Workgroup: Remote ATtestation ProcedureS
Internet-Draft:
draft-cds-rats-intel-corim-profile-00
Published: 23 June 2023
Intended Status: Informational
Expires: 25 December 2023
Authors: S. Cen
Intel Corporation
A. Draper
Intel Corporation
N. Smith
Intel Corporation

Intel Profile for CoRIM

Abstract

This document describes extensions to the CoRIM schema that support Intel specific Attester implementations. Multiple Evidence formats are compatible with base CoRIM, but extensions to evidence formats may be required to fully support the CoMID schema extensions defined in this profile. The concise evidence definition uses the CoMID schema such that extensions to CoMID are inherited by concise evidence. Reference Value Providers may use this profile to author mainifests containing Reference Values and Endorsements. RATS Verifiers recognize this profile by it's profile identifier and implement support for the extentions defined.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://nedmsmith.github.io/draft-cds-rats-intel-corim-profile/draft-cds-rats-intel-corim-profile.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-cds-rats-intel-corim-profile/.

Discussion of this document takes place on the Remote ATtestation ProcedureS Working Group mailing list (mailto:rats@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/rats/. Subscribe at https://www.ietf.org/mailman/listinfo/rats/.

Source for this draft and an issue tracker can be found at https://github.com/nedmsmith/draft-cds-rats-intel-corim-profile.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Copyright Notice

Table of Contents

1. Introduction

This profile describes extensions and restrictions placed on Reference Values, Endorsements, and Evidence that support attestation capabilities of Intel products including Intel(R) SGX(TM), and products that contain a DICE [DICE.engine] root of trust, DICE layers [DICE.layer], or modules that implement SPDM [DMTF.SPDM] endpoints.

The CoMID schema [DICE.CoRIM] and data model [I-D.ietf-rats-corim] is a baseline for Reference Values that expects Evidence is matched using values that are identical. This profile anticipates Reference Values that are a set or range of values where an Evidence value is within the reference set or range. This document describes schema and data model extensions for matching based on membership in a set, masked values, and numeric ranges.

The baseline CoRIM schema defines a spartan set of measurement values that are etended by this profile to better support Intel(r) products. However, the defined extensions may be generally useful such that implementation of the Intel profile need not imply the Attester, Verifier, Relying Party, Reference Value Provider, or Endorser must be Intel products.

This profile extends CoMID schema measurement-values-map, as defined by [I-D.ietf-rats-corim], with measurements that may be unique to Intel products or are not defined anywhere else. Some measurement definitions are specific to Reference Values such that multiple Reference Values may be specified and an operator instructs Verifiers regarding the matching algorithm to apply. For example, a numeric operator 'greater-than' instructs the Verifier to match a numeric Evidence value if it is greater than one or more numeric Reference Values.

This profile follows the Verifier behavior defined by [I-D.ietf-rats-corim] and extends Verifier behavior to include non-exact matching as indicated by a supplied operator. If no operator is specified by Reference Value statements, the Verifier defaults to exact matching. If Evidence matches Reference Values and Endorsements apply, endorsed values are added to the the accetped measurements. When all Evidence and Endorsements are processed, the Verifier's set of accepted measurements is used to produce Attestation Results.

This profile is compatible with multiple Evidence formats, as defined by [DICE.Attest] and SPDM [DMTF.SPDM]. It describes considerations when mapping Evidence formats to CoRIM that a Verifier may use when doing matching.

2. Conventions and Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The reader is assumed to be familiar with the terms defined in Section 4 of [RFC9334].

3. Background

Complex platforms may contain a variety of hardware components, several of which may contain a hardware root of trust. The root of trust may anchor one or more layers [DICE.layer] resulting in multiple instances of attestation Evidence. Evidence may be protected by an wrapping structure, such as a certificate [DICE.Attest] or a secure transport over a bus interface [DMTF.SPDM] may provide integrity protection. For example, a system bus may allow dynamically configured peripheral devices that have attestation capabilities. Confidential computing environments, such as SGX, may extend an initial boundary to include a peripheral, or a

peer enclave, that together forms a network of trustworthy nodes that a remote attestation Verifier may need to appraise.

Such a complex platform may rely one or more endpoints that communicate with a remote Verifier to convey a structure that is a conglomeration of Evidence instances. A complex platform may consist of multiple instances of a subsystem, such as multiple network adapters, storage controllers, or processors. Even though they may be identical copies, each instance should have its own Evidence instance. Insertion and removal of a configurable component may affect the composition of Evidence.

4. Profile Identifier

This profile applies to Reference Values from a CoRIM manifest that a Verifier uses to process Evidence.

The profile identifier structure is defined by CoRIM as:

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

The profile identifier for this profile is the OID:

{joint-iso-itu-t(2) country(16) us(840) organization(1) intel(113741) (1) intel-comid(16) profile(1)}

2.16.840.1.113741.1.16.1

5. CoMID Schema Extensions

The baseline CoMID schema for Reference Values is extended with an attribute that informs a Verifier as to which matching semantics to apply, whether they are equivalance, range, or set membership semantics.

This profile extends measurement-values-map with additional measurements that are used by Evidence, Reference Values, and Endorsed Values.

5.1. Expressions

Expressions are used to specify richer Reference Values measurements that expect non-exact matching semantics. The Reference Value expression instructs the Verifier regarding matching parameters, such as greater-than or less-than, ranges, sets, etc. Typically, the Evidence value is one operand of an expression and the Reference Value contains the operator and additional operands.

The operator and remaining operands are contained in an array. Expression arrays have an operator followed by zero or more operands. The operator definition identifies the additional operands and their data types. A Verifier forms an expression using Evidence as the first operand, obtains the operator from the first entry in the expression array, and any remaining array entries are operands.

This document describes operations using *infix* notation where the first operand, *operand_1*, is obtained from Evidence, followed by the operator, followed by any remaining operands: *operand_2*, *operand_3*, etc...

For example:

* From Evidence: operand_1, from Reference Values: [operator, operand_2, operand_3, ...].

Expression records are CBOR tagged to indicate the following CBOR is to be evaluated as an expression. Expression statements found in Reference Values informs the Verifier that Evidence is needed to complete the expression equation. Appraisal processing **MUST** evaluate the expression.

