TRUSTED® COMPUTING GROUP

SPECIFICATION

# DICE Attestation Architecture

Version 1.1
Revision 0.17
May 4, 2023

Contact: admin@trustedcomputinggroup.org

Public Review

## Work in Progress

*This document is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based on this document.*

# DISCLAIMERS, NOTICES, AND LICENSE TERMS

# CONTENTS

 

# FIGURES





 

# 1 SCOPE

This specification defines an attestation architecture for DICE layering architectures and X.509 certificate extensions for attestation Evidence and Endorsements.

## 1.1 Key Words

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

## 1.2 Statement Type

Please note a very important distinction between different sections of text throughout this document. There are two distinctive kinds of text: informative comment and normative 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**

# 2 REFERENCES


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

[2] Trusted Computing Group, "TCG Attestation Framework Part 1," 2023. [Online].

[3] Trusted Computing Group, "DICE Endorsement Architecture for Devices," 2023. [Online].

[4] Trusted Computing Group, "DICE Layering Architecture," 2020. [Online]. Available: https://www.trustedcomputinggroup.org/.

[5] Trusted Computing Group, "TCG Reference Integrity Manifest (RIM) Information Model," 2019. [Online].

[6] Trusted Computing Group, "TCG Trusted Attestation Protocol Information Model for TPM families 1.2 and 2.0 and DICE Family 1.0," 2019. [Online]. Available: https://www.trustedcomputinggroup.org.

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

[8] Internet Engineering Task Force, "Concise Binary Object Representation (CBOR)," 2020. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8949.html.

[9] Internet Engineering Task Force, "Concise Software Identification Tags," 2019. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-sacm-coswid/.

[10] Internet Engineering Task Force, "The Entity Attestation Token," 2019. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-rats-eat/.

[11] Internet Engineering Task Force, "CBOR Web Token (CWT)," 2018. [Online]. Available: https://tools.ietf.org/html/rfc8392.

[12] Internet Engineering Task Force, "Remote Attestation Procedures (RATS) Architecture," January 2023. [Online]. Available: https://www.ietf.org/rfc/rfc9334.html.

[13] Internet Engineering Taskforce, "RATS Conceptual Messages Wrapper," October 2022. [Online]. Available: https://datatracker.ietf.org/doc/draft-ftbs-rats-msg-wrap/.

[14] Internet Engineering Taskforce, "Key Attestation Extension for Certificate Management Protocols," October 2022. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-lamps-key-attestation-ext/.

[15] Internet Engineering Taskforce, "Web Authentication: An API for Accessing Public Key Credentials Level 3," January 2023. [Online]. Available: https://w3c.github.io/webauthn/#sctn-attestation-formats.

[16] Internet Engineering Taskforce, "On Stable Storage for Items in Concise Binary Object Representation (CBOR)," August 2022. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc9277.

[17] Internet Engineering Taskforce, "Media Type Specifications and Registration Procedures," January 2013. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6838.

[18] Internet Engineering Taskforce, "The Constrained Application Protocol (CoAP)," June 2014. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7252.

[19] W3C, "Extensible Markup Language (XML) 1.0 (Fifth Edition)," 2008. [Online]. Available: https://www.w3.org/TR/xml/.

[20] NIST, "Guidelines for the Creation of Interoperable Software Identification (SWID) Tags," 2016. [Online]. Available: https://www.nist.gov.

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

[22] Internet Engineering Task Force, "Uniform Resource Identifier (URI): Generic Syntax," 2005. [Online]. Available: https://tools.ietf.org/html/rfc3986.

[23] Internet Engineering Task Force, "An Internet Attribute Certificate Profile for Authorization," 2010. [Online]. Available: https://tools.ietf.org/html/rfc5755.

[24] NIST, "Security Requirements for Cryptographic Modules," 2019. [Online]. Available: https://csrc.nist.gov/publications/detail/fips/140/3/final.



 

# 3 TERMS AND DEFINITIONS

For the purposes of this document, the following terms and definitions apply. Some of these terms have related definitions in the Trusted Computing Group Glossary [1].

## 3.1 Glossary

| TERM | DEFINITION |
|---|---|
| **Actor** | A computing entity (e.g., device, server, service) that hosts or otherwise implements one or more attestation Roles. |
| **Appraisal, of Evidence** | Evaluation of Evidence for the purpose of assessing Attester status according to Reference Values. |
| **Appraisal, of Attestation Results** | Evaluation of Attestation Results for the purpose of altering Relying Party behavior according to the trustworthiness of an Attester. |
| **Appraisal Policy for Attestation Results** | A set of rules that direct the evaluation by a Relying Party of the validity of information about an Attester. Typically, such policies are authorized by the Relying Party Owner. See [2]. |
| **Appraisal Policy for Evidence** | A set of rules, instructions, configurations, or other input that directs the evaluation by a Verifier of the validity of Evidence about and Endorsements for the Attester. Such policies are authorized by the Verifier Owner. See [2]. |
| **Attestation** | The process of generating, conveying, and appraising Claims, backed by cryptographic Evidence, about Attester trustworthiness characteristics that may include the trustworthiness of the supply chain, identity, device provenance, software configuration, device composition, compliance to test suites, functional and assurance evaluations. *See* [1] – *Attestation.* |
| **Attestation Results** | The results of Evidence appraisal that are generated by a Verifier, and typically include information about an Attester. |
| **Attestation Service Provider** | A service provider entity that implements the Verifier role. Typically, the ASP is remote with respect to the Device / Attester. It may also be remote relative to a supply chain entity / Endorser and Resource Manager / Relying Party. |
| **Attester** | An attestation Role that contains at least one Attesting Environment and implements Attester functions (e.g., measurement, reporting, storage, etc.). Attesters convey Evidence that vouches for Attester integrity and veracity to a Verifier. |
| **Attribute Certificate** | A structure containing signed Claims that complies with a standard certificate format and encoding such as [3]. *See Endorsements, Evidence.* |
| **Assertion** | An abstract expression (or information) describing a property that is used to appraise trustworthiness or integrity. See also Reference Value. |
| **Claim, Measurement** | A machine-readable assertion about an Attester that has trustworthiness properties, attributes or identifiers that can be included in Evidence, Endorsements, or Attestation Results. See [1] – *Integrity Measurement*. |
| **Composite Attester** | The Attester in a Composite Device. |
| **Composite Device** | A device with an integrated set of components. |

 

| | |
|---|---|
| **Conveyance** | A mechanism for transferring Evidence, Endorsements, Attestation Results, or policies. |
| **Device** | An implementation of an Actor that performs attestation Roles, typically an Attester. *See* [1]– *Trusted Device.* |
| **Device Identity** | A value that identifies and authenticates an Actor such as a device, TCB, or RoT. A Device Identity has a credential that authenticates its identifier, such as an IEEE IDevID certificate [4]. |
| **Endorsements** | Authenticatable Claims about Attester trustworthiness properties that are either Reference Values or Endorsed Values (e.g., a Reference Integrity Manifest – see [5]) or credentials that authenticate Attester identity (e.g., device identity certificates, see [4], [3]). |
| **Endorser** | An attestation Role that creates, provisions, or conveys Endorsements to Verifiers. |
| **Evidence** | Authenticatable Claims asserted by an Attester about one or more Target Environments that is conveyed from the Attester to a Verifier. |
| **Manifest** | A structure that contains Endorsements, Evidence, or Attestation Results. |
| **Platform** | *See Device. See* [1] – *Platform, Trusted Platform.* |
| **Relying Party** | An attestation Role, typically an entity that manages resources or grants access, that accepts Attestation Results from a Verifier. |
| **Relying Party Owner** | An attestation Role that conveys Appraisal Policy for Attestation Results to a Relying Party. |
| **Resource Manager** | An entity that hosts the Relying Party. |
| **Role** | Attestation behaviors and characteristics distinguished by their role name: Attester, Endorser, Verifier, Relying Party, Verifier Owner, and Relying Party Owner. Roles are implemented by one or more Actors. |
| **Root of Trust** | *See* [1] – *Trust, Root of Trust* |
| **Topology Model** | The organizational structure of a role composition. This specification provides a canonicalization of commonly used role compositions. These include Passport, Background Check, and Multi-Party Background Check. |
| **Trusted Computing Base** | Protected capabilities and shielded locations that exist because of protected state transitions. See [1] – *Trusted Building Block*, *Trusted Component*, *Trusted Device* |
| **Verifier** | An attestation Role that accepts Evidence from Attesters, Endorsements from Endorsers, and conveys Attestation Results to Relying Parties. The Verifier typically appraises Evidence to determine Attester trustworthiness. See [2]– *Verifier.* |
| **Verifier Owner** | An attestation Role that conveys Appraisal Policy for Evidence to a Verifier. See [2]– *Owner.* |

 

## 3.2 Abbreviations

| ABBREVIATION | DESCRIPTION |
|---|---|
| **ASP** | Attestation Service Provider |
| **CDI** | Compound Device Identifier |
| **CRL** | Certificate Revocation List |
| **ECA** | Embedded Certificate Authority |
| **IDevID** | Initial Device ID |
| **RoT** | Root of Trust |
| **TCB** | Trusted Computing Base |
| **TCI** | TCB Component Identifier |
| **UEID** | Universal Entity ID |



 

# 4 INTRODUCTION

**Start of informative comment**

Trustworthiness attributes are not a finite set of values. Attester environments can vary widely, ranging from those highly resistant to attack to those having little or no resistance. Configuration options, if set poorly, can result in a highly resistant environment being operationally less resistant. Computing environments are typically updatable, being constructed from reprogrammable hardware, firmware, software, and memory. When a trustworthy environment changes, it is often necessary to determine whether the change transitioned the environment from a trustworthy state to an untrustworthy state. An attestation architecture provides a framework for anticipating when a trust relevant change occurs, what changed, and whether the change is relevant to device security. An attestation framework also creates a context for enabling appropriate responses by applications, system software, and protocol endpoints when trust relevant changes do occur.

A trustworthiness assertion is information that describes the properties of a device that affects the Verifier or Relying Party perception of the device's integrity. The set of possible assertions is expected to be determined by the computing environments that support attestation. In many cases, there will be a set of assertions that is widely applicable across most, if not all, computing environments of a particular type. Conversely, there will be assertions that are unique to specific environments or devices. Therefore, this attestation architecture incorporates extensible mechanisms for representing assertions.

Computing environments can be structurally complex and consist of multiple components (memory, CPU, storage, networking, firmware, software). Components are often linked and composed to form computational pipelines, arrays, networks, etc. Not every component is expected to be capable of attestation, and attestation capable components may not be capable of attesting to every computing element that interacts with the computing environment. This attestation architecture anticipates use of information modeling techniques that describe computing environment architectures so that verification operations may rely on the information model as an interoperable way to navigate structural complexity.

The attestation capability itself is a computing environment. The act of monitoring trustworthiness attributes, collecting them into an interoperable format, integrity protecting, authenticating, and conveying them employs a computing environment - one that is separate from the one being attested. The trustworthiness of the attestation capability is also a consideration for the attestation architecture. It should be possible for a Verifier to understand the trustworthiness properties of the attestation capability for any set of assertions of an attestation flow. This attestation architecture anticipates recursive trust properties and the need for termination. Ultimately, a portion of the computing environment trustworthiness is established via non-automated means. For example, design reviews, manufacturing process audits, and physical security. For this reason, a trustworthy attestation mechanism depends on trustworthy manufacturing and supply chain practices.

**End of informative comment**

 

# 5 ATTESTATION ARCHITECTURE

**Start of informative comment**

This section offers a review of attestation framework requirements [2] and layering architecture [4]. Attestation can take many forms ranging from local to remote and implicit to explicit. Implicit and explicit attestation are defined more completely in [6]. This attestation architecture can use both implicit and explicit attestation in both local and remote deployments. All forms of attestation can coexist, and implicit and explicit attestation forms are not mutually exclusive and can be asserted in the same attestation event.

