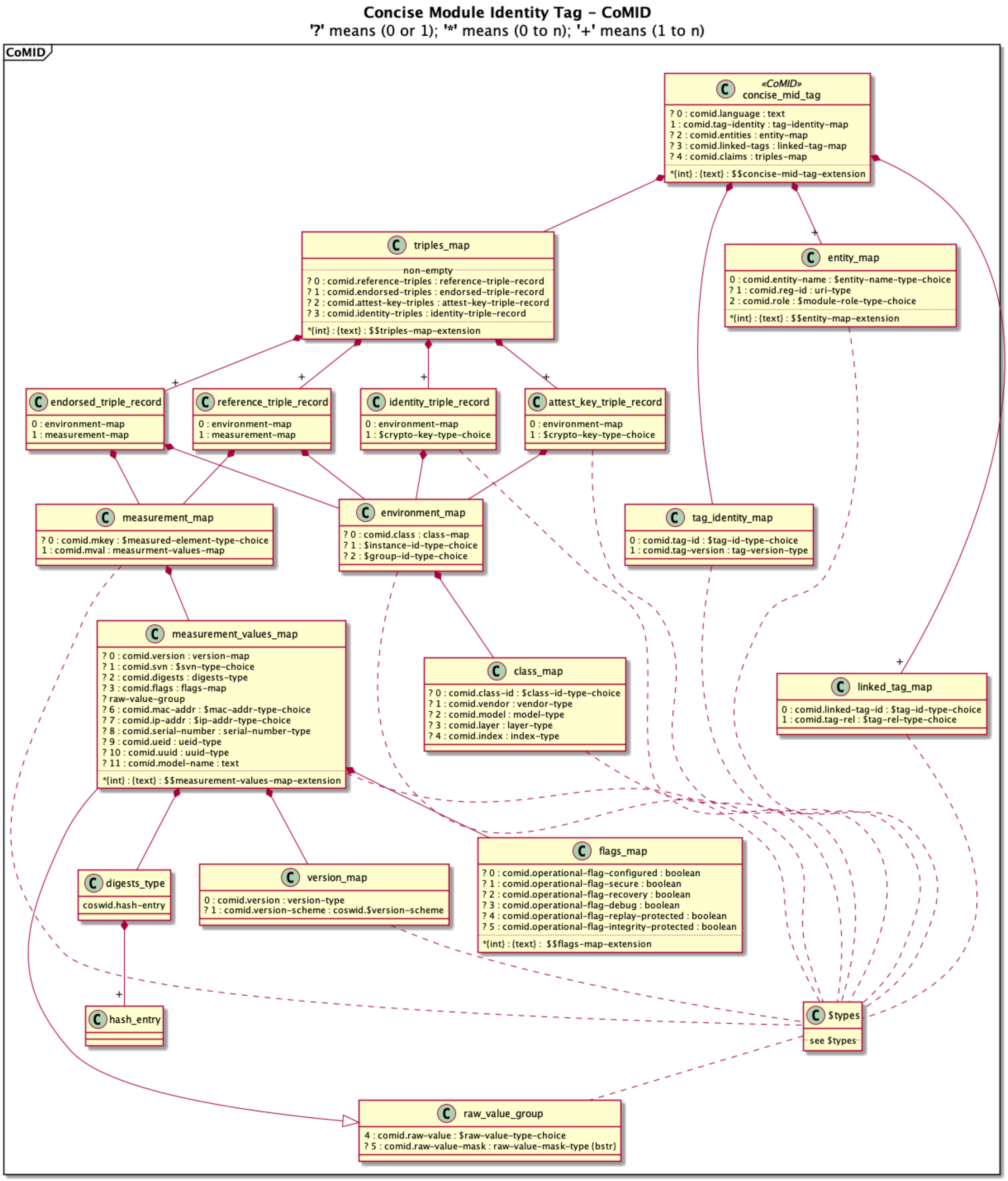

Figure 18: CoMID data model in UML

Figure 19: CoRIM and Xcorim data types in UML.