This profile anticipates use of the CBOR tag #6.60010 to identify expression arrays. See Section 10

For example:

* #6.60010([operator, operand_2, operand_3, ...]).

5.1.1. Expression Operators

There are several classes of operators as follows:

1. **Numeric**: The numeric operand can be an integer, unsigned integer, or floading point value.

2. **Set**: The set operand can be an set (array) of any primitive value or a set of arrays containing any primitive value. The position of the items in a set is unordered, while the position of items in an array within a set is ordered.

3. **Date-Time**: The date-time operators compare two date-time values, while the epoch operators determine if a date-time value is within a range defined by an epoch and a grece period relative to the epoch.

4. **Mask**: The mask operator compares two bit fields where a bit-field is compared to a second bit-field using a bit-field-mask that selects the bits to be compared. The bit-field may contain

a sequence of bytes of any length. The bit-field-mask should be the same length as the bit-field.

5.1.1.1. Equivalence Operator

By default, *exact* match rules are assumed. Consequently, no operator artifact is needed when Evidence values are identical to Reference Values.

5.1.2. Numeric Expressions

Numeric operators apply to values that are integers, unsigned integers or floating point numbers. There are four numeric operators:

1. **greater-than** (gt),
2. **greater-than-or-equal** (ge),
3. **less-than** (lt), and
4. **less-than-or-equal** (le).

The equals operator is not defined because an exact match rule is the default rule when an Evidence value is identical to a Reference Value.

The numeric operator data type definitions are as follows:

```
gt = 1
ge = 2
lt = 3
le = 4
numeric-type = integer / unsigned / float
numeric-operator = gt / ge / lt / le
numeric-expression = [ numeric-operator, numeric-type ]
tagged-numeric-expression = #6.60010(numeric-expression)
```

A numeric expression is an array containing a numeric operator and a numeric operand. The operand contains a numeric Reference Value that is matched with a numeric Evidence value.

Evidence and Reference Values **MUST** be the same numeric type. For example, if a Reference Value numeric type is integer, then the Evidence numeric value must also be integer.

This profile defines four macro numeric expressions, one for each numeric operator:

*tagged-numeric-gt,

*tagged-numeric-ge,

*tagged-numeric-lt, and

*tagged-numeric-le.

In each case, the numeric operator is used to evaluate a Reference Value operand against an Evidence value operand that is obtained from Evidence.

If the expression is read using *infix* notation, the first operand is Evidence, followed by the operator, followed by the Reference Value operand.

Example:

*<evidence_numeric> <le> <reference_numeric>

The numeric expression definitions are as follows:

```
tagged-numeric-gt = #6.60010( [
    greater-than: gt .within numeric-operator,
    reference-value: numeric-type ] )
tagged-numeric-ge = #6.60010( [
    greater-than: ge .within numeric-operator,
    reference-value: numeric-type ] )
tagged-numeric-lt = #6.60010( [
    greater-than: lt .within numeric-operator,
    reference-value: numeric-type ] )
tagged-numeric-le = #6.60010( [
    greater-than: le .within numeric-operator,
    reference-value: numeric-type ] )
```

5.1.3. Set Expressions

Set operators allow Reference Values, that are expressed as a set, to be compared with Evidence that is expressed as either a primitive value or a set.

Set expression statements have two forms:

1. A binary relation between a primitive object 'O' and a set 'S', and

2. A binary relation between two sets, an Evidence set 'S1' and a Reference Values set 'S2'.

The first form, relation between an object and a set, has two operators:

* **member** and

* **not-member**.

The Evidence object 'O' is evaluated with the Reference Values set 'S'.

The op.member and op.not-member operators expect a Reverence Value set as *operand_2* and a primitive Evidence value as *operand_1*. Evaluation tests whether *operand_1* is a member of the set *operand_2*.

Example:

* <evidence-value> <set-operator> <reference-set>

The set data type definitions are defined as follows:

```
member = 6
not-member = 7

set-type = [ * any ]
set-operator = member / not-member
set-expression = [ set-operator, set-type ]
tagged-set-expression = #6.60010( set-expression )
```

A set expression array contins a set operator followed by a set of Reference Values. The set is defined by set-type.

The set expression type definitions are as follows:

```
tagged-exp-member = #6.60010([
    member .within set-operator, set-type ])

tagged-exp-not-member = #6.60010([
    not-member .within set-operator, set-type ])
```

Examples:

* <evidence_object> <member> <reference_set>

* <evidence_object> <not-member> <reference_set>

The Evidence object **MUST NOT** be nil.

The Reference Values set may be the empty set.

The second form, a relation between two sets, has three operators:

- **subset**,
- **superset**, and
- **disjoint**.

The fist set, S1 is Evidence and set, S2 is the Reference Values set.

The subset, superset, and disjoint operators test whether an Evidence set value satisfies a set operation, given a Reverence Value set.

Examples:

- <evidence_set> <subset> <reference_set>
- <evidence_set> <superset> <reference_set>
- <evidence_set> <disjoint> <reference_set>

The Reference Values and Evidence sets may be the empty set.

The set of sets data type definitions are as follows:

```
subset = 8
superset = 9
disjoint = 10
set-of-set-type = [ * [ + any ]]
set-of-set-operator = subset / superset / disjoint
set-of-set-expression = [ set-of-set-operator, set-of-set-type ]
tagged-set-of-set-expression = #6.60010(set-of-set-expression)
```

For subset, every member in the Evidence set 'S1', **MUST** map to a member in the Reference Values set 'S2'. The Evidence set, 'S1', **MAY** be a proper subset of 'S2'. For example, if every member in set 'S1' maps to every member in set 'S2' then matching is successful. A successful result produces the set 'S1'.

For superset, every member in the Reference Values set 'S2', **MUST** map to a member in the Evidence set 'S1'. A successful result produces the set 'S1'.

For disjoint, every member in the Evidence set 'S1' **MUST NOT** map to any member in the Reference Values set 'S2'. A successful result produces the empty set.