A set of roles and conceptual messages capture attestation flow. Roles are performed by Actors (deployed entities) that together instantiate different deployment models. Roles and Actors may combine or partition attestation flow into a variety of possible deployments. However, deployment models do not fundamentally modify the expected attestation flow where the conceptual message always originates from the identified role, and always is consumed by the identified role.

This attestation architecture defines certificate extensions that may be used to construct attestation Evidence or Reference Values.

The basic functions of this attestation architecture are the creation, conveyance, and appraisal of attestation Evidence. The Attester creates attestation Evidence that is conveyed to a Verifier for appraisal. The appraisals compare Evidence with Endorsements. Endorsements are the possible values that the Verifier expects to find in Evidence. Endorsements are obtained from manufacturers, vendors, and other supply chain entities called Endorsers. There can be multiple forms of appraisal (e.g., software integrity verification, device composition and configuration verification, device identity and provenance verification). Attestation Results are the output of appraisals that are conveyed to Relying Parties. Attestation Results provide the basis by which the Relying Party may determine a level of confidence in subsequent operations.

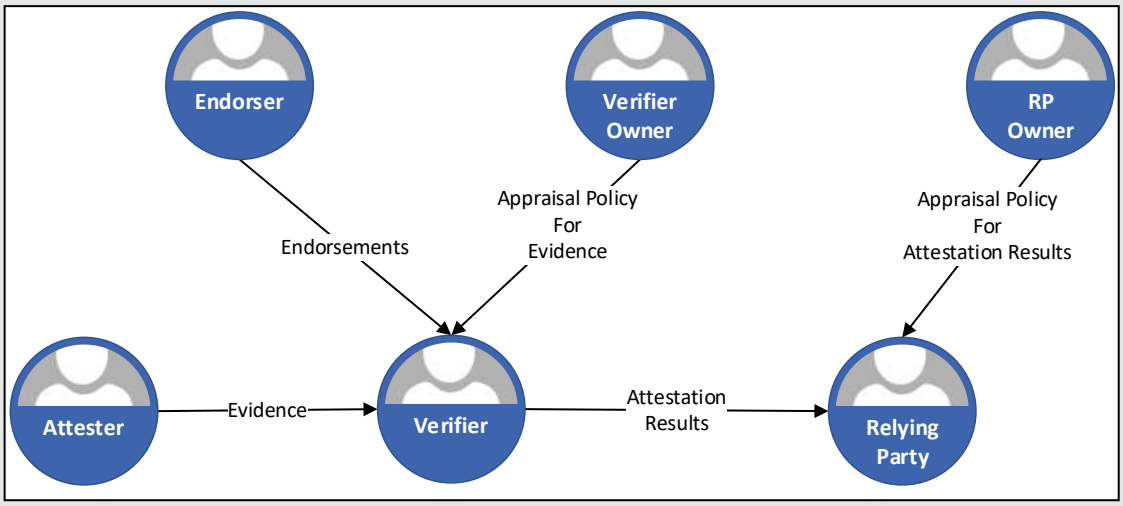


*Figure 1: Attestation Roles and message flow*

This architecture defines attestation Roles (i.e., Attester, Verifier, Endorser, Relying Party, and Owner) and the messages they exchange. Message structure and the various ways in which Roles may be hosted, combined and divided are also part of the architecture. Messages are protected either by a data structure approach (e.g., X.509 certificates, RFC8392) and/or by a conveyance protocol (e.g., RFC5246).

 

## 5.1 Attestation Roles

The attestation roles architecture primarily focuses on the trust model elements of a system. There are five roles defined by the attestation roles architecture, as illustrated in Figure 1. Roles consume and/or produce attestation related information. There are a variety of possible configurations in which role interactions may occur. The attestation roles architecture is a canonical model for a broad range of attestation scenarios. Different scenarios may require topological and/or deployment specific considerations. The primary objective of the attestation roles architecture is to define the functions pertaining to roles and the information exchanged between roles. Attestation roles may be combined and separated as needed to accommodate the requirements of a deployment or use case.

The roles' workflow produces and consumes attestation messages (see §5.2). There are a variety of possible configurations. The workflow shown in Figure 2 is the canonical interaction. The canonical interaction is preserved across topological models described in §5.3.

### 5.1.1 Attester Role

The Attester Role provides attestation Evidence to a Verifier. The Attester has an attestation identity that is used to authenticate Evidence. The attestation identity is often established as part of a manufacturing process that embeds identity credentials in the entity that implements an Attester.

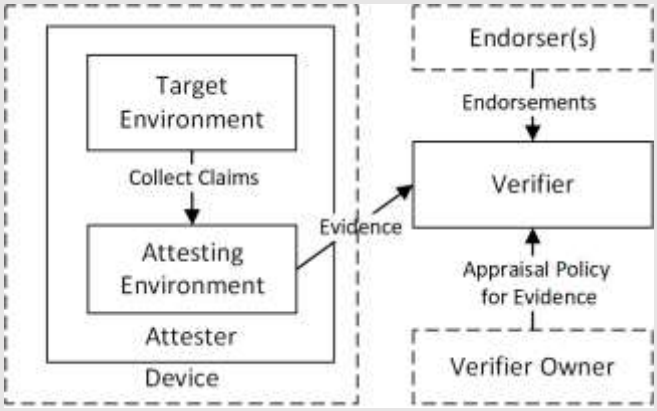


*Figure 2: Device with Attesting Environment and Target Environment*

The Attester consists of an Attesting Environment and a Target Environment. The Attesting Environment collects assertions, called Claims, about the trustworthiness properties of the Target Environment. Claims are packaged as Evidence by the Attesting Environment, integrity protected and authenticated. The Attesting Environment may also supply additional claims that attest the freshness and recentness of collected claims.

Each TCB in a layered device can be an Attesting Environment that may generate Evidence.

When a DICE layer $L_N$ is a Target Environment, the DICE layer $L_{N-1}$ is an Attesting Environment that attests the state of layer $L_N$, and so forth.

In this scenario, prior to executing layer N, layer N is the Target Environment that is measured by the layer N-1 Attesting Environment (assuming layer N-1 implements the Attester role). When layer N-1 transitions control to layer N, layer N acts as the Attesting Environment for Layer N+1 (assuming layer N also implements the Attester role), and so forth.

The Attester may interact with the Endorser to obtain device identity and Endorsed Values. Typically, this happens as part of a manufacturing process involving the construction of the device.

 

If the Attester and Endorser roles are implemented across DICE layers, the previous layer ($L_{N-1}$) may implement an Endorser role while the current layer ($L_N$) or higher layers ($L_{N+y}$) may implement the Attester role.

### 5.1.2 Endorser Role

An Endorser role is typically implemented by a supply chain entity that creates reference Endorsements (i.e., values or measurements that are known to be correct, e.g., a reference manifest). Endorsements contain assertions about a device's intrinsic trustworthiness properties. Endorsers implement manufacturing, productization, or other techniques that establish the trustworthiness properties of the Attesting Environments. DICE RoT and TCB layers contain Attesting Environments. There may be multiple Endorsers for a given device or DICE layer. Endorsers typically authorize Endorsements using digital signatures. For example, certificates [7], manifests [8], [9], or packages.

### 5.1.3 Verifier Role

The Verifier role is implemented by an Actor that accepts Endorsements and Evidence, and then conveys Attestation Results to one or more Relying Parties. Typically, for remote attestation, a service provider entity implements this role. The Verifier has a trust relationship with its Owner(s) and obtains and applies Appraisal Policies for Evidence as part of Evidence appraisal. The Verifier needs to authenticate Owner policies and the Verifier is trusted to correctly apply supplied policies.

### 5.1.4 Verifier Owner Role

The Verifier Owner role provides the policy oversight for the Verifier. The Verifier Owner generates Appraisal Policy for Evidence and conveys the policy to the Verifier. The Verifier Owner sets policy for acceptable (or unacceptable) Evidence and Endorsements that may be supplied by Attesters and Endorsers. The policies determine the trustworthiness state of the Attester and how best to represent the state to Relying Parties in the form of Attestation Results.

The Verifier Owner manages Endorsements supplied by Endorsers and may maintain a database of acceptable and/or unacceptable Endorsements. The Verifier Owner authenticates Endorsements and maintains a list of trustworthy Endorsers.

Verifier Owner policies are conveyed to Verifiers. The Verifier works on behalf of the Verifier Owner.

A Verifier Owner is typically implemented by an Actor that deploys management consoles, network management equipment, security enforcement equipment, etc., or performs operational and system lifecycle management functions. The Verifier Owner and Verifier, or Verifier Owner and Relying Party, typically have an established legal or business relationship.

### 5.1.5 Relying Party Role

The Relying Party role is typically implemented by a resource manager that accepts Attestation Results from a Verifier**.** The Relying Party trusts the Verifier to correctly evaluate Attestation Evidence and Appraisal Policies, and to produce correct Attestation Results. The Relying Party evaluates Attestation Results according to Appraisal Policies for Attestation Results that it receives from the Relying Party Owner.

The Relying Party may take actions based on its evaluation and appraisals. For example, actions may include admitting or denying access, applying remediations, making entries in an audit log, or triggering a financial or other form of transaction. Actions taken by a Relying Party are out of scope of the Attestation Roles model with one exception; a Relying Party may return Results to the Attester for sharing with other Relying Parties.

 

### 5.1.6 Relying Party Owner Role

The Relying Party Owner Role (RP Owner) has policy oversight over the Relying Party. The RP Owner sets appraisal policy regarding acceptable (or unacceptable) attestation results about an Attester produced by the Verifier.

The RP Owner attestation policies are made available to the relevant services, management consoles, network equipment, etc., that enforce the policies set by the RP Owner. These are ancillary to this specification's definition of RP Owner and out of scope for this specification.

Note that, as with some other Attestation Roles, the Replying Party Owner Role and the Relying Party Role may be co-located. This means that a single Actor (e.g., a cloud service provider) may implement both the Relying Party Owner Role and the Relying Party Role directly.

## 5.2 Role Messages

Role messages consist of assertions about trustworthiness properties. Role messages flow between the various roles. The Actor exchanging a role message authenticates the message so that the Actor receiving the message can determine that the originator of the message is expected to perform the role, and so that message integrity is protected. The originator of the message ensures message veracity that the receiver verifies as part of the attestation trust model. Role messages consist of trustworthiness assertions, or Claims. Claims are explicitly realized in tag-value form or as an expression in a data definition language. Actors evaluate role message veracity according to the reputation or trust anchor of the entity asserting the claim.

### 5.2.1 Evidence

Evidence is a role message containing assertions, i.e., Claims, from the Attester. Evidence should contain freshness and recentness Claims that help establish Evidence relevance. For example, a Verifier supplies a nonce that can be included with the Evidence supplied by the Attester. Evidence typically describes the state of the device or entity. Normally, Evidence is collected in response to a request, i.e., challenge. Evidence may also describe historical device states, e.g., the state of the Attester during initial boot. It may also describe operational states that are dynamic and likely to change from one request to the next. Attestation protocols may be helpful in providing timing context for correct evaluation of Evidence that is highly dynamic.

If the Attesting Environment at layer N-1 collects claims about a Target Environment at layer N, a DICE layer ($L_{N-1}$) may supply Evidence about layer ($L_N$) in a certificate, dynamically issued by layer ($L_{N-1}$).

A Target Environment may assert Claims about itself or some other environment. Such claims are accepted if the Verifier accepts the Evidence about the Target Environment.

### 5.2.2 Appraisal Policy for Evidence

An Appraisal Policy for Evidence is an input to a Verifier that contains policies that reconcile trustworthiness Claims in Evidence with expected operational conditions involving the Attester.

### 5.2.3 Endorsements

Endorsement structures contain Assertions that are signed by an Actor performing the Endorser role. Endorsements are Endorsed Values and Reference Values that may be used by Verifiers when appraising Evidence.

