TRUSTED® COMPUTING GROUP

SPECIFICATION

DICE Attestation Architecture

Version 1.3 RC1
July 23, 2026

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. This specification relies on the terminology, concepts, and requirements set forth in part 1 of the TCG Attestation Framework specification [1].

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 Attestation Framework Part 1: Terminology, Concepts, and Requirements," 1 November 2025. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TCG-Attestation-Framework-Part-1_pub.pdf.

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

[3] Trusted Computing Group, "DICE Endorsement Architecture for Devices," 2022. [Online]. Available: https://www.trustedcomputinggroup.org.

[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]. Available: https://www.trustedcomputinggroup.org.

[6] 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.

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

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

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

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

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

[12] 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/.

[13] 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.

[14] 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.

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

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

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

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

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

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

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

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

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

[24] 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.

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 [2].

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.
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.
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* [2] – *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 [2] – *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* [2]– *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* [2] – *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* [2] – *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 [2] – *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.
Verifier Owner	An attestation Role that conveys Appraisal Policy for Evidence to a Verifier.

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 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

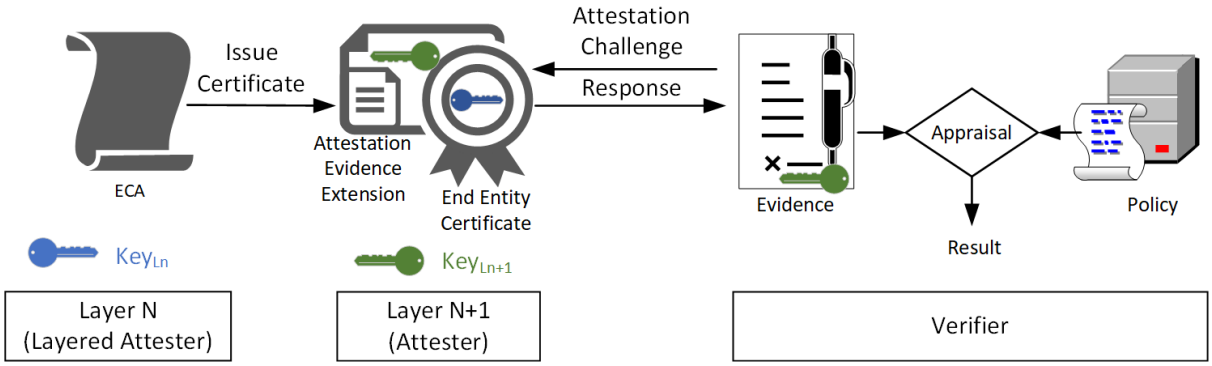

Figure 1: Layered Attestation

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.

5.1 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 [6] 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 can 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, can 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 Container: `tcg OBJECT IDENTIFIER ::= {2 23 tcg(133)}`
DICE Container: `tcg-dice OBJECT IDENTIFIER ::= { tcg platformClass(5) dice(4) }`
TcbInfo Container: `tcg-dice-TcbInfoContainer ::= { tcg-dice tcbinfo(1) tcb-info-container(0) }`

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

5.1.1 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 tcbinfo(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
   integrityRegisters  [11] IMPLICIT IrList              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 §5.1.1.1).
- *vendorInfo* – vendor supplied values that encode vendor, model, or device specific state.
- *flagsMask* – Used to communicate which bits within *flags* are meaningful. (see §5.1.1.2)
- *integrity*Registers – a list of named digests (see §5.1.1.4).

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) MUST be 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

5.1.1.1 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 might 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 four (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. The `fixedWidth` bit 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 5.1.1.2) 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 might 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` are 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

5.1.1.2 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 5.1.1.1), 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 5.1.1.1 for a description of the `fixedWidth` bit. It has the same purpose and meaning within the `OperationalFlagsMask` field as `fixedWidth` does for `OperationalFlags`.

5.1.1.3 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 alias(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.

5.1.1.4 Integrity Registers

Start of informative comment

The `integrityRegisters` field models a series of algorithm agile measurements such as an accumulation of runtime updates, a sequence of boot time events, or a set of closely related objects that are typically bundled. Each integrity register is identified by a text or numeric label to aid in comparison operations. Integrity registers support algorithm agile architectures where multiple hash algorithms may be used to generate multiple digests for a given input.

End of informative comment

The `DiceTcbInfo` attributes both identify the target environment and contain target environment measurements. Target environments can contain multiple objects that can be hashed, or an object can be hashed multiple times over some period.

The `integrityRegisters` field consists of a list (`IrList`) of integrity registers (`IntegrityRegister`). Each register can be named using a textual or numeric name. The register name facilitates comparison with Reference Values given differently ordered lists. Either `registerName` or `registerNum` MUST be supplied.

```
IrList ::= SEQUENCE SIZE (1..MAX) OF IntegrityRegister

IntegrityRegister ::= SEQUENCE {
   registerName     [0] IMPLICIT IA5String OPTIONAL,
   registerNum      [1] IMPLICIT INTEGER OPTIONAL,
   registerDigests  [2] IMPLICIT FWIDLIST
}
```

5.1.2 Multiple DiceTcbInfo Structures Extension

The initial state of a Target Environment could 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 multi-tcbinfo(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.