The set of sets expression definitions are as follows:

```
tagged-exp-subset = #6.60010([
    subset .within set-of-set-operator, set-of-set-type ])

tagged-exp-superset = #6.60010([
    superset .within set-of-set-operator, set-of-set-type ])

tagged-exp-disjoint = #6.60010([
    disjoint .within set-of-set-operator, set-of-set-type ])
```

5.1.4. Time Expressions

5.1.4.1. Date-Time Expressions

Date-time can be expressed in both string tdate or numeric time formats. There are four operators that are defined for date-time expressions:

1. **lt** determines if a date-time Evidence value is less than a Reference Values date-time.

2. **le** determines if a date-time Evidence value is less than or equal to a Reference Values date-time.

3. **gt** determines if a data-time Evidence value is greater than a Reference Values date-time.

4. **ge** determines if a data-time Evidence value is greater than or equal to a Reference Values date-time.

The date-time operators are in fact numeric. Expressions involving string date-time values must be converted to numeric format before the numeric comparison operator can be applied.

The numeric date-time data type definitions are as follows:

```
time-operator = numeric-operator
time-expression = [ time-operator, time ] ;#6.1(number)
tagged-time-expression = #6.60010( time-expression )
```

The string date-time data type definitions are as follows:

```
tdate-operator = numeric-operator ; converts tdate to numeric
tdate-expression = [ tdate-operator, tdate ] ;#6.0(string)
tagged-tdate-expression = #6.60010( tdate-expression )
```

The date-time expressions for evaluating time consist of a CBOR tagged record containing a time operator followed by a date-time value.

The gt expression compares a date-time value contained in Evidence to a Reference Values date-time. If the Evidence date-time value is greater-than to the Reference Value, then the Evidence value is accepted.

The ge expression compares a date-time value contained in Evidence to a Reference Values date-time. If the Evidence date-time value is greater-than-or-equal to the Reference Value, then the Evidence value is accepted.

The lt expression compares a date-time value contained in Evidence to a Reference Values date-time. If the Evidence date-time value is less-than to the Reference Value, then the Evidence value is accepted.

The le expression compares a date-time value contained in Evidence to a Reference Values date-time. If the Evidence date-time value is less-than-or-equal to the Reference Value, then the Evidence value is accepted.

In *infix* notation, a date-time value reported as Evidence in *operand_1*. The Reference Value expression contains a time comparison operator and *operand_2* contains a reference date-time.

Example:

*<evidence_date_time> <gt> <reference_date_time>

The numeric date-time expression definitions are as follows:

```
tagged-exp-time-gt = #6.60010([
    gt .within time-operator,
    time ])

tagged-exp-time-ge = #6.60010([
    ge .within time-operator,
    time ])

tagged-exp-time-lt = #6.60010([
    lt .within time-operator,
    time ])

tagged-exp-time-le = #6.60010([
    le .within time-operator,
    time ])
```

The string date-time expression definitions are as follows:

```
tagged-exp-tdate-gt = #6.60010([
    gt .within tdate-operator,
    tdate ])

tagged-exp-tdate-ge = #6.60010([
    ge .within tdate-operator,
    tdate ])

tagged-exp-tdate-lt = #6.60010([
    lt .within tdate-operator,
    tdate ])

tagged-exp-tdate-le = #6.60010([
    le .within tdate-operator,
    tdate ])
```

5.1.4.2. Epoch Expressions

An epoch expression defines a timing window that can be used to determine recentness of a time stampped message. By default, the Verifier's current time implicitly defines an epoch. A grace period defines the epoch window differential, in seconds, that a timestamp must fall within to be valid.

Epochs don't have to rely on current time. [I-D.birkholz-rats-epoch-markers] defines several.

The epoch data type definitions are as follows:

```
epoch-operator = numeric-operator
epoch-seconds-type = int
epoch-expression = [
    epoch-operator,
    grace-period: epoch-seconds-type,
    ? $tagged-epoch-id
]
tagged-epoch-expression = #6.60010( epoch-expression )
$tagged-epoch-id /= tdate
$epoch-timestamp-type /= $tagged-epoch-id
```

An epoch expression array contains an epoch-operator followed by a grace period in seconds that is optionally followed by a $tagged-epoch-id. Epoch operators can be: gt, ge, lt, or le. The operator defines the position of the grace window relative to the epoch and epoch-grace-seconds defines the size of the window in seconds. If the default epoch type is not used, $tagged-epoch-id defines the alternate epoch scheme.

The $epoch-timestamp-type defines the timestamp type for the epoch scheme. By default, the timestamp is tdate.

The tagged-epoch-expression is an epoch-expression preceded by a CBOR tag for an expression array that has the value #6.60010.

The epoch expression definitions are as follows:

```
tagged-exp-epoch-gt = #6.60010([
    gt .within epoch-operator
    grace-period: epoch-seconds-type ])

tagged-exp-epoch-ge = #6.60010([
    ge .within epoch-operator
    grace-period: epoch-seconds-type ])

tagged-exp-epoch-lt = #6.60010([
    lt .within epoch-operator
    grace-period: epoch-seconds-type ])

tagged-exp-epoch-le = #6.60010([
    le .within epoch-operator
    grace-period: epoch-seconds-type ])
```

A variety of epoch expressions can be defined that convenently constrain epoch definition. The tagged-exp-epoch-gt expression defines an epoch window that is greater than the current date and time by the supplied grace period.

In *infix* notation, the timestamp value is within an epoch window that is defined by the current time, plus the grace period, and where the epoch-operator defines the shape of the epoch window.

Example epoch expression:

```
  *<evidence_timestamp> <epoch-operator> <grace_period>
   <current_time>
```

The Verifier adds grace_period to current_time to obtain the epoch window then applies the operator to determine if the evidence_timestamp is within the window.

5.1.5. Mask Expression

Reference Values expressed as an array of bits or bytes that uses a mask can indicate to a Verifier which bits or bytes of Evidence to ignore.

Reference Value and mask arrays **MUST** be the same length for the mask to be applied correctly. Normally, Evidence would not supply a mask, while Endorsed Values would. The mask-eq operator indicates that an Evidence value of type mask-type is compared with a Reference Value of type mask-type, and that a mask of type mask-type is applied to

both values before comparing values. If the operator is mask-eq, then a binary equivalence comparison is applied.