A DICE layer ($L_{N-1}$) may supply Endorsements about layer ($L_N$) when Endorsement values are created by layer ($L_{N-1}$). For example, if layer N-1 randomizes layer N memory layout as part of loading an executable into memory and subsequently collects measurements for the randomized memory layout of layer N. Reference Values for layer N may be supplied by a layer (N-1).

#### 5.2.3.1 Endorsed Values

 

Endorsed Values are trustworthiness properties that are asserted by an Endorser that do not have matching Evidence. Endorsed Values derive from design, implementation, validation, and manufacturing processes applied to the Attester. The Endorser may claim the trustworthiness properties are immutable or intrinsic.

### 5.2.4 Attestation Results

Attestation Results are messages containing the results of attestation Evidence appraisals. Attestation Results may contain Claims or other application specific Assertions meaningful to the Relying Party. Attestation Results are authenticated, and integrity and confidentiality protected by the Verifier. Attestation Results from a Verifier are presumed to comply with Verifier Owner policies. Consequently, Attestation Results are actionable values in the context of the Relying Party.

### 5.2.5 Appraisal Policy for Attestation Results

An Appraisal Policy for Attestation Results is an input to a Relying Party that contains policies that reconcile trustworthiness claims in Attestation Results with expected operational conditions involving the Attester.

### 5.2.6 Message Freshness

The freshness of Role messages affects trustworthiness. The efficacy of trustworthiness properties can deteriorate over time or change after the collection and reporting of Evidence. For example, when an operational mode changes, or a configuration setting is applied, or environmental conditions change, or physical damage or wear occurs.

Message freshness may be achieved in the following ways:

a) Requester supplied nonce
b) Timestamp
c) Validity period

It may be necessary to include freshness claims as part of Evidence or in conveyance protocols.

## 5.3 Topology Models

Attestation message exchanges may occur according to a variety of stereotypical patterns. This section identifies several popular message exchange patterns.

### 5.3.1 Passport Model

The passport model illustrated in Figure 3 defines a sequence of message exchanges that fits a well-known pattern. The inspiration for the Passport Topology Model is government issued passports. The passport holder presents identity credentials to the passport issuer who constructs the passport document. The passport document contains markings or other factors that enables a third party to verify the authenticity of the passport document.

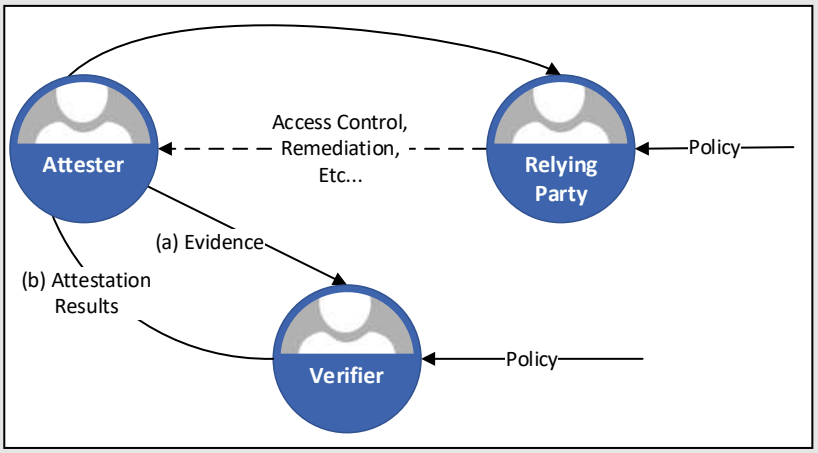


*Figure 3: Passport Topology Model*

 

The sequence of steps in the passport model for attestation consists of the following:

a) The Attester presents the Evidence message to a Verifier. The Verifier checks message freshness, integrity, and origin. It may be necessary for the Verifier to first provide a nonce to the Attester to guarantee freshness. The Assertions within the message are evaluated against a Policy that identifies Assertions that are acceptable, unacceptable, or unspecified by the Owner. The Verifier creates a Results message containing Assertions or other expressions that represent the attestation evaluation result. The results message is signed by the Verifier or otherwise contains a credential allowing a Relying Party to authenticate the Results from the Verifier.

b) Attestation Results are delivered to the Attester and later forwarded to the Relying Party. The Relying Party authenticates Results, originating from the Verifier, that were provided by the Attester. The Relying Party may verify freshness from both Attester and Verifier. The Relying Party processes the Results according to application defined actions.

In the case of a failed attestation, the Relying Party may need to take one or more implementation-specific actions, such as access control alerts, logging, and/or remediation. Protocols implementing the passport model may need to anticipate ways to perform implementation-specific actions as the conclusion of the previous sequence of steps.

### 5.3.2 Background Check Model

The background check model illustrated in Figure 4 defines a sequence of message exchanges that fits a background check pattern where the entity receiving credentials is unable to directly process them. Instead, they are processed by a backend entity.

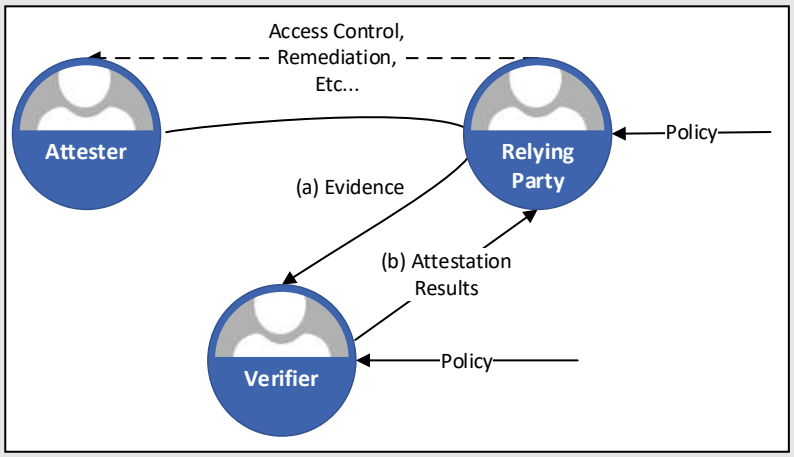


*Figure 4: Background Check Topology Model*

The sequence of steps in the background check model for attestation consists of the following:

a) The Attester presents the Evidence message to a Relying Party. The Relying Party checks message freshness, integrity, and origin. It may be necessary for the Relying Party to first provide a nonce to the Attester to guarantee freshness. The Relying Party forwards Evidence to the Verifier.

   The Verifier checks message freshness, integrity, and origin. It may be necessary for the Verifier to first provide a nonce to the Relying Party (which the Relying Party, in turn, provides to the Attester prior to Evidence collection) to guarantee freshness. The Verifier performs appraisal of Evidence as defined in step (a) of the passport model.

b) The Verifier delivers Results to the Relying Party. The Relying Party evaluates Results as defined in step (b) of the passport model.

In the case of a failed attestation, the Relying Party may need to take one or more implementation-specific actions, such as access control alerts, logging, and/or remediation. Protocols implementing the passport model may need to anticipate ways to perform implementation-specific actions as the conclusion of the previous sequence of steps.

 

### 5.3.3 Multi-party Background Check Model

The multi-party background check model illustrated in Figure 5 defines a sequence of message exchanges similar to the background check model except that there are multiple Relying Party entities involved.

The sequence of steps in the multi-party background model for attestation is the same as the steps for the background check model. The multi-party background check model differs in that the Attester forwards the Results to additional Relying Parties that each will evaluate the Results.

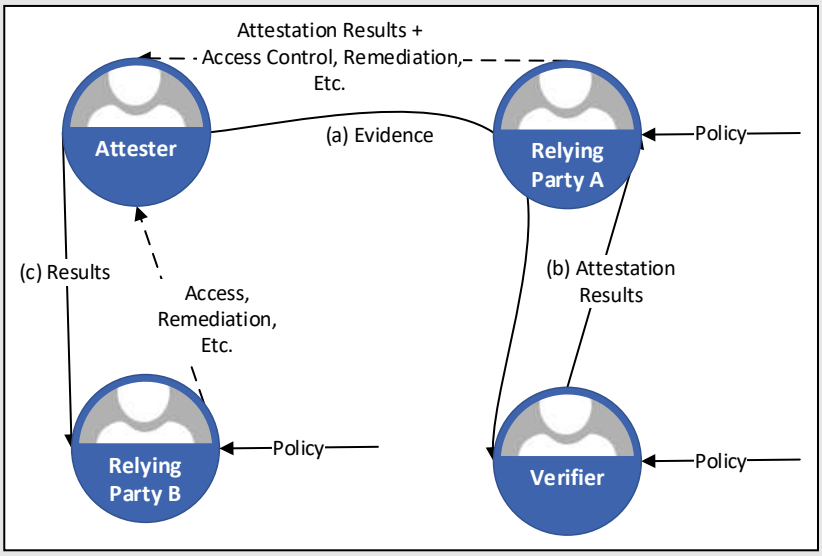


*Figure 5: Multi-Party Background Check Topology Model*

The role interactions described here, as well as others not explicitly illustrated, all preserve the basic pattern described by the Attestation Roles Architecture diagram (See §5.1). The other patterns described are additions to the basic pattern that do not alter the basic role function.

The topological models presented in this section are for illustrative purposes and are not intended to imply a limitation on the number of role interactions, nor their organization or complexity.

## 5.4 Assignment of Roles to Actors

Entities that implement Attestation Roles are known as Actors. There are many possible ways to assign roles to Actors. This section identifies common patterns involving role-actor combinations. Actor entities are the deployment environments that host and implement attestation roles (e.g., users, organizations, execution environments, service providers, servers, networks, devices, TEEs, DICE layers, Roots of Trust, etc.).

Actors implement interfaces or protocols used to convey role messages. Conveyance mechanisms are either local or remote. Local conveyance exists when the same Actor is used to perform multiple roles where role message conveyance is internal to that actor. Local conveyance means the protocols for authenticating, protecting, and transmitting role messages are trusted and opaque from the perspective of the co-resident roles. See Figure 6.

 

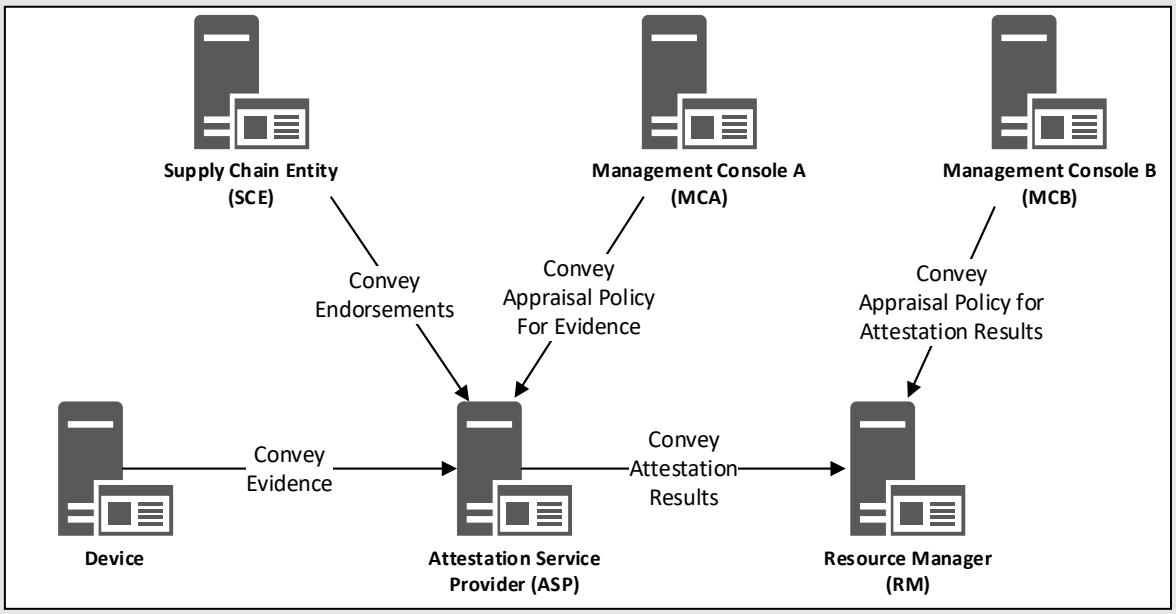


*Figure 6: Attestation Actors*

The Actor abstraction helps separate the operational elements of attestation from trust model elements.