5.1.3 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 multi-tcbinfo-comp(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`.

5.1.4 UEID Extension

This extension contains a UEID [7] 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 ueid(4) }
```

The ASN.1 definition is as follows:

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

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

5.1.5 CWT Claims Set Evidence Extension

The CBOR Web Token (CWT) specification [8] defines a CBOR encoding of a claim set that can 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 uccs-evidence(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 include content this extension that 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.

5.1.6 Manifest Evidence Extension

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

The OID declaration of `DiceManifestEvidence` is as follows:

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

The `tcg-dice-manifest-evidence` object identifier is used with the `Manifest` sequence (See §5.5.1.1). 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.

5.1.7 AuthorityKeyIdentifier Certificate Extension

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

5.1.8 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 [9], 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, [10] 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 [11] 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 [12].

Conceptual message may be used by Web authentication protocols that rely on public key credentials such as X.509 certificates. [13] 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 cmw(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 [10], or the tagged CBOR form, see [11], [14], and [10].

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 [15] and [9] for guidance on text encoding constraints that are applied to the `media-type` statement.

The conceptual message wrapper in the array form, see [10], is used when the conceptual message type has been registered as a `media-type` [15] or as a `coap-content-format` [16]. The CBOR tag form, see [9], is used when the conceptual message type has been registered as a CBOR tag, see [11], or when a CBOR tag is derived from a `coap-content-format` using the TN() transform as defined in [14].

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

5.2 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

5.3 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 tcb-freshness(11) }
```

The ASN.1 definition is as follows:

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


The `DiceTcbFreshness` extension SHOULD be marked critical.

5.4 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.

5.5 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 can also contain Endorsed Values that are assertions that are not matched with Evidence, but are associated with the Attester, and might 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 need to all be known at the time the certificate is issued.

5.5.1 Endorsements as X.509 Certificate Extensions

5.5.1.1 Manifest as an X.509 Certificate Extension

Start of informative comment

X.509 certificates [6] 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 [17] and contains a SWID Tag manifest as defined by [18].
- *coswid-cbor* – The manifest format is CBOR [11] and contains a CoSWID manifest as defined by [19].
- *coswid-json* – The manifest format is JSON [20] 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.

5.5.1.2 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 [21] that contains an object reference to a manifest. For example, a SWID Tag schema contains a 'tagId' attribute that could be encoded in an emURI.

5.5.2 Endorsements Using X.509 Attribute Certificates

Start of informative comment

X.509 attribute certificates [22] 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.

5.5.3 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.

5.6 Attestation Results

Start of informative comment

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

End of informative comment

5.6.1 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` can be used to convey Attestation Results within an X.509 certificate extension.

6 ATTESTING ENVIRONMENT

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

6.1 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 is likely 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.

6.2 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 [23] 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.

6.2.1 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 2. 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.

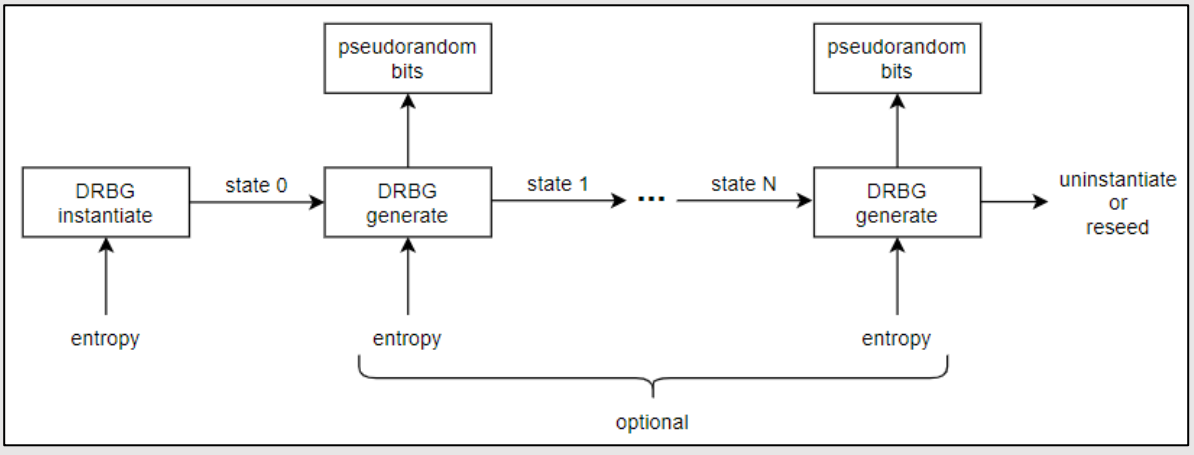

Figure 2: Cryptographic Key Origination in FIPS, DRBG States

For DICE implementations 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.

6.2.2 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 3.