The Verifier **MUST** ensure the lengths of values and mask are equivalent. If the mask is shorter than the values, the mask is padded with zeros (0) until it is the same length as the largest value. If the Evidence or Reference Value length is shorter than the mask, the value is padded with zeros (0) to the length of the mask.

The masked data type definitions are as follows:

```
mask-eq = 1
mask-operator = mask-eq
mask-type = bstr
mask-expression = [ mask-operator, value: mask-type, mask: mask-type ]
tagged-mask-expression = #6.60010( mask-expression )
```

If the Evidence bit field is a different length from the Reference Value and mask, the shorter length bit field is padded with zeros to accommodate the larger bit field.

The tagged-exp-mask-eq expression defines a tagged expression that applies the mask equivalence operator to an Evidence value and a Reference Value using the supplied mask.

In *infix* notation, the Evidence value is *operand_1*, followed by the mask operator, followed by a Reference Value, *operand_2*, followed by the mask, *operand_3*.

Example:

*<evidence_value> <mask-eq> <reference_value> <mask>

If each bit in Evidence has a corresponding matching bit in the Reference Value, then the Evidence value is accepted.

The masked data expression definitions are as follows:

```
tagged-exp-mask-eq = #6.60010([
    mask-eq .within mask-operator,
    value: mask-type, mask: mask-type ] )
```

5.2. Measurement Extensions

This profile extends the CoMID measurement-values-map with additional code point definitions, that accommodate Intel SGX and similar architectures. Measurement extensions don't change Verifier behavior. An extension enables the Verifier to validate the profile compliance of the input evidence and reference values, as it defines the acceptable data types in evidence and the expression operator

that is explicitly supplied with the Reference Values, see Section 5.1.1.

In cases where Evidence does not exactly match Reference Values, the operator definition determines the expected data types of the operands. Expected Verifier behavior is defined in Section 7

5.2.1. The tee-advisory-ids-type Measurement Extension

The tee.advisory-ids extension allows the Attester to report known security advisories and a Reference Values Provider (RVP) to assert updated security advisories.

The $tee-advisory-ids-type is used to specify a set of security advisories, where each is identified by a string identifier. Evidence may report a set of advisories the Attester believes are relevant. The set of advisories are constrained by the set-of-set-type structure.

A Reference Values expression record is defined for this extension that applies the disjoint set operation to determine if there are advisories outstanding. If no advisories are outstanding, then the empty set signifies successful matching.

The $tee-advisory-ids-type is a list of advisories when used as Endorsements or Evidence and a disjoint set of advisories when used as a Reference Value.

```
$$measurement-values-map-extension //= (
  &(tee.advisory-ids: -89) => $tee-advisory-ids-type
)
$tee-advisory-ids-type /= ([ + tstr  ])
$tee-advisory-ids-type /= tagged-exp-disjoint
```

5.2.2. The tee-attributes-type Measurement Extension

The tee.attributes extension enables the Attester to report TEE attributes and an RVP to assert a reference TEE attributes and mask.

The $tee-attributes-type is used to specify TEE attributes in 8 or 16 byte words. If Evidence uses an 8 byte mask, then the Reference Values expression also uses an 8 byte value and mask.

The $tee-attributes-type is a singleton value omitting the mask value when used as Endorsement or Evidence and a tuple containing the reference and mask when used as a Reference Value.

```
$$measurement-values-map-extension //= (
  &(tee.attributes: -82) => $tee-attributes-type
)
$tee-attributes-type /= mask-type
$tee-attributes-type /= tagged-exp-mask-eq
```

5.2.3. The tee-date-type Measurement Extension

The tee.tcbdate extension enables the Attester or Endorser to report the TEE date attribute and a RVP to assert a valid TEE matching operation.

The $tee-date-type is a date-time string when used for Evidence or Endorsement. When used for a Reference Value, either a tagged-tdate-expression or a tagged-epoch-expression describes the TCB validity.

```
$$measurement-values-map-extension //= (
  &(tee.tcbdate: -72) => $tee-date-type
)
$tee-date-type /= tdate
$tee-date-type /= tagged-exp-tdate-ge
$tee-date-type /= tagged-exp-epoch-ge
```

tdate strings must be converted to numeric time before the tdate-operator, which is a numeric operator, can be applied.

5.2.4. The tee-digest-type Measurement Extension

The tee.mrenclave and tee.mrsigner extensions enable the Attester to report digests for the SGX enclave, a.k.a., Target Environment, and the signer, a.k.a., Attesting Environment, of the enclave digest.

The $tee-digest-type is a record that contains the hash algorithm identifier and the digest value when used as Evidence. When used as Reference Values, a set of digests can be asserted. The Evidence digest record must be a member of the Reference Values set.

```
$$measurement-values-map-extension //= (
  &(tee.mrenclave: -83) => $tee-digest-type
)
$$measurement-values-map-extension //= (
  &(tee.mrsigner: -84) => $tee-digest-type
)
$tee-digest-type /= hash-entry .within set-type
$tee-digest-type /= tagged-exp-member
```

5.2.5. The tee-epoch-type Measurement Extension

The tee.epoch extension enables the Attester to report an epoch Evidence measurement, and a RVP to assert an epoch Reference Value.

As Evidence, the $tee-epoch-type is a tdate timestamp.

As a Reference Value, the $tee-epoch-type is a grace period, in seconds, that is relative to the Verifier's current date and time, and an epoch operator: gt, ge, lt, or le. The Verifier evaluates whether the timestamp is within the grace period relative to the current date and time. The current date and time is implicit, and is assumed to be in tdate format.

The $tee-epoch-type is an $epoch-timestamp-type when used as Evidence or Endorsement and a $tagged-epoch-expression when used as a Reference Value.

```
$$measurement-values-map-extension //= (
  &(tee.epoch: -90) => $tee-epoch-type
)
$tee-epoch-type /= $epoch-timestamp-type
$tee-epoch-type /= tagged-exp-epoch-gt
```

5.2.6. The tee-fmspc-type Measurement Extension

The tee.fmspc extension enables the Attester to report the (TBD:fmspc-description) Evidence value and the RVP to assert an exact-match Reference Value.

The $tee-fmspc-type is a 6 byte word.