Deployment architectures can differ significantly to address business, performance, geographic, or political and regulatory considerations. Actor names, credentials, and infrastructure often are reflective of the deployment architecture. For example, an Actor identified by organization name may be issued a certificate that is used to authenticate the Actor to another Actor. Their certifying infrastructures may be leveraged by attestation infrastructure to convey attestation information and to link roles to authentication credentials.

### 5.4.1 Role-Actor Composition

This section describes scenarios where two or more Actors are combined or co-located. The roles performed by discrete Actors are co-located but are not collectively considered a new hybrid role. Rather, they are recognized as separate roles being hosted by the same device, service, or entity. Actor composition semantics may also apply when virtual environments are dynamically instantiated. Both environments may exist on the same physical device yet have different actor contexts.

#### 5.4.1.1 Co-located Verifier and Relying Party Example

The Resource Manager Actor composition illustrated in Figure 7 co-locates an Attestation Service Provider (ASP) that normally performs the Verifier role with a Resource Manager that normally performs the Relying Party role. The user may dedicate a single server, multiple servers inside a private network, or outsource to a cloud services provider for hosting both roles. Verifier and Relying Party role message interactions have local conveyance properties.

 

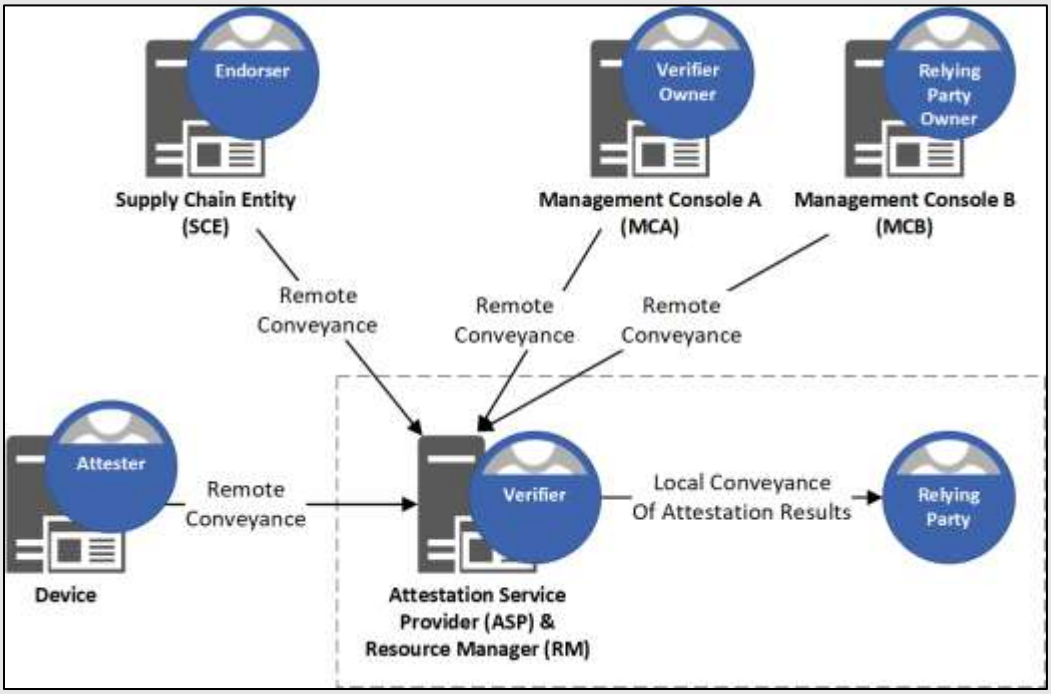


*Figure 7: Role-Actor Composition – Combined Verifier and Relying Party Example*

#### 5.4.1.2 Composite Attestation Example

In a Composite Device attestation scenario, components have attestation capabilities that generate Evidence. Evidence is conveyed locally to a Composite Attester that assembles the various sets of Claims. The Evidence might also include Claims the Composite Attester directly collects or provides. The Composite Attester conveys Evidence to a remote service provider that hosts a Verifier. Figure 8 provides an illustration of this example.

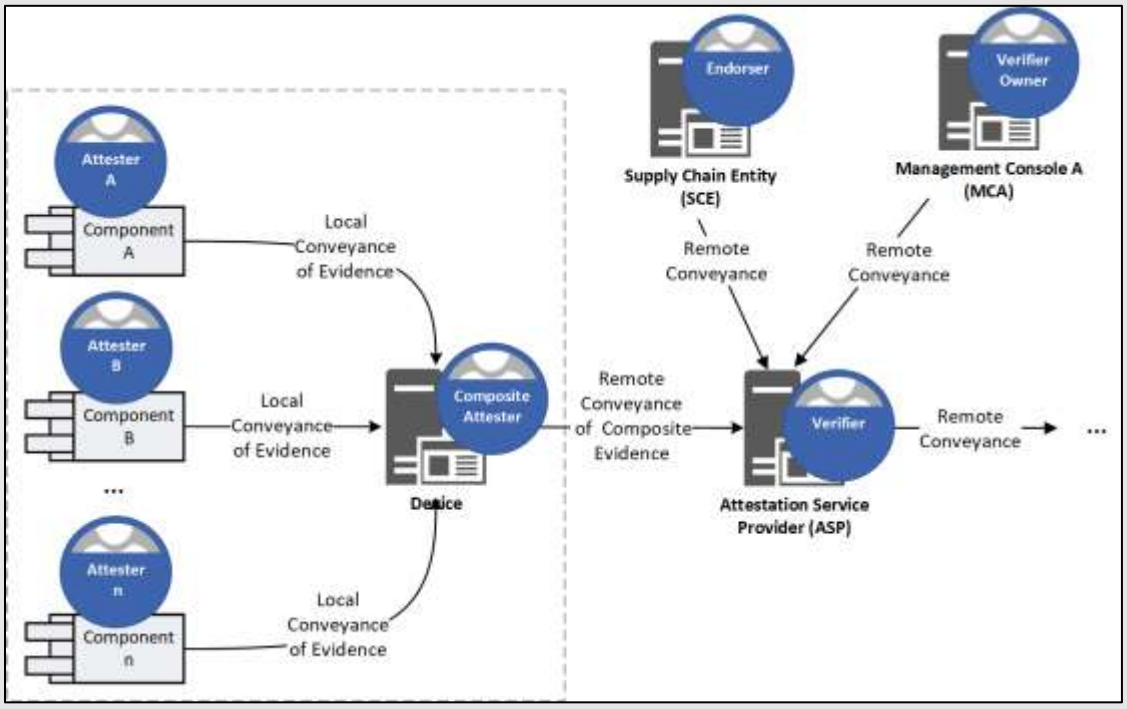


*Figure 8: Role-Actor Composition – Composite Device Attestation Example*

#### 5.4.1.3 Local Verifier Example

In a local Verifier scenario, local components are Attesters that use a local conveyance mechanism to deliver Evidence to the local Verifier for appraisal. The appraisal becomes Attestation Results that are conveyed to a remote Resource Manager that hosts the Relying Party. This example, illustrated in Figure 9, shows the local Verifier having a Local Verifier Owner, so appraisal policies are locally conveyed. The local Verifier relies on Endorsements from a supply chain entity that are remotely conveyed.

 

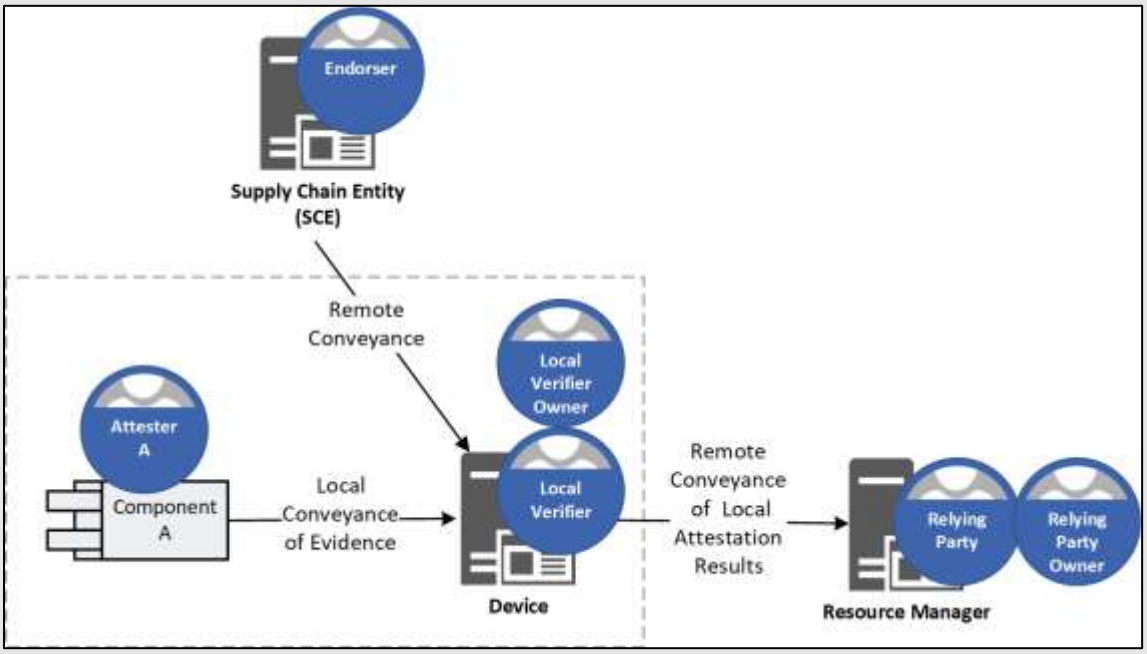


*Figure 9: Role-Actor Composition – Local Verifier Example*

#### 5.4.1.4 Layered Device Attestation Example

In a layered device attestation scenario, a set of layered components each attest the state of the next component. Evidence from each layer is verified by a remote Verifier using an attestation service: therefore, most Evidence is protected for remote appraisals but is conveyed locally. A layered Attester identifies the next layer Attester designated to convey its Evidence. The designation becomes part of the Evidence it produces. This is illustrated in Figure 10.

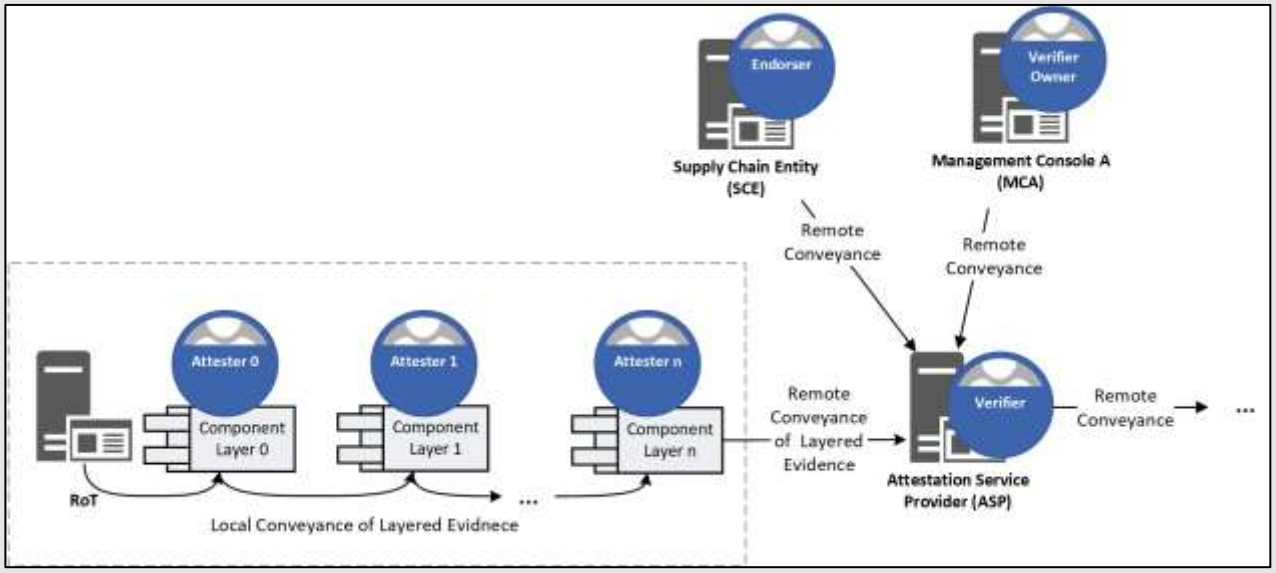


*Figure 10: Role-Actor Composition – Layered Attester Example*

### 5.4.2 Actor Composition Summary

Actor composition allows flexibility when determining which Roles an Actor may perform. When multiple Roles are performed by the same Actor, Roles do not interfere with each other.