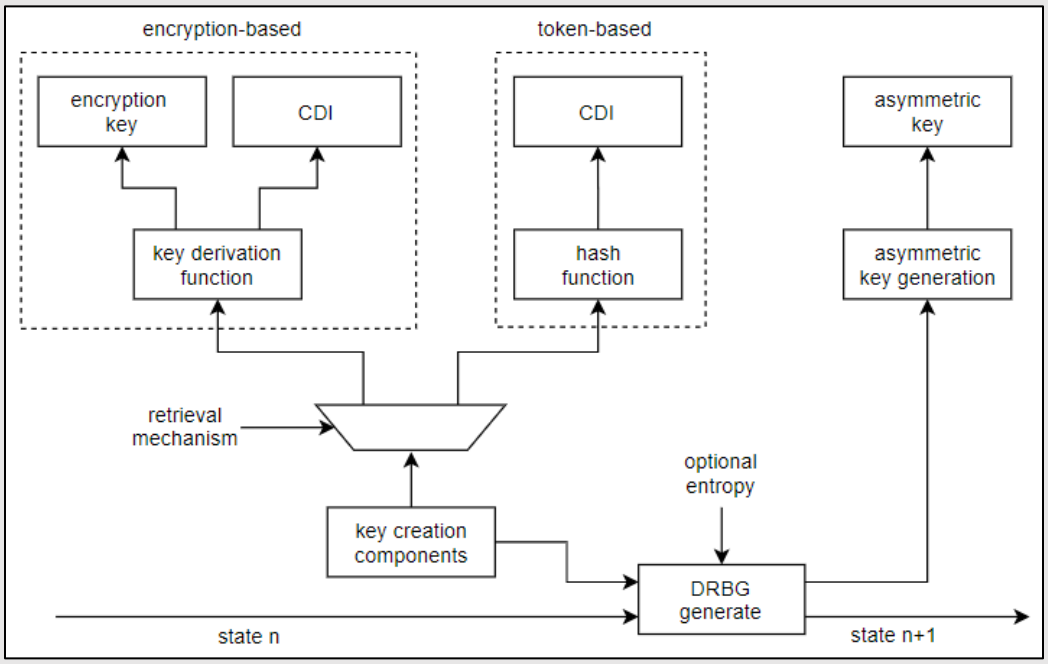

Figure 3: Key Generation for Retrieval Mechanisms

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.

6.2.3 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 needs 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.

6.3 Evidence

6.3.1 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.

6.3.2 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

Appendix A – ASN.1 Exports

This appendix contains ASN.1 exports as defined in this specification.

```
TcgDiceAttestation DEFINITIONS AUTOMATIC TAGS ::= BEGIN

EXPORTS ALL;

tcg OBJECT IDENTIFIER ::= {2 23 tcg(133)}
tcg-dice OBJECT IDENTIFIER ::= {tcg platformClass(5) dice(4)}
tcg-dice-TcbInfo OBJECT IDENTIFIER ::= {tcg-dice tcbinfo(1)}
tcg-dice-TcbInfoAlias OBJECT IDENTIFIER ::= {tcg-dice-TcbInfo alias(1)}
tcg-dice-endorsement-manifest OBJECT IDENTIFIER ::= {tcg-dice endorsement-manifest(2)}
tcg-dice-endorsement-manifest-uri OBJECT IDENTIFIER ::= {tcg-dice manifest-uri(3)}
tcg-dice-Ueid OBJECT IDENTIFIER ::= {tcg-dice ueid(4)}
tcg-dice-MultiTcbInfo OBJECT IDENTIFIER ::= {tcg-dice multi-tcbinfo(5)}
tcg-dice-UCCS-evidence OBJECT IDENTIFIER ::= {tcg-dice uccs-evidence(6)}
tcg-dice-manifest-evidence OBJECT IDENTIFIER ::= {tcg-dice manifest-evidence(7)}
tcg-dice-MultiTcbInfoComp OBJECT IDENTIFIER ::= {tcg-dice multi-tcbinfo-comp(8)}
tcg-dice-conceptual-message-wrapper OBJECT IDENTIFIER ::= {tcg-dice cmw(9)}
tcg-dice-TcbFreshness OBJECT IDENTIFIER ::= {tcg-dice tcb-freshness(11)}


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
   integrityRegisters  [11] IMPLICIT IrList             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)
}

IrList ::= SEQUENCE SIZE (1..MAX) OF IntegrityRegister

IntegrityRegister ::= SEQUENCE {
      registerName     [0] IMPLICIT IA5String OPTIONAL,
      registerNum      [1] IMPLICIT INTEGER OPTIONAL,
      registerDigests  [2] IMPLICIT FWIDLIST
}

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

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

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

TcgUeid ::= SEQUENCE {
  ueid OCTET STRING
}

UccsEvidence ::= SEQUENCE {
  uccs OCTET STRING
}

Manifest ::= SEQUENCE {
  format ManifestFormat,
  manifest OCTET STRING
}

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



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

AttestationResults ::= SEQUENCE {
  taggedAttestationResults OCTET STRING
}

DiceTcbFreshness ::= SEQUENCE {
  nonce OCTET STRING
}

DiceConceptualMessageWrapper ::= SEQUENCE {
  cmw OCTET STRING
}

END
```