The $tee-fmspc-type is an exact match measurement.

```
$$measurement-values-map-extension //= (
  &(tee.fmspc: -75) => $tee-fmspc-type
)
$tee-fmspc-type /= bstr .size 6
```

5.2.7. The tee-instance-id-type Measurement Extension

The tee.instance-id extension enables the Attester to report the (TBD:instance-id-description) Evidence value and the RVP to assert an exact-match Reference Value.

The $tee-instance-id-type is an unsigned integer.

The $tee-instance-type is an exact match measurement.

```
$$measurement-values-map-extension //= (
  &(tee.instance-id: -77) => $tee-instance-id-type
)
$tee-instance-id-type /= uint
```

5.2.8. The tee-isvprodid-type Measurement Extension

The tee.isvprodid extension enables the Attester to report the ISV product identifier Evidence value and the RVP to assert an exact-match Reference Value.

The $tee-isvprodid-type is an unsigned integer.

The $tee-isvprodid-type is an exact match measurement.

```
$$measurement-values-map-extension //= (
  &(tee.isvprodid: -85) => $tee-isvprodid-type
)
$tee-isvprodid-type /= uint
```

5.2.9. The tee-miscselect-type Measurement Extension

The tee.miscselect extension enables the Attester to report the (TBD:miscselect-description) Evidence value and the RVP to assert a Reference Value and mask.

The $tee-miscselect-type is a 4 byte value and mask.

The $tee-miscselect-type is a singleton mask-type value when used as Endorsement or Evidence and a tagged-mask-expression when used a Reference Value.

```
$$measurement-values-map-extension //= (
  &(tee.miscselect: -81) => $tee-miscselect-type
)
$tee-miscselect-type /= mask-type
$tee-miscselect-type /= tagged-exp-mask-eq
```

5.2.10. The tee-model-type Measurement Extension

The tee.model extension enables the Attester to report the TEE model string as Evidence and the RVP to assert an exact-match Reference Value.

The $tee-model-type is a string.

The $tee-model-type is an exact match measurement.

```
$$measurement-values-map-extension //= (
  &(tee.model: -71) => $tee-model-type
)
$tee-model-type /= tstr
```

5.2.11. The tee-pceid-type Measurement Extension

The tee.pceid extension enables the Attester to report the (TBD:pceid-description) as Evidence and the RVP to assert an exact-match Reference Value.

The $tee-pceid-type is a string.

The $tee-pceid-type is an exact match measurement.

```
$$measurement-values-map-extension //= (
  &(tee.pceid: -80) => $tee-pceid-type
)
$tee-pceid-type /= tstr
```

5.2.12. The tee-ppid-type Measurement Extension

The tee.ppid extension enables the Attester to report the (TDB:ppid-description) as Evidence and the RVP to assert an exact-match Reference Value.

The $tee-ppid-type is an unsigned integer.

The $tee-ppid-type is an exact match measurement.

```
$$measurement-values-map-extension //= (
  &(tee.ppid: -78) => $tee-ppid-type
)
$tee-ppid-type /= uint
```

5.2.13. The tee-sgx-type Measurement Extension

The tee.sgx-type extension enables the Attester to report the (TDB:sgx-type-description) as Evidence and the RVP to assert an exact-match Reference Value.

The $tee-sgx-type is an unsigned integer.

The $tee-sgx-type is an exact match measurement.

```
$$measurement-values-map-extension //= (
  &(tee.sgx-type: -76) => $tee-sgx-type
)
$tee-sgx-type /= uint
```

5.2.14. The tee-svn-type Measurement Extension

The tee.isvsvn extension enables the Attester to report the SVN for the independent software vendor supplied component as Evidence and

the RVP to assert a Reference Value that is greater-than-or-equal to the reported SVN.

The tee.pcesvn extension enables the Attester to report the SVN for the supplied TCB component as Evidence and the RVP to assert a Reference Value that is greater-than-or-equal to the reported SVN.

The $tee-svn-type is either an unsigned integer when reported as Evidence, or a tagged numeric expression that contains an SVN and a numeric greater-than-or-equal operator. The Verifier ensures the Evidence value is greater-that-or-equal to the Reference Value.

The $tee-svn-type is a numeric when used as Endorsement or Evidence and a tagged-numeric-expression when used as a Reference Value.

```
$$measurement-values-map-extension //= (
  &(tee.isvsvn: -73) => $tee-svn-type
)

$$measurement-values-map-extension //= (
  &(tee.pcesvn: -74) => $tee-svn-type
)
$tee-svn-type /= numeric-type
$tee-svn-type /= tagged-numeric-ge
```

5.2.15. The tee-tcb-comp-svn-type Measurement Extension

The tee.tcb-comp-svn extension enables the Attester to report an array of SVN values for the TCB when asserted as Evidence and an array of tagged-numeric-ge entries when asserted as a Reference Value.

The $tee-tcb-comp-svn-type is an array containing 16 SVN values when reported as Evidence and an array of 16 expression records each containing the numeric ge operator and a reference SVN value. The Verifier evaluates each SVN in the Evidence array with the corresponding reference expression, by array position. If all Evidence values match their respective expressions, evaluation is successful. The array of SVN Evidence is accepted.

```
$$measurement-values-map-extension //= (
  &(tee.tcb-comp-svn: -79) => $tee-tcb-comp-svn-type
)
$tee-tcb-comp-svn-type /=
  [ 16*16 svn-type .within numeric-type ]
$tee-tcb-comp-svn-type /=
  [ 16*16 tagged-numeric-ge ]
```

5.2.16. The tee-tcb-eval-num-type Measurement Extension

The tee.tcb-eval-num extension enables the Attester to report a (TBD:tcb-eval-num-description) as Evidence and the RVP to assert a Reference Value expression that compares the (TBD:tcb-eval-num-description) Evidence to the Reference Value (TBD:tcb-eval-num-description) using the greater-than-or-equal operator.

The $tee-tcb-eval-num-type is an unsigned integer when reported as Evidence and a tagged numeric expression when asserted as Reference Values.

```
$$measurement-values-map-extension //= (
  &(tee-tcb-eval-num: -86) => $tee-tcb-eval-num-type
)
$tee-tcb-eval-num-type /= uint .within numeric-type
$tee-tcb-eval-num-type /= tagged-numeric-ge
```

5.2.17. The tee-tcbstatus-type Measurement Extension

The tee.tcb-status extension enables the Attester to report the status of the TEE trusted computing base (TCB) as an array of status strings, as Evidence, and the RVP to assert an array of status arrays as Reference Values where the Evidence array is a subset of the reference status arrays.