 

Local conveyance of Role information must be trustworthy, but its definition is out-of-scope for this specification as it is implementation-specific. Remote conveyance expects that Role information will be communicated via untrusted transports and therefore needs to be protected. Protocol binding specifications are needed to address specific threats.

The examples presented in this specification are for illustrative purposes and are not intended to imply a limitation on the number, organization, or complexity of Role-Actor compositions.

**End of informative comment**

# 6 Layered Device Attestation

Layered attestation refers to the traversal of DICE layers where the current layer attests to the state of the next layer. Evidence about the next layer is signed by the current layer. Trust in a current DICE layer depends on the trustworthiness of all previous layers.

**Start of informative comment**

Consequently, a Verifier of layered attestation evaluates attestation Evidence of the dependent layers before it can reason about trust in the current layer.

Verifiers recognize when an Attester is performing layered attestation. Inclusion of attestation Evidence in a certificate extension issued to a DICE layer by a previous DICE layer helps a Verifier to deduce the presence of layered attestation. The Verifier of a layered attestation always processes this certificate extension if it is supplied.

**End of informative comment**

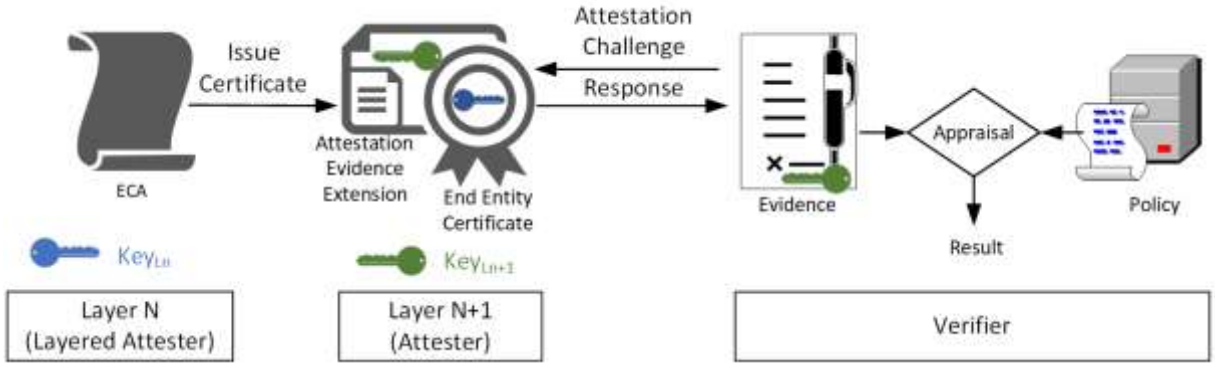


*Figure 11: Layered Attestation*

Attestation Verifiers require attestation Evidence. There are several possible techniques for conveying Evidence to a Verifier. This specification specifies the following approaches:

(i) X.509 identity certificates and certificate revocation lists (CRLs) that contain Evidence
(ii) X.509 attribute certificates containing Evidence
(iii) Manifests containing Evidence.

## 6.1 Evidence as X.509 Certificate Extensions

This section defines X.509v3 certificate and CRL extensions. These extensions encode reference Endorsements about a Target Environment. Certificates containing these extensions are RFC5280 [7] compliant.

Certificate revocation involving a DICE layer can benefit from the added context that attestation Evidence provides. Certificate Evidence extensions can be used with certificate revocation lists. Consequently, it may be appropriate to include Evidence extension in CRLs.

A certificate issuer uses an extension to assert the trustworthiness claims that apply to the Attesting Environment that protects the subject private key that is identified by the certificate subject public key. When the certificate is presented to a Verifier, the reference Endorsements are available for trustworthiness evaluation.

A CRL issuer uses an extension to assert that these trustworthiness claims apply to the Attesting Environment that protects the subject private key that is identified by the certificate serial number that identifies the certificate that identifies the subject public key. A Verifier, having the CRL, may use Endorsements contained in the extension to evaluate the TCB properties associated with the revocation request. Endorsements in a CRL describe claims that are no longer trustworthy.

 

The following extensions use an OID arc from the TCG namespace.

```
TCG Arc: tcg OBJECT IDENTIFIER ::= {2 23 133}
DICE Arc: tcg-dice OBJECT IDENTIFIER ::= { tcg platformClass(5) 4 }
```

**Start of informative comment**

It is recommended that Verifiers process all Evidence extensions defined by this specification if present in a certificate, to ensure the Target Environment is trustworthy.

**End of informative comment**

### 6.1.1 TCB Info Evidence Extension

This extension defines attestation Evidence about a Target Environment that is measured by an Attesting Environment that controls the Subject key. The certificate `Subject` and `SubjectPublicKey` MAY identify the entity (a.k.a., Target Environment) to which the `DiceTcbInfo` extension applies. When this extension is used, the measurements in Evidence usually describe software or firmware that will execute within the Target Environment.

The `AuthorityKeyIdentifier` extension MUST be supplied when the `DiceTcbInfo` extension is supplied. This allows the Verifier to locate the signer's certificate.

**Start of informative comment**
Inclusion of the `DiceTcbInfo` extension is optional. However, if omitted, an alternative method for conveying the `DiceTcbInfo` information to the Verifier needs to be provided.
**End of informative comment**

The `DiceTcbInfo` extension SHOULD be marked critical. The `DiceTcbInfo` extension SHOULD be included with CRL entries that revoke the certificate that originally included the `DiceTcbInfo` extension.

The `DiceTcbInfo` OID is as follows:

```
tcg-dice-TcbInfo OBJECT IDENTIFIER ::= {tcg-dice 1}
```

The `DiceTcbInfo` fields are:

```
DiceTcbInfo ::== SEQUENCE {
      vendor      [0] IMPLICIT UTF8String           OPTIONAL,
      model       [1] IMPLICIT UTF8String           OPTIONAL,
      version     [2] IMPLICIT UTF8String           OPTIONAL,
      svn         [3] IMPLICIT INTEGER              OPTIONAL,
      layer       [4] IMPLICIT INTEGER              OPTIONAL,
      index       [5] IMPLICIT INTEGER              OPTIONAL,
      fwids       [6] IMPLICIT FWIDLIST             OPTIONAL,
      flags       [7] IMPLICIT OperationalFlags     OPTIONAL,
      vendorInfo  [8] IMPLICIT OCTET STRING         OPTIONAL,
      type        [9] IMPLICIT OCTET STRING         OPTIONAL,
      flagsMask   [10]IMPLICIT OperationalFlagsMask OPTIONAL
}
```

*Name Fields:*

- *vendor* – the entity that created the measurement of the Target Environment (e.g., a TCI value).
- *model* – the product name associated with the measurement of the Target Environment.
- *layer* – the DICE layer associated with this measurement of the Target Environment.
- *index* – a value that distinguishes different instances of the same type of Target Environment.
- *type* – a machine readable description of the measurement.

Measurement Fields:

- *version* – the revision string associated with the Target Environment.
- *svn* – the security version number associated with the Target Environment.
- *fwidlist* – a list of FWID values. FWIDs are computed by the DICE layer that is the Attesting Environment and certificate Issuer. Generally, construction and evaluation of a FWID list is defined by Reference Values.

```
FWIDLIST ::== SEQUENCE SIZE (1..MAX) OF FWID
FWID ::== SEQUENCE {
          hashAlg          OBJECT IDENTIFIER,
          digest           OCTET STRING
}
```

- *hashAlg* – an algorithm identifier for the hash algorithm used to produce a digest value. The algorithm identifier MUST match the object identifier used in the RIM containing the Reference Values.

  **Start of informative comment**

  Note: algorithm identifiers are not necessarily limited to those defined by TCG.

  **End of informative comment**

- *digest* – a digest of firmware, initialization values, or other settings of the Target Environment.
- *flags* – a list of flags that enumerate potentially simultaneous operational states of the Target Environment (see §6.1.1.1).
- *vendorInfo* – vendor supplied values that encode vendor, model, or device specific state.

When filling in the `DiceTcbInfo` extension, the issuer (layer N) must ensure that any field that contributes to the CDI that generates the subject key (such that a change in the value will cause a change in the CDI) is included in a field of the `DiceTcbInfo` extension.

**Start of informative comment**

Constant values, i.e., values that are physically unchangeable on the device, need not be included in measurements and, therefore, need not be included in a `DiceTcbInfo` extension.

**End of informative comment**

The Verifier queries a database containing Endorsements that correspond to this Evidence using a combination of any fields from the `DiceTcbInfo`.

**Start of informative comment**

For example, the Verifier could query by digest or by vendor, model, type and version values. The vendor, model, type, layer and index fields of the `DiceTcbInfo` extension describe the measurement and are chosen by the entity performing the measurement. The remaining fields of the `DiceTcbInfo` extension describe properties of the entity being measured.

Endorsements describe how a *digest* is computed.

Note: both Verifier and Attesting Environments need to consistently apply the *digest* computation method.

**End of informative comment**

#### 6.1.1.1 Operational Flags

Operational flags are Evidence claims that, when collected, describe environmental attributes affecting trustworthy operation. For each mode, the Attesting Environment (e.g., layer N-1) determines whether the Target Environment (layer N) currently has, or will have, one or more properties.

```
OperationalFlags ::= BIT STRING {
      notConfigured (0),
      notSecure (1),
      recovery (2),
      debug (3),
      notReplayProtected (4),
      notIntegrityProtected (5),
      notRuntimeMeasured (6),
      notImmutable (7),
      notTcb (8),
      fixedWidth (31)
}
```

- `notConfigured` – The Target Environment is not configured for normal operation.
- `notSecure` – The Target Environment is insecure.
- `recovery` – The Target Environment is recovering (e.g., from a failure).
- `debug` – The Target Environment can be debugged.

**Start of informative comment**

The specific debug resources that may be accessed are environment specific. The Target Environment and system software typically determine which debug resources are accessible.

**End of informative comment**

 

- `notReplayProtected` – The Target Environment is vulnerable to replay attack.
- `notIntegrityProtected` – The Target Environment is vulnerable to modification by unauthorized updates.
- `notRuntimeMeasured` – The Target Environment is not measured after being loaded into memory.
- `notImmutable` – The measured Target Environment is mutable.
- `notTcb` – The Target Environment measurements are not measurements of a Trusted Computing Base (TCB).
- `fixedWidth` – This field (bit 31) is used to force a fixed width for the `OperationalFlags` bit string, regardless of which bits may be unused. Setting the `fixedWidth` bit will ensure the `OperationalFlags` field is 32 bits in length and, therefore, the resulting encoding will always contain a length of four (4) bytes.

**Start of informative comment**

An ASN.1 BIT STRING length is determined by the highest order bit that is SET. Some implementations benefit from a predictable fixed size encoding for X.509 certificates. Setting the `fixedWidth` bit in `OperationalFlags` guarantees a 32-bit `OperationalFlags` BIT STRING length which, in turn, guarantees the underlying encoding of this field will always be three (4) bytes in length.