 © TCG 2020

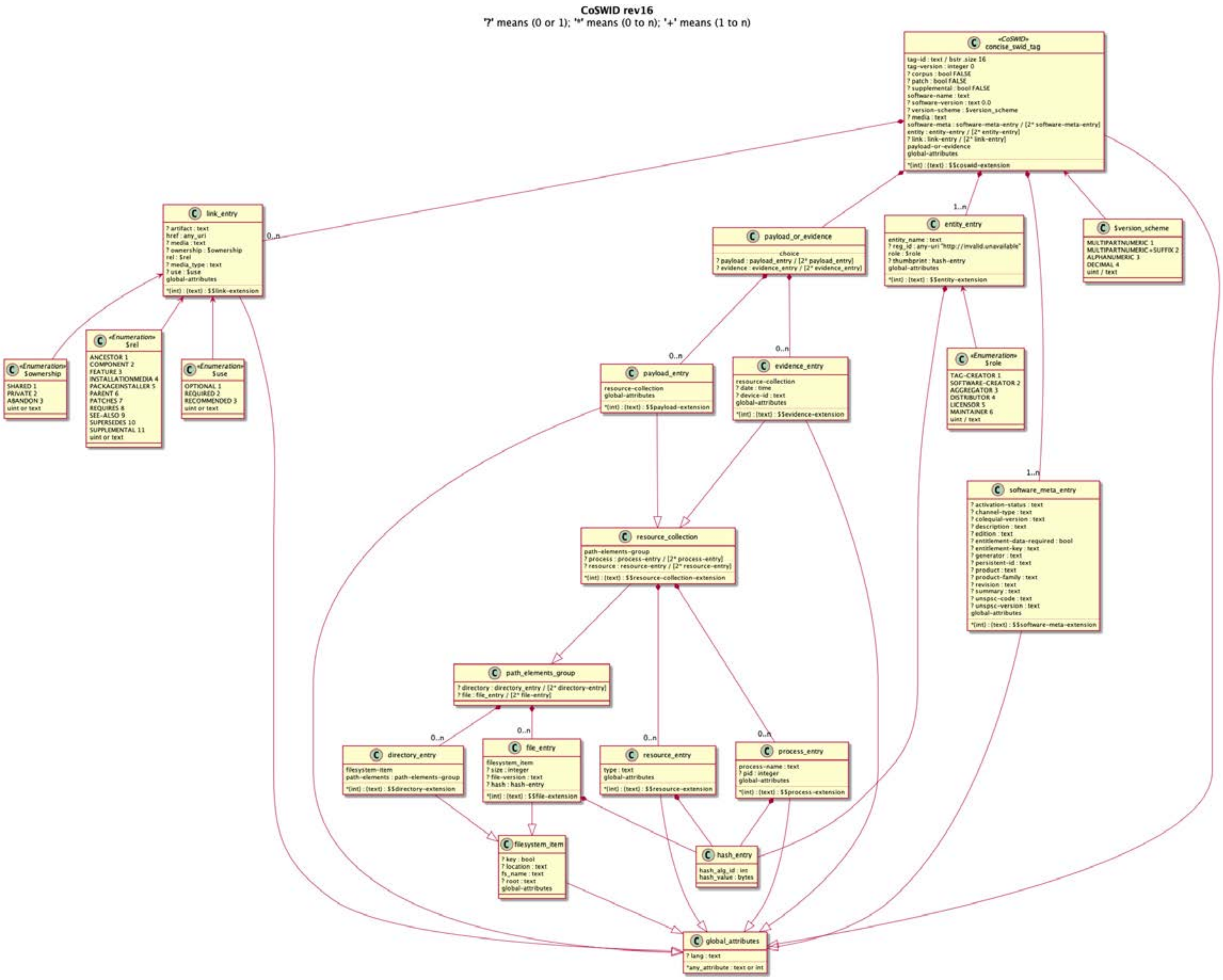

Figure 20: CoSWID data model in UML.