The $tee-tcbstatus-type is a status array with a set expressions containing the subset operator when used as Evidence. When used as a Reference Value it is an array of status arrays.

```
$$measurement-values-map-extension //= (
  &(tee.tcbstatus: -88) => $tee-tcbstatus-type
)
$tee-tcbstatus-type /= ([ + tstr ])
$tee-tcbstatus-type /= tagged-exp-subset
```

5.2.18. The tee-vendor-type Measurement Extension

The tee.vendor extension enables the Attester to report the TEE vendor name as Evidence and for the RVP to assert the TEE vendor name.

The $tee-vendor-type is a string containing the vendor name as a string. The vendor string in Evidence must exactly match the vendor string in Reference Values.

The $tee-vendor-type is an exact match measurement.

```
$$measurement-values-map-extension //= (
  &(tee.vendor: -70) => $tee-vendor-type
)
$tee-vendor-type /= tstr
```

6. Intel Evidence Profile

Evidence may be integrity protected in various ways including: certificates [RFC5280], SPDM transcript [DMTF.SPDM], and CBOR web token (CWT) [RFC8392]. Evidence contained in a certificate may be encoded using DiceTcbInfo and DiceTcbInfoSeq [DICE.Attest]. Evidence contained in an SPDM payload may be encoded using the SPDM Measurement Block [DMTF.SPDM]. Evidence may be formatted as concise-evidence [TCG.concise-evidence] and included in an alias certificate or an SPDM Measurement Manifest.

The DiceTcbInfo and SPDM Evidence formats can be translated to the CoMID schema. The concise evidence format is native to CoMID [I-D.ietf-rats-corim]. This profile documents evidence mapping from DiceTcbInfo and SPDM Measurement Block to the CoMID schema, as defined by [I-D.ietf-rats-corim].

The CoMID schema extensions defined by this profile, see Section 5.2, are applied to concise-evidence so that Verifiers that support this profile can consistently apply a common schema across Evidence, Reference Values, and Endorsements.

6.1. Evidence Hierarchy

Evidence hierarchy refers to SGX layering where the SGX Platform Services Enclave (PSE) collects measurements of the Quoting Enclave (QE) and the Quoting Enclaves collect measurments of their respective ISV enclaves (ISVE). A hierarchy of Evidence consisting of one PCE Evidence, one QE Evidence and one ISVE Evidence.

A complex device may have multiple roots of trust, such as [DICE.engine], each contributing an evidence hierarchy that results in several Evidence "chains", that together, constitute a complete Evidence hierarchy for the Attester device.

The Evidence hierarchy should form a spanning tree that contains all Attester Evidence. All Attesting Environments within the device produce the spanning tree. CoRIM manifests contain Reference Values for the spanning tree so that Verifiers do not assume the spanning tree is defined by Evidence. Note that a failure or comporomise within the Attester device could result in a portion of the spanning tree being omitted.

Example spanning tree:

- A DICE certificate chain with a DiceTcbInfo extension, a DiceTcbInfoSeq extension, and a ConceptualMessageWrapper (CMW) [I-D.ftbs-rats-msg-wrap] extension containing a CBOR-encoded tagged-concise-evidence.

- An SPMD alias intermediate certification chain containing a CMW extension, and an SPDM measurement manifest containing tagged-concise-evidence.

6.2. Concise Evidence

Concise evidence is a CDDL representation of an evidence schema that extends CoMID and CoSWID [I-D.ietf-sacm-coswid] schemas. Nevertheless, evidence describes the actual state of the Attester. tagged-concise-evidence is a CBOR tag for a concise evidence [TCG.concise-evidence]. This profile is compatible with tagged-concise-evicence. CoRIM schema extensions defined by this profile are inherited by tagged-concise-evidence through measurement-values-map extensions.