As its purpose is only to guarantee that the encoding of `OperationalFlags` is of a fixed width, the `fixedWidth` bit is of no evidentiary value, it is not needed in Endorsements, and is otherwise ignored.

**End of informative comment**

A value of 1 (SET) in an `OperationalFlags` bit means the corresponding mode is active. A value of 0 (CLEAR) means the corresponding mode is not active. If the corresponding mask bit (see section 6.1.1.2) is CLEAR, the mode is unknown.

Operational flags can be incorporated into attestation Evidence in several ways:

a) An operational flag might indicate that different firmware is used when in an alternative mode. The firmware digest values may differ, according to the firmware used, resulting in a different CDI value. The Reference Values manifest for the Target Environment image MUST specify which operational flags are allowed in an alternative mode by defining an `OperationalFlagsMask` mask for the alternative mode.
b) The operational flag MAY be reported using the `DiceTcbInfo.flags` bits. Hence a CDI value could be unintuitively different when operating in an alternative mode despite the firmware being the same.
c) The operational flag MAY be reported using `DiceTcbInfo.vendorInfo`.
d) The operational flags MAY be attested using an Evidence format other than `DiceTcbInfo`.
e) The operational flags MAY represent something other than normal operation and, as a result, the Attesting Environment (layer n-1) MAY withhold the CDI for the Target Environment (layer n). If the Target Environment attempts to obtain its CDI from the Attesting Environment, the Attesting Environment MUST generate a fault to indicate an abnormal Target Environment operation: and that none of options (a), (b), (c), or (d) are in use.

This specification does not define whether combinations of modes are mutually exclusive. The vendor of the Target Environment SHOULD incorporate that information when defining Endorsed Values, Reference Values, and Evidence.

When both `OperationalFlags` and `OperationalFlagsMask` is provided, each `OperationalFlags` bit, except for `fixedWidth`, MUST be ignored, regardless of value, unless the corresponding bit position within the `OperationalFlagsMask` bit string is SET.

If `OperationalFlags` is provided but `OperationalFlagsMask` is not provided, then each `OperationalFlags` bit, except for `fixedWidth`, SHALL be interpreted as if the corresponding `OperationalFlagsMask` bit is SET.

 

**Start of informative comment**

The `fixedWidth` bit was not present and was unnecessary for previous versions of this specification because, regardless of how operational flags are set, the resultant length of the `OperationalFlags` encoding was constant.

**End of informative comment**

#### 6.1.1.2 Operational Flags Mask

The `OperationalFlagsMask` bit string is used to communicate which bits within the `OperationFlags` bit string are meaningful. When a bit is SET in the `OperationalFlagsMask` bit string, the bit at the corresponding position in `OperationalFlags`, apart from the `fixedWidth` bit (see section 6.1.1.1), is meaningful to a Verifier.

The `OperationalFlagsMask` bits directly correspond to the bits defined within the `OperationalFlags` bit string. The definition for `OperationalFlagsMask` is as follows:

```
OperationalFlagsMask ::= BIT STRING {
      notConfigured (0),
      notSecure (1),
      recovery (2),
      debug (3),
      notReplayProtected (4),
      notIntegrityProtected (5),
      notRuntimeMeasured (6),
      notImmutable (7),
      notTcb (8),
      fixedWidth (31)
}
```

If a bit in the `OperationalFlagsMask` bit string is SET, then the bit in the corresponding position in the `OperationalFlags` bit string SHALL be interpreted, irrespective of value.

If a bit in the `OperationalFlagsMask` bit string is CLEAR, then the bit in the corresponding position in the `OperationalFlags` bit string SHALL be ignored, irrespective of value.

For example, if `OperationalFlagsMask.debug` is SET then if `OperationalFlag.debug` is CLEAR, it is interpreted as an assertion that debug mode is not active. However, if `OperationalFlagMask.debug` is CLEAR then the `OperationalFlags.debug` bit has no meaning regardless of its value and is ignored.

See section 6.1.1.1 for a description of the `fixedWidth` bit. It has the same purpose and meaning within the `OperationalFlagsMask` field as `fixedWidth` does for `OperationalFlags`.

#### 6.1.1.3 DiceTcbInfoAlias

**Start of informative comment**

This section defines an extension identical to `DiceTcbInfo` but with an alternative identifier, as an option for legacy implementations that use the `DiceTcbInfo` OID `{tcg-dice 1}` in a vendor-specific manner that does not conform to the `DiceTcbInfo` definition in section 6.1.1.

Prior to the definition of `DiceTcbInfo`, some implementations had used the `{tcg-dice 1}` identifier to reference vendor-defined extension formats. In such cases, Verifiers need to detect vendor-specific (non-conforming) implementations through a vendor-defined mechanism. Vendors with such implementations are responsible for notifying Verifiers of such non-conforming implementations and for providing the criteria by which these non-conforming implementations can be detected (for example, by checking the certificate issuer for O=VENDOR).

Vendors with non-conforming uses of `{tcg-dice 1}` are recommended to use the `DiceTcbInfoAlias` extension for all conforming implementations so that Verifiers are required to provide special handling only when both OID={tcg-dice 1} AND vendor criteria is met.

**End of informative comment**

`DiceTcbInfoAlias` is identical to `DiceTcbInfo`, except as noted here:

```
tcg-dice-TcbInfoAlias OBJECT IDENTIFIER ::= {tcg-dice-TcbInfo 1}
```

The `DiceTcbInfoAlias` extension criticality flag SHOULD be marked critical.

The `DiceTcbInfoAlias` extension SHOULD NOT be used when either the `DiceTcbInfo` or `DiceTcbInfoSeq` can reasonably be used.

### 6.1.2 Multiple DiceTcbInfo Structures Extension

The initial state of a Target Environment may be represented by multiple measurements, for example, when it is composed of elements supplied by different vendors or when other inputs (for example fuses) that affect the functionality of the Target Environment need to be measured. This certificate extension defines a sequence of `DiceTcbInfo` structures, one for each measurement.

The declaration of `DiceMultiTcbInfo` is as follows:

```
tcg-dice-MultiTcbInfo OBJECT IDENTIFIER ::= {tcg-dice 5}
DiceTcbInfoSeq ::= SEQUENCE SIZE (1..MAX) OF DiceTcbInfo
```

Use of both the `DiceTcbInfo` and `DiceTcbInfoSeq` extensions independently as top-level entities is not recommended. However, if both are used, the `DiceTcbInfo` extension SHALL be treated as the first element of the list of `DiceTcbInfo` structures contained in `DiceTcbInfoSeq`.

The `DiceTcbInfoSeq` extension criticality flag SHOULD be marked critical.

### 6.1.3 Compression Extension for Multiple DiceTcbInfo Structures

**Start of informative comment**

Fields of a `DiceTcbInfo` structure that are repeated for each entry in a sequence can be compressed using the `DiceTcbInfoComp` extension. This certificate extension compresses a `DiceTcbInfoSeq` by extracting the elements of a `DiceTcbInfoSeq` that would be repeated within each `DiceTcbInfo` structure, and instead including the repeated fields only once in a single `DiceTcbInfo` structure, in `commonFields`. The entries within the `DiceTcbInfoSeq` provided in `evidenceValues` do not contain these repeated fields.

Including a field within the `commonFields` structure causes the `evidenceValues` sequence to be interpreted as if each field in `commonFields` is also part of each structure in the `evidenceValues` sequence.

**End of informative comment**

The OID declaration is as follows:

```
tcg-dice-MultiTcbInfoComp OBJECT IDENTIFIER ::= {tcg-dice 8}
```

The ASN.1 definition is as follows:

```
DiceTcbInfoComp ::= SEQUENCE SIZE (1..MAX) OF TcbInfoComp
TcbInfoComp ::= {
   commonFields [0] IMPLICIT DiceTcbInfo,
   evidenceValues [1] IMPLICIT DiceTcbInfoSeq
```

 

```
}
```

The `DiceTcbInfo` extension in `commonFields` SHALL comprise all fields that are common to every entry within the `DiceTcbInfoSeq` sequence in `evidenceValues`.

Every `DiceTcbInfo` extension in the `DiceTcbInfoSeq` sequence in `evidenceValues` MUST be different to any `DiceTcbInfo` extension in `commonFields`.

When decompressing, the `DiceTcbInfo` extension in `commonFields` SHALL be prepended to every `DiceTcbInfo` extension in the `DiceTcbInfoSeq` sequence in `evidenceValues`.

### 6.1.4 UEID Extension

This extension contains a UEID [10] that identifies the device containing the private key and is identified by the certificate's subjectPublicKey. In the case of its inclusion as a CRL extension, the device containing the private key is identified by the certificate serial number, which identifies the certificate containing the subjectPublicKey.

The OID declaration of `DiceUeid` is as follows:

```
tcg-dice-Ueid OBJECT IDENTIFIER ::= {tcg-dice 4}
```

The ASN.1 definition is as follows:

```
TcgUeid ::= SEQUENCE {
      ueid OCTET STRING
}
```

When filling in the UEID extension, the issuer must ensure that the content of this extension contributes to the CDI which generated the subject key (such that a change in the field value will cause a change in the CDI).

### 6.1.5 CWT Claims Set Evidence Extension

The CBOR Web Token (CWT) specification [11] defines a CBOR encoding of a claim set that may be used to contain Evidence. A variant of CWT that does not contain integrity protection, unprotected CWT Claims Set (UCCS) defines a certificate Evidence extension containing UCCS formatted Evidence.

The `tcg-dice-UCCS-evidence` is an unsigned structure because the certificate signature authenticates, and integrity protects UCCS contents.

The OID declaration of `DiceUccsEvidence` is as follows:

```
tcg-dice-UCCS-evidence OBJECT IDENTIFIER ::= {tcg-dice 6}
```

The ASN.1 definition is as follows:

```
UccsEvidence ::= SEQUENCE {
      uccs OCTET STRING
}
```

This extension MAY be used in addition to or in place of `DiceTcbInfoSeq` or `DiceTcbInfo`.

When filling in the UCCS extension, the issuer MUST ensure that this field contributed to the CDI that generated the subject key (such that a change in the field value will also reflect a change in the CDI).

The `DiceUccsEvidence` extension criticality flag SHOULD be marked critical.

### 6.1.6 Manifest Evidence Extension

A SWID or CoSWID manifest may be used to contain Evidence.

The OID declaration of `DiceManifestEvidence` is as follows:

```
tcg-dice-manifest-evidence OBJECT IDENTIFIER ::= {tcg-dice 7}
```

The `tcg-dice-manifest-evidence` object identifier is used with the `Manifest` sequence (See §6.5.1.1). The extension SHOULD contain information equivalent to `DiceTcbInfoSeq` or `DiceTcbInfo` sequences.

`The tcg-dice-manifest-evidence` MUST use an unsigned manifest because the certificate signature authenticates, and integrity protects, manifest contents.

When filling in the manifest Evidence extension, the issuer MUST ensure that this field contributed to the CDI that generated the subject key (such that a change in the field value will also reflect a change in the CDI).

The `DiceManifestEvidence` extension criticality flag SHOULD be marked critical.

### 6.1.7 AuthorityKeyIdentifier Certificate Extension

If the `AuthorityKeyIdentifier` extension is supplied, the `keyIdentifier` must identify the `Issuer` public key.

### 6.1.8 Conceptual Message Wrapper Extension

**Start of Informative Comment**

The conceptual message wrapper extension is used to convey a *conceptual message*, such as Evidence or Attestation Results [12], in an X.509 certificate extension. Typically, X.509 extensions are described using an ASN.1 encoding format, but other encapsulation formats may be used. For example, [13] defines a message wrapping structure that may be encoded using CBOR or JSON.

This section defines a Conceptual Message Wrapper (CMW) certificate extension that uses a CBOR or JSON encoding of a type-value array containing a content type identifier, a conceptual message, and a conceptual message type. The CMW content may contain multiple conceptual messages of varying types. The conceptual message type structure enumerates each conceptual message type contained in the CMW content.