Visual inspection of Figure 20 may require magnification and a high-resolution display.

 © TCG 2020

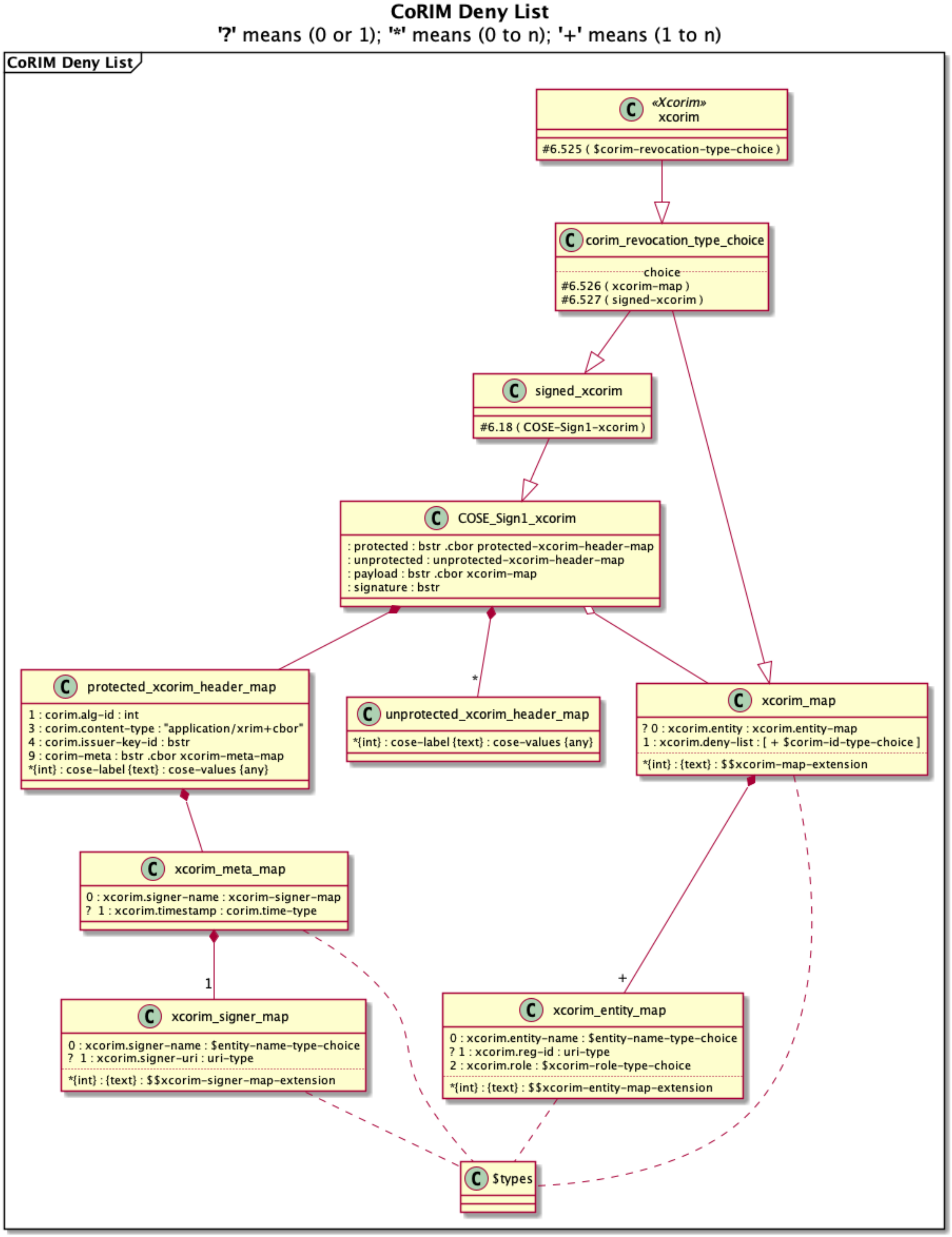

Figure 21: Xcorim - a CoRIM deny list in UML.