The concise evidence schema is defined as follows:

```
tagged-concise-evidence = #6.571(concise-evidence-map)
concise-evidence = concise-evidence-map
concise-evidence-map = {
  &(ce.ev-triples: 0) => ev-triples-map
  ? &(ce.evidence-id: 1) => $evidence-id-type-choice
  * $$concise-evidence-map-extension
}
$evidence-id-type-choice /= tagged-uuid-type
; additional evidence identifier types may be added here

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

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

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

7. Intel Verifier Profile

The verifier algorithm in this document describes the actions of a simplified Verifier that may lack performance optimizations. A verifier implementation that appears outwardly identical to the Verifier described here is treated as meeting this profile.

The Intel verifier profile builds on the verifier defined in Section 5 of [I-D.ietf-rats-corim]. This profile extends the verifier to recognize the expressions operator extensions defined by this profile. For example, if a reference numeric value of 15, the expressions operator representation is a CBOR tagged array containing the operator, gt, which is CBOR encoded as 1, followed by the reference value 15, which is a numeric-type. The reference value might be: #6.60010([1, 15]), while the evidence value is simply a numeric-type, such as '14'. The verifier compares 14 to 15, evaluating whether 14 is greater-than 15.

7.1. Complex Expressions

Complex expressions assess whether the Target Environment is in a particular actual state before asserting additional claims. For example, if an SGX enclave has an svn value that is less than the prescribed minimum svn, the enclave status may be considered "OutOfDate" or may have a known security advisory. The CoMID conditional-endorsement-triples or conditional-endorsement-series-triples describe complex expressions.

This profile uses these triples with the reference measurement values extensions described in Section 5.2.

8. Reporting Attestation Results

Attestation verification can be performed by a pipeline consisting of multiple stages where each input manifest demarks a stage. The final stage prepares Attestation Results according to Relying Party specifications. This profile expects the Relying Party will, in some fashion, negotiate the expected results format. The Attestation Results format may expect a summary result such as [I-D.ietf-rats-ar4si] only, or may expect the accepted-claims in its entirety.

The precise Attestation Results format used, if negotiated by Verifier and Relying Party, should reference this profile to acknowledge that the Relying Party and Verifier both support the schema extensions defined in this document.

9. Security Considerations

TODO Security

10. IANA Considerations

This document uses the IANA CBOR tag registry. See [IANA.CBOR]

The document requests reservation of the following CBOR tag:

- *Requested tag: 60010
- *Data item: array
- *Semantics: a CBOR encoded array
- *Point of contact: ned.smith&intel.com
- *Description of semantics: this document

11. References

11.1. Normative References

[DICE.Attest] Trusted Computing Group (TCG), "DICE Attestation Architecture", Version 1.00, Revision 0.23 , March 2021, <https://trustedcomputinggroup.org/wp-content/uploads/DICE-Attestation-Architecture-r23-final.pdf>.

[DICE.CoRIM] Trusted Computing Group (TCG), "DICE Endorsement Architecture for Devices", Version 1.0, Revision 0.38 , November 2022, <https://trustedcomputinggroup.org/wp-content/uploads/TCG-Endorsement-Architecture-for-Devices-V1-R38_pub.pdf>.

[DICE.layer] Trusted Computing Group (TCG), "DICE Layering Architecture", Version 1.0, Revision 0.19 , July 2020, <https://trustedcomputinggroup.org/wp-content/uploads/DICE-Layering-Architecture-r19_pub.pdf>.

[I-D.ftbs-rats-msg-wrap] Birkholz, H., Smith, N., Fossati, T., and H. Tschofenig, "RATS Conceptual Messages Wrapper", Work in Progress, Internet-Draft, draft-ftbs-rats-msg-wrap-03, 15 June 2023, <https://datatracker.ietf.org/doc/html/draft-ftbs-rats-msg-wrap-03>.

[I-D.ietf-rats-corim] Birkholz, H., Fossati, T., Deshpande, Y., Smith, N., and W. Pan, "Concise Reference Integrity Manifest", Work in Progress, Internet-Draft, draft-ietf-rats-corim-01, 9 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-corim-01>.

[I-D.ietf-sacm-coswid] Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", Work in Progress, Internet-Draft, draft-ietf-sacm-coswid-24, 24 February 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-sacm-coswid-24>.

[IANA.CBOR] Internet Assigned Numbers Authority (IANA), "Concise Binary Object Representation (CBOR) Tags", September 2013, <https://www.iana.org/assignments/cbor-tags/cbor-tags.xhtml>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/

RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[TCG.concise-evidence] Trusted Computing Group (TCG), "TCG DICE Concise Evidence Binding for SPDM", Version 1.0, Revision 0.53 , June 2023.

11.2. Informative References

[DICE.engine] Trusted Computing Group (TCG), "Requirements for a Device Identifier Composition Engine", Family "2.0", Level 00 Revision 78 , March 2018, <https://trustedcomputinggroup.org/wp-content/uploads/Hardware-Requirements-for-Device-Identifier-Composition-Engine-r78_For-Publication.pdf>.

[DMTF.SPDM] Distributed Managability Task Force (DMTF), "Security Protocol and Data Mmodel (SPDM) Specification", Version 1.2.1 , May 2022, <https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.2.1.pdf>.

[I-D.birkholz-rats-epoch-markers] Birkholz, H., Fossati, T., Pan, W., and C. Bormann, "Epoch Markers", Work in Progress, Internet-Draft, draft-birkholz-rats-epoch-markers-04, 13 March 2023, <https://datatracker.ietf.org/doc/html/draft-birkholz-rats-epoch-markers-04>.

[I-D.ietf-rats-ar4si] Voit, E., Birkholz, H., Hardjono, T., Fossati, T., and V. Scarlata, "Attestation Results for Secure Interactions", Work in Progress, Internet-Draft, draft-ietf-rats-ar4si-04, 2 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-ar4si-04>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/rfc/rfc8392>.

[RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS)

Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <https://www.rfc-editor.org/rfc/rfc9334>.

Appendix A. Full Intel Profile CDDL

```
tagged-numeric-gt = #6.60010( [
    greater-than: gt .within numeric-operator,
    reference-value: numeric-type ] )
tagged-numeric-ge = #6.60010( [
    greater-than: ge .within numeric-operator,
    reference-value: numeric-type ] )
tagged-numeric-lt = #6.60010( [
    greater-than: lt .within numeric-operator,
    reference-value: numeric-type ] )
tagged-numeric-le = #6.60010( [
    greater-than: le .within numeric-operator,
    reference-value: numeric-type ] )

gt = 1
ge = 2
lt = 3
le = 4
numeric-type = integer / unsigned / float
numeric-operator = gt / ge / lt / le
numeric-expression = [ numeric-operator, numeric-type ]
tagged-numeric-expression = #6.60010(numeric-expression)

member = 6
not-member = 7

set-type = [ * any ]
set-operator = member / not-member
set-expression = [ set-operator, set-type ]
tagged-set-expression = #6.60010( set-expression )

tagged-exp-member = #6.60010([
    member .within set-operator, set-type ])

tagged-exp-not-member = #6.60010([
    not-member .within set-operator, set-type ])

subset = 8
superset = 9
disjoint = 10
set-of-set-type = [ * [ + any ]]
set-of-set-operator = subset / superset / disjoint
set-of-set-expression = [ set-of-set-operator, set-of-set-type ]
tagged-set-of-set-expression = #6.60010(set-of-set-expression)

tagged-exp-subset = #6.60010([
    subset .within set-of-set-operator, set-of-set-type ])

tagged-exp-superset = #6.60010([
    superset .within set-of-set-operator, set-of-set-type ])
```