A conceptual message may also have a CBOR tag [8] that encodes the message type followed by the conceptual message. This certificate extension supports all three forms of conceptual messages: CBOR encoded CMW array, JSON encoded CMW array, and CBOR tagged conceptual message.

Conceptual messages may be used by Certificate Authorities (CA) when issuing new certificates or refreshing existing certificates. The conceptual message wrapper extension may be useful to certificate enrollment requests as described in [14].

Conceptual message may be used by Web authentication protocols that rely on public key credentials such as X.509 certificates. [15] describes a Web API for exchanging public key credentials containing attestation conceptual messages.

**End of Informative Comment**

The OID declaration of `DiceConceptualMessageWrapper` is as follows:

```
tcg-dice-conceptual-message-wrapper OBJECT IDENTIFIER ::= {tcg-dice 9}
```

The ASN.1 definition is as follows:

```
 ConceptualMessageWrapper ::= SEQUENCE {
    cmw OCTET STRING
}
```

 

The `ConceptualMessageWrapper` extension MAY be used in addition to or in place of `DiceTcbInfoSeq` or `DiceTcbInfo` extensions.

The `ConceptualMessageWrapper` sequence SHALL contain an OCTET STRING containing a CBOR, JSON, or tagged CBOR encoded conceptual message wrapper in either the array form, see §3.1 of [13], or the tagged CBOR form, see [8], [16], and [13].

The `ConceptualMessageWrapper` sequence contents, in the array form, are described by the following CDDL:

```
cmw = [ type, value, ? bytes .bits cm-type ]
type = coap-content-format / media-type
coap-content-format = uint .size 2
media-type = text .abnf ("media-type" .cat RFC6838)
value = cbor-bytes / base64-string
cbor-bytes = bytes
base64-string = text .regexp "[A-Za-z0-9_-]+"
cm-type = &( reference-values: 0,
 endorsements: 1,
 evidence: 2,
 attestation-results: 3
)
```

When filling in the `tcg-dice-conceptual-message-wrapper` extension, the issuer MUST ensure that this Evidence extension contributed to the CDI that generated the certificate's subject key (such that a change in the Evidence will reflect a change in the CDI).

The `tcg-dice-conceptual-message-wrapper` extension criticality flag SHOULD be marked critical.

Inclusion of the `tcg-dice-conceptual-message-wrapper` extension is OPTIONAL.

**Start of Informative Comment**

Refer to [17] and [12] for guidance on text encoding constraints that are applied to the `media-type` statement.

The conceptual message wrapper in the array form, see [13], is used when the conceptual message type has been registered as a `media-type` [17] or as a `coap-content-format` [18]. The CBOR tag form, see [12], is used when the conceptual message type has been registered as a CBOR tag, see [8], or when a CBOR tag is derived from a `coap-content-format` using the TN() transform as defined in [16].

The `ConceptualMessageWrapper` sequence contents can be encoded as JSON, CBOR, or tagged CBOR. A parser decodes the octet string into a byte buffer and then does a 1-byte lookahead, as illustrated in the following pseudo code, to decide which format to use to decode the remainder of the byte buffer:

```
switch b[0] {
case 0x82:
        return CBORArray
case 0x5b:
        return JSONArray
default:
        return CBORTag
}
```

When the conceptual message wrapper extension is marked critical, the recipient is expected to fully parse and process the Evidence, including CBORArray, JSONArray or CBORTag contents.

**End of Informative Comment**

 

## 6.2 CRL Extensions

The Evidence extensions defined above MAY be included as a certificate revocation list (CRL) extension.

**Start of informative comment**

If `DiceTcbInfo` or `DiceTcbInfoSeq` extensions are present in a CRL entry, then the Verifier needs to use a `DiceTcbInfo` for verification instead of verifying against issuer and serialNumber fields as normal. If a match condition is found, the Verifier considers the Target Environment invalid.

If a traditional revocation is needed, the CRL is issued with serialNumber only (omitting `DiceTcbInfo` or `DiceTcbInfoSeq`). Revoking a certificate containing Evidence extensions (e.g., `DiceTcbInfo` or `DiceTcbInfoSeq`) also invalidates the Evidence contained within the revoked certificate.

a) A certificate is revoked if a `DiceTcbInfo` entry in the CRL matches the `DiceTcbInfo` or a subset of a `DiceTcbInfoSeq` in the certificate (i.e., `DiceTcbInfo` in CRL = `DiceTcbInfo` in Certificate or `DiceTcbInfo` in CRL ⊆ `DiceTcbInfoSeq` in Certificate).

b) A certificate is revoked if a `DiceTcbInfoSeq` in the CRL matches a subset of the `DiceTcbInfoSeq` in the certificate (i.e., `DiceTcbInfoSeq` in CRL ⊆ `DiceTcbInfoSeq` in Certificate).

**End of informative comment**

## 6.3 Evidence as an X.509 Attribute Certificate

Evidence might be created using an X.509 attribute certificate that is signed by an attestation key. Evidence is collected about a Target Environment by the Attesting environment that controls the attestation signing key.

Attribute certificate structures that contain Evidence SHOULD include a caller-supplied freshness nonce.

The OID declaration of `DiceTcbFreshness` is as follows:

```
tcg-dice-TcbFreshness OBJECT IDENTIFIER ::= {tcg-dice 11}
```

The ASN.1 definition is as follows:

```
DiceTcbFreshness::= SEQUENCE {
    nonce OCTET STRING
}
```

The `DiceTcbFreshness` extension SHOULD be marked critical.

## 6.4 Evidence as a Manifest

Evidence might be created using a manifest that is signed by an attestation key. Evidence is collected about a Target Environment by the Attesting Environment that controls the attestation signing key.

## 6.5 Endorsements

Attestation Verifiers require attestation Endorsements. Endorsers (i.e., manufacturers and suppliers) create Endorsements that contain Endorsed Values and Reference Values. Reference Values are used by a Verifier to appraise Evidence. Endorsements may also contain Endorsed Values that are assertions that are not matched with Evidence, but are associated with the Attester, and may be used by an Appraisal Policy.

This specification specifies the following approaches for encoding Endorsements:

(i) X.509 identity certificate extensions.

 

(ii) X.509 attribute certificates.
(iii) Manifests, e.g., CoRIM, SWID.

Including Endorsements with an identity certificate adds a manufacturing constraint that Endorsed Values, Reference Values and certificate public keys must all be known at the time the certificate is issued.

## 6.5.1 Endorsements as X.509 Certificate Extensions

### 6.5.1.1 Manifest as an X.509 Certificate Extension

**Start of informative comment**

X.509 certificates [7] support extensions that may contain attestation manifests. The `DiceEndorsementManifest` certificate extension contains a manifest structure that is signed by an Endorser. The manifest may contain Endorsed Values and Reference Values about one or more Target Environments. The manifest can be used by a Verifier to appraise Evidence, for example, `DiceTcbInfo`.

The certificate signature provides integrity protection of the `DiceEndorsementManifest` contents. The certificate signer may not be the originator of the manifest. If so, the Endorser should integrity protect the manifest before including it in the certificate (see [22]).

**End of informative comment**

The OID declaration of `DiceEndorsementManifest` is as follows:

```
tcg-dice-endorsement-manifest OBJECT IDENTIFIER ::= {tcg-dice 2}
```

The ASN.1 definition is as follows:

```
Manifest ::== SEQUENCE {0
      format      ManifestFormat,
      manifest    OCTET STRING,
}
ManifestFormat ::= ENUMERATED {
      swid-xml                (0),
      coswid-cbor       (1),
      coswid-json       (2),
      tagged-cbor       (3)
}
```

The `Manifest` Format fields are:

- *format* – defines the manifest schema and encoding format:
  - *swid-xml* – The manifest format is XML [19] and contains a SWID Tag manifest as defined by [20].
  - *coswid-cbor* – The manifest format is CBOR [8] and contains a CoSWID manifest as defined by [9].
  - *coswid-json* – The manifest format is JSON [21] and contains a CoSWID manifest.
  - *tagged-cbor* – The manifest format is CBOR [8] and contains a manifest as defined by a CBOR tag (e.g., #6.xxx(bytes). CBOR tags are assigned by the IANA [21] registry.
- *manifest* – a signed or unsigned manifest containing Endorsed Values or Reference Values about a Target Environment.

Inclusion of the tcg-dice-endorsement-manifest extension is OPTIONAL.

### 6.5.1.2 Endorsement URI as an X.509 Certificate Extension

**Start of informative comment**

 

This extension contains a URI that locates a manifest. The manifest contains Endorsed Values or Reference Values about one or more Target Environments. The manifest can be used by a Verifier to appraise Evidence such as `DiceTcbInfo`.

**End of informative comment**

The OID declaration for `DiceEndorsementManifestUri` is as follows:

```
tcg-dice-endorsement-manifest-uri OBJECT IDENTIFIER ::= {tcg-dice 3}
```

The ASN.1 definition is as follows:

```
EndorsementManifestURI ::== SEQUENCE {
      emUri       UTF8String,
}
```

The `DiceEndorsementManifestUri` field is:

- *emUri* – is a universal resource identifier [22] that contains an object reference to a manifest. For example, a SWID Tag schema contains a 'tagId' attribute that may be encoded in an emURI.

### 6.5.2 Endorsements Using X.509 Attribute Certificates

**Start of informative comment**

X.509 attribute certificates [23] may contain attribute values that endorse an Attester.

**End of informative comment**

Attribute certificate structures, other than the Endorsement certificate extensions defined here, that contain Endorsements, are outside the scope of this specification.

### 6.5.3 Endorsements Using Stand-alone Manifests

Endorsement manifests are any authenticatable data structure that contains Endorsements, and typically rely on a schema that defines syntactic and semantic constraints that apply to manifest construction, parsing and processing.

## 6.6 Attestation Results

**Start of informative comment**

Attestation Verifiers generate Attestation Results that may be conveyed to a Relying Party.

**End of informative comment**

### 6.6.1 Attestation Results as X.509 Certificate Extensions

**Start of informative comment**

Attesters may produce multiple instances of Evidence to completely attest a device. Some Attestation Results may be conveyed indirectly to a Relying Party via the Attester entity. Attestation Results may be forwarded, via the Attester, to a Relying Party. Attestation Results are integrity protected by the Verifier but may rely on the Attester for conveyance / forwarding. An X.509 certificate may contain Attestation Results.

**End of informative comment**

The `tcg-dice-conceptual-message-wrapper` may be used to convey Attestation Results within an X.509 certificate extension.

 

# 7 Attesting Environment

This section provides additional guidance, requirements, and design considerations for Attesting Environments.

## 7.1 Compound Device Identifiers

The Attesting Environment (i.e., the issuer of Evidence) MUST ensure that each field that has contributed to a corresponding CDI value appears in Evidence. If a CDI value of an Attester changes, then at least one Evidence field has also changed, and vice versa. This ensures consistency between what is asserted as Evidence and actual conditions described by Evidence.

When generating attestation keys, if the subject key is not derived or generated using the CDI or the CDI is not consistent with actual conditions, then implicitly attested component state may be inaccurate. For fields included in Evidence, the issuer MUST ensure that it derives the value by measuring the Target Environment whenever a change is made.

## 7.2 Security Validation

**Start of Informative Comment**

It is often necessary or desirable to ensure an Attester, and therefore, an Attesting Environment, has been validated and complies with a standard set of security requirements. The Federal Information Processing Standard (FIPS) Publication 140-3 [24] is one important example. It is a U.S. government standard that defines minimum security requirements for cryptographic modules in information technology products. This section provides guidance to help facilitate compliance for DICE-based Attesters.

### 7.2.1 Cryptographic Keys

Cryptographic keys in FIPS must be generated from the output of an approved Deterministic Random Bit Generator (DRBG). First, a DRBG is instantiated to create its initial internal state, as illustrated in Figure 12. Once instantiated, a DRBG acts as a one-way function in combination with a monotonic counter and optional entropy for prediction resistance. The monotonic counter represents the DRBG's internal state.

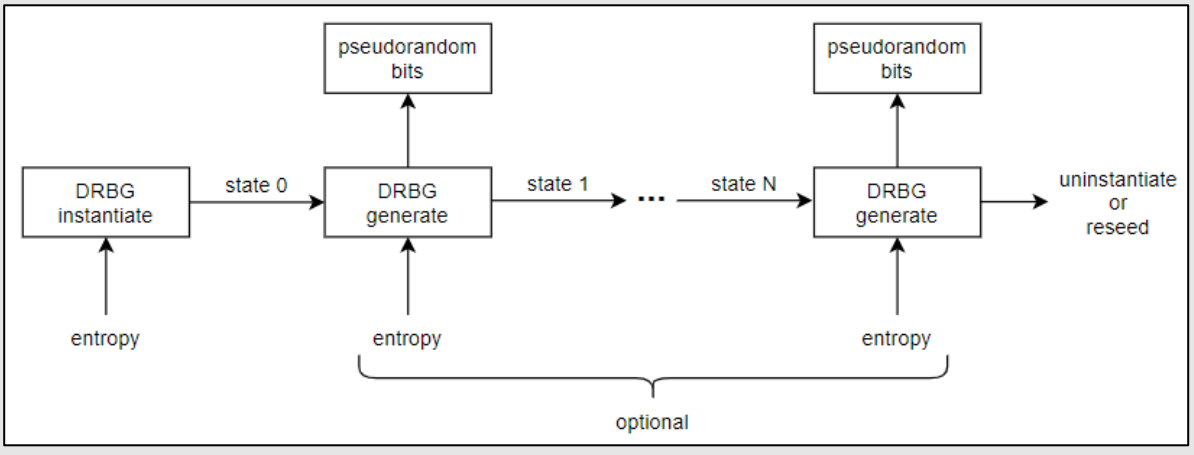


*Figure 12: Cryptographic Key Origination in FIPS, DRBG States*

For DICE implementations in which in which it is not possible or desirable to reinstantiate the DRBG with the same random seed on each power-on reset cycle, an asymmetric key of any DICE type (ECA, attestation, identity) can be generated exactly once and stored. The dependency of the generated keys on a CDI is achieved by inputting its creation components into the DRBG. After generated keys are created, they can be stored instead of being regenerated by reusing DRBG state. A retrieval mechanism can be used to protect the generated keys. A retrieval

 

mechanism that relies on a CDI value (and therefore, code measurements) is required to guarantee that any change to the original creation components of a key result in either a different key or an inability to retrieve the key.

Therefore, a key retrieval mechanism of adequate cryptographic strength is required to accommodate all key types. The strength of the retrieval mechanism must be equal to or higher than the cryptographic strength of the keys to which the retrieval mechanism controls access.

### 7.2.2 Retrieval Mechanisms

A key retrieval mechanism controls access to stored asymmetric keys. One way to control access is to encrypt an asymmetric key with an encryption key linked to the creation components of the asymmetric key. This way both asymmetric and encryption keys are derived from the same origin and are bound cryptographically. The asymmetric key is generated once, and the encryption key is recalculated periodically. Any change to the creation components produces an invalid decryption key and precludes retrieval of the valid asymmetric key. An attractive feature of this approach is the ability to store encrypted asymmetric keys in untrusted memory. However, doing so allows for unauthorized modification and substitution which FIPS requires protection against, so additional measures, such as integrity checks, are required. The strength of this retrieval mechanism is dependent on the combination of selected encryption, key derivation, and integrity protection algorithms. See Figure 13.

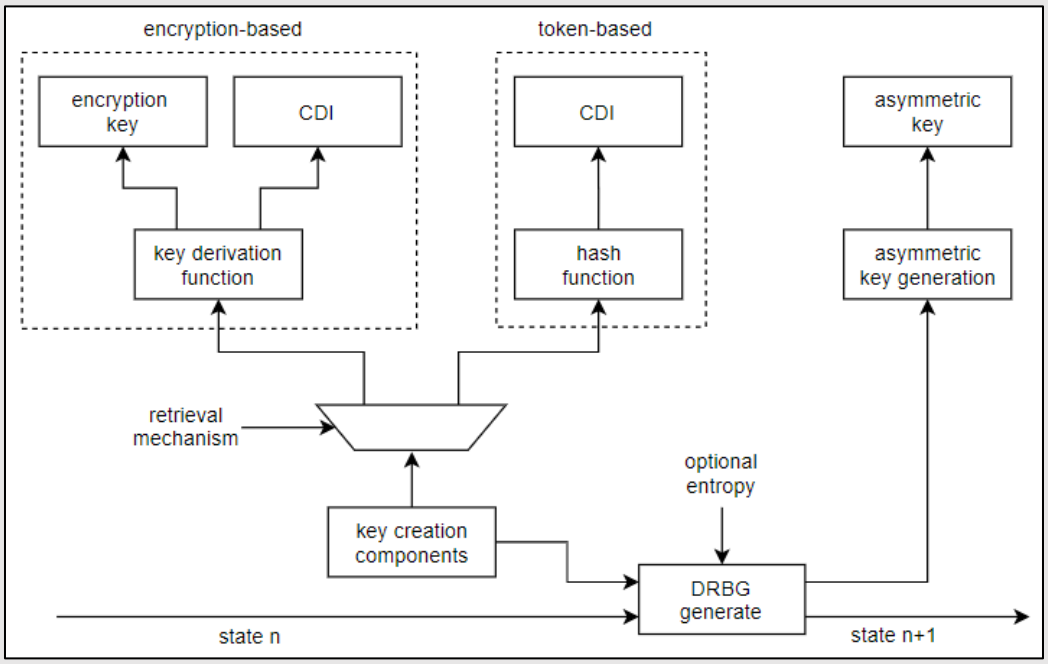


*Figure 13: Key Generation for Retrieval Mechanisms*

Another way to control access is to employ a logical safeguard mechanism. To retrieve a valid asymmetric key from storage, the requesting party must prove the integrity of key creation components by calculating a retrieval token. Because the requesting party itself is one of the creation components, access is by default self-authenticated. A major benefit of this approach is the possibility of replacing encryption and key derivation algorithms by a single hash function.

In FIPS, a cryptographic key is associated with a single purpose and cryptographic algorithm. As a result, the encryption-based approach requires the CDI to be separated from an encryption key as they are used by different algorithms. On the other hand, the token-based approach allows for creation of a single value per layer (the CDI) which is then used for both linkage to subsequent layers, and access to stored asymmetric keys.

 

### 7.2.3 Protected Storage

Protected storage is used by retrieval mechanisms to support integrity checks. Regardless of whether the keys are stored encrypted or not, unauthorized modification of stored values need to be detected. In addition to integrity checks, the token-based approach also stores and protects expected retrieval tokens for comparison with the periodically calculated tokens.

## 7.3 Evidence

### 7.3.1 Freshness

A freshness attribute should be included with collected claims that describes the period of time where changes to Evidence could have escaped detection by the Attesting Environment. Verifiers are expected to determine whether freshness values are sufficient.

### 7.3.2 Privacy

If the manifest, attribute certificate, or identity certificate containing attestation extensions does not contain the component's identity or firmware measurement, then an attestation Verifier might not be able to associate attestation Evidence with an appraisal policy for Evidence.

Evidence and Endorsement values conveyed over a public network might be subject to privacy sensitive observation. It is the responsibility of the conveyance protocol carrying Evidence or Endorsement values to confidentiality protect its payloads.

**End of Informative Comment**



 

# 8 Appendix A – Complete ASN.1

This appendix summarizes the OIDs and structures defined in this specification.

## 8.1 OIDs

```
tcg OBJECT IDENTIFIER ::= {2 23 133}
tcg-dice OBJECT IDENTIFIER ::= { tcg platformClass(5) 4 }
tcg-dice-TcbInfo OBJECT IDENTIFIER ::= {tcg-dice 1}
tcg-dice-TcbInfoAlias OBJECT IDENTIFIER ::= {tcg-dice-TcbInfo 1}
tcg-dice-endorsement-manifest OBJECT IDENTIFIER ::= {tcg-dice 2}
tcg-dice-endorsement-manifest-uri OBJECT IDENTIFIER ::= {tcg-dice 3}
tcg-dice-Ueid OBJECT IDENTIFIER ::= {tcg-dice 4}
tcg-dice-MultiTcbInfo OBJECT IDENTIFIER ::= {tcg-dice 5}
tcg-dice-UCCS-evidence OBJECT IDENTIFIER ::= {tcg-dice 6}
tcg-dice-manifest-evidence OBJECT IDENTIFIER ::= {tcg-dice 7}
tcg-dice-MultiTcbInfoComp OBJECT IDENTIFIER ::= {tcg-dice 8}
tcg-dice-conceptual-message-wrapper OBJECT IDENTIFIER ::= {tcg-dice 9}
tcg-dice-attestation-evidence OBJECT IDENTIFIER ::= {tcg-dice 10}
tcg-dice-TcbFreshness OBJECT IDENTIFIER ::= {tcg-dice 11}
```

## 8.2 Structures

```
DiceTcbInfo ::== SEQUENCE {
      vendor      [0] IMPLICIT UTF8String           OPTIONAL,
      model       [1] IMPLICIT UTF8String           OPTIONAL,
      version     [2] IMPLICIT UTF8String           OPTIONAL,
      svn         [3] IMPLICIT INTEGER              OPTIONAL,
      layer       [4] IMPLICIT INTEGER              OPTIONAL,
      index       [5] IMPLICIT INTEGER              OPTIONAL,
      fwids       [6] IMPLICIT FWIDLIST             OPTIONAL,
      flags       [7] IMPLICIT OperationalFlags     OPTIONAL,
      vendorInfo  [8] IMPLICIT OCTET STRING         OPTIONAL,
      type        [9] IMPLICIT OCTET STRING         OPTIONAL,
      flagsMask   [10]IMPLICIT OperationalFlagsMask OPTIONAL
}

FWIDLIST ::== SEQUENCE SIZE (1..MAX) OF FWID

FWID ::== SEQUENCE {
      hashAlg           OBJECT IDENTIFIER,
      digest            OCTET STRING
}

OperationalFlags ::= BIT STRING {
      notConfigured (0),
      notSecure (1),
      recovery (2),
      debug (3),
      notReplayProtected (4),
      notIntegrityProtected (5),
      notRuntimeMeasured (6),
      notImmutable (7),
      notTcb (8),
      fixedWidth (31)
}
```



 

```
OperationalFlagsMask ::= BIT STRING {
      notConfigured (0),
      notSecure (1),
      recovery (2),
      debug (3),
      notReplayProtected (4),
      notIntegrityProtected (5),
      notRuntimeMeasured (6),
      notImmutable (7),
      notTcb (8),
      fixedWidth (31)
}

DiceTcbInfoSeq ::= SEQUENCE SIZE (1..MAX) OF DiceTcbInfo

DiceTcbInfoComp ::= SEQUENCE SIZE (1..MAX) OF TcbInfoComp

TcbInfoComp ::= {
   commonFields [0] IMPLICIT DiceTcbInfo,
   evidenceValues [1] IMPLICIT DiceTcbInfoSeq
}

TcgUeid ::== SEQUENCE {
      ueid OCTET STRING
}

UccsEvidence ::== SEQUENCE {
      uccs OCTET STRING
}

Manifest ::== SEQUENCE {0
      format      [0] ManifestFormat,
      manifest    [1] OCTET STRING,
}

ManifestFormat ::= ENUMERATED {
      swid-xml          (0),
      coswid-cbor       (1),
      coswid-json       (2),
      tagged-cbor       (3)
}

EndorsementManifestURI ::== SEQUENCE {
      emUri UTF8String,
}

TaggedEvidence ::= SEQUENCE {
      taggedEvidence OCTET STRING
}

AttestationResults ::= SEQUENCE {
      taggedAttestationResults OCTET STR
}
```



 