```
tagged-exp-disjoint = #6.60010([
    disjoint .within set-of-set-operator, set-of-set-type ])

mask-eq = 1
mask-operator = mask-eq
mask-type = bstr
mask-expression = [ mask-operator, value: mask-type, mask: mask-type ]
tagged-mask-expression = #6.60010( mask-expression )

tagged-exp-mask-eq = #6.60010([
    mask-eq .within mask-operator,
    value: mask-type, mask: mask-type ] )

tagged-exp-tdate-gt = #6.60010([
    gt .within tdate-operator,
    tdate ])

tagged-exp-tdate-ge = #6.60010([
    ge .within tdate-operator,
    tdate ])

tagged-exp-tdate-lt = #6.60010([
    lt .within tdate-operator,
    tdate ])

tagged-exp-tdate-le = #6.60010([
    le .within tdate-operator,
    tdate ])

tdate-operator = numeric-operator ; converts tdate to numeric
tdate-expression = [ tdate-operator, tdate ] ;#6.0(string)
tagged-tdate-expression = #6.60010( tdate-expression )

tagged-exp-time-gt = #6.60010([
    gt .within time-operator,
    time ])

tagged-exp-time-ge = #6.60010([
    ge .within time-operator,
    time ])

tagged-exp-time-lt = #6.60010([
    lt .within time-operator,
    time ])

tagged-exp-time-le = #6.60010([
    le .within time-operator,
    time ])

time-operator = numeric-operator
```

```
time-expression = [ time-operator, time ] ;#6.1(number)
tagged-time-expression = #6.60010( time-expression )

tagged-exp-epoch-gt = #6.60010([
    gt .within epoch-operator
    grace-period: epoch-seconds-type ])

tagged-exp-epoch-ge = #6.60010([
    ge .within epoch-operator
    grace-period: epoch-seconds-type ])

tagged-exp-epoch-lt = #6.60010([
    lt .within epoch-operator
    grace-period: epoch-seconds-type ])

tagged-exp-epoch-le = #6.60010([
    le .within epoch-operator
    grace-period: epoch-seconds-type ])

epoch-operator = numeric-operator
epoch-seconds-type = int
epoch-expression = [
    epoch-operator,
    grace-period: epoch-seconds-type,
    ? $tagged-epoch-id
]
tagged-epoch-expression = #6.60010( epoch-expression )
$tagged-epoch-id /= tdate
$epoch-timestamp-type /= $tagged-epoch-id

$$measurement-values-map-extension //= (
  &(tee.advisory-ids: -89) => $tee-advisory-ids-type
)
$tee-advisory-ids-type /= ([ + tstr  ])
$tee-advisory-ids-type /= tagged-exp-disjoint

$$measurement-values-map-extension //= (
  &(tee.attributes: -82) => $tee-attributes-type
)
$tee-attributes-type /= mask-type
$tee-attributes-type /= tagged-exp-mask-eq

$$measurement-values-map-extension //= (
  &(tee.tcbdate: -72) => $tee-date-type
)
$tee-date-type /= tdate
$tee-date-type /= tagged-exp-tdate-ge
$tee-date-type /= tagged-exp-epoch-ge

$$measurement-values-map-extension //= (
```

```
  &(tee.mrenclave: -83) => $tee-digest-type
)
$$measurement-values-map-extension //= (
  &(tee.mrsigner: -84) => $tee-digest-type
)
$tee-digest-type /= hash-entry .within set-type
$tee-digest-type /= tagged-exp-member

$$measurement-values-map-extension //= (
  &(tee.epoch: -90) => $tee-epoch-type
)
$tee-epoch-type /= $epoch-timestamp-type
$tee-epoch-type /= tagged-exp-epoch-gt

$$measurement-values-map-extension //= (
  &(tee.fmspc: -75) => $tee-fmspc-type
)
$tee-fmspc-type /= bstr .size 6

$$measurement-values-map-extension //= (
  &(tee.instance-id: -77) => $tee-instance-id-type
)
$tee-instance-id-type /= uint

$$measurement-values-map-extension //= (
  &(tee.isvprodid: -85) => $tee-isvprodid-type
)
$tee-isvprodid-type /= uint

$$measurement-values-map-extension //= (
  &(tee.miscselect: -81) => $tee-miscselect-type
)
$tee-miscselect-type /= mask-type
$tee-miscselect-type /= tagged-exp-mask-eq

$$measurement-values-map-extension //= (
  &(tee.model: -71) => $tee-model-type
)
$tee-model-type /= tstr

$$measurement-values-map-extension //= (
  &(tee.pceid: -80) => $tee-pceid-type
)
$tee-pceid-type /= tstr

$$measurement-values-map-extension //= (
  &(tee.ppid: -78) => $tee-ppid-type
)
$tee-ppid-type /= uint
```

```
$$measurement-values-map-extension //= (
  &(tee.sgx-type: -76) => $tee-sgx-type
)
$tee-sgx-type /= uint

$$measurement-values-map-extension //= (
  &(tee.isvsvn: -73) => $tee-svn-type
)

$$measurement-values-map-extension //= (
  &(tee.pcesvn: -74) => $tee-svn-type
)
$tee-svn-type /= numeric-type
$tee-svn-type /= tagged-numeric-ge

$$measurement-values-map-extension //= (
  &(tee.tcb-comp-svn: -79) => $tee-tcb-comp-svn-type
)
$tee-tcb-comp-svn-type /=
  [ 16*16 svn-type .within numeric-type ]
$tee-tcb-comp-svn-type /=
  [ 16*16 tagged-numeric-ge ]

$$measurement-values-map-extension //= (
  &(tee-tcb-eval-num: -86) => $tee-tcb-eval-num-type
)
$tee-tcb-eval-num-type /= uint .within numeric-type
$tee-tcb-eval-num-type /= tagged-numeric-ge

$$measurement-values-map-extension //= (
  &(tee.tcbstatus: -88) => $tee-tcbstatus-type
)
$tee-tcbstatus-type /= ([ + tstr ])
$tee-tcbstatus-type /= tagged-exp-subset

$$measurement-values-map-extension //= (
  &(tee.vendor: -70) => $tee-vendor-type
)
$tee-vendor-type /= tstr
```

```
tagged-concise-evidence = #6.571(concise-evidence-map)
concise-evidence = concise-evidence-map
concise-evidence-map = {
  &(ce.ev-triples: 0) => ev-triples-map
  ? &(ce.evidence-id: 1) => $evidence-id-type-choice
  * $$concise-evidence-map-extension
}
$evidence-id-type-choice /= tagged-uuid-type
; additional evidence identifier types may be added here

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

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

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

Acknowledgments

TODO acknowledge.

Authors' Addresses

Shanwei Cen
Intel Corporation

Email: shanwei.cen@intel.com

Andrew Draper
Intel Corporation

Email: andrew.draper@intel.com

Ned Smith
Intel Corporation

Email: ned.smith@intel.com