Workgroup: Remote ATtestation ProcedureS
Internet-Draft: draft-ietf-rats-corim-03
Published: 23 October 2023
Intended Status: Standards Track
Expires: 25 April 2024
Authors: H. Birkholz
Fraunhofer SIT
T. Fossati
arm
Y. Deshpande
arm
N. Smith
Intel
W. Pan
Huawei Technologies

Concise Reference Integrity Manifest

Abstract

Remote Attestation Procedures (RATS) enable Relying Parties to assess the trustworthiness of a remote Attester and therefore to decide whether to engage in secure interactions with it. Evidence about trustworthiness can be rather complex and it is deemed unrealistic that every Relying Party is capable of the appraisal of Evidence. Therefore that burden is typically offloaded to a Verifier. In order to conduct Evidence appraisal, a Verifier requires not only fresh Evidence from an Attester, but also trusted Endorsements and Reference Values from Endorsers and Reference Value Providers, such as manufacturers, distributors, or device owners. This document specifies the information elements for representing Endorsements and Reference Values in CBOR format.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 25 April 2024.

Table of Contents

1. Introduction

In order to conduct Evidence appraisal, a Verifier requires not only fresh Evidence from an Attester, but also trusted Endorsements (e.g., test results or certification data) and Reference Values (e.g., the version or digest of a firmware component) associated with the Attester. Such Endorsements and Reference Values are obtained from the relevant supply chain actors, such as manufacturers, distributors, or device owners. In a complex supply chain, it is likely that multiple actors will produce these values at different points in time. Besides, one supply chain actor will only provide the subset of characteristics that they know about the Attester. Attesters vary from one vendor to another, and for a given vendor from one product to another. Not only Attesters can evolve and therefore new measurement types need to be expressed, but an Endorser may also want to provide new security relevant attributes about an Attester at a future point in time.

This document specifies Concise Reference Integrity Manifests (CoRIM) a CBOR [STD94] based data model addressing the above challenges by using an extensible format common to all supply chain actors and

Verifiers. CoRIM enables Verifiers to reconcile a complex and scattered supply chain into a single homogeneous view.

1.1. Terminology and Requirements Language

This document uses terms and concepts defined by the RATS architecture. For a complete glossary see Section 4 of [RFC9334].

The terminology from CBOR [STD94], CDDL [RFC8610] and COSE [STD96] applies; in particular, CBOR diagnostic notation is defined in Section 8 of [STD94] and Appendix G of [RFC8610].

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.

1.2. CDDL Typographical Conventions

The CDDL definitions in this document follow the naming conventions illustrated in Table 1.

Type trait	Example	Typographical convention
extensible type choice	int / text / ...	$NAME-type-choice
closed type choice	int / text	NAME-type-choice
group choice	(1 => int // 2 => text)	$$NAME-group-choice
group	(1 => int, 2 => text)	NAME-group
type	int	NAME-type
tagged type	#6.123(int)	tagged-NAME-type
map	{ 1 => int, 2 => text }	NAME-map
flags	&(a: 1, b: 2)	NAME-flags

Table 1: Type Traits & Typographical Conventions

1.3. Common Types

The following CDDL types are used in both CoRIM and CoMID.

1.3.1. Non-Empty

The non-empty generic type is used to express that a map with only optional members MUST at least include one of the members.

```
non-empty<M> = (M) .and ({ + any => any })
```

1.3.2. Entity

The entity-map is a generic type describing an organization responsible for the contents of a manifest. It is instantiated by supplying two parameters:

- A role-type-choice, i.e., a selection of roles that entities of the instantiated type can claim

- An extension-socket, i.e., a CDDL socket that can be used to extend the attributes associated with entities of the instantiated type

```
entity-map<role-type-choice, extension-socket> = {
  &(entity-name: 0) => $entity-name-type-choice
  ? &(reg-id: 1) => uri
  &(role: 2) => [ + role-type-choice ]
  * extension-socket
}

$entity-name-type-choice /= text
```

The following describes each member of the entity-map.

- entity-name (index 0): The name of entity which is responsible for the action(s) as defined by the role. $entity-name-type-choice can only be text. Other specifications can extend the $entity-name-type-choice (see Section 8.4).

- reg-id (index 1): A URI associated with the organization that owns the entity name

- role (index 2): A type choice defining the roles that the entity is claiming. The role is supplied as a parameter at the time the entity-map generic is instantiated.

- extension-socket: A CDDL socket used to add new information structures to the entity-map.

Examples of how the entity-map generic is instantiated can be found in Section 2.1.5 and Section 3.1.2.

1.3.3. Validity

A validity-map represents the time interval during which the signer warrants that it will maintain information about the status of the signed object (e.g., a manifest).

In a validity-map, both ends of the interval are encoded as epoch-based date/time as per Section 3.4.2 of [STD94].

```
validity-map = {
  ? &(not-before: 0) => time
  &(not-after: 1) => time
}
```

*not-before (index 0): the date on which the signed manifest validity period begins

*not-after (index 1): the date on which the signed manifest validity period ends

1.3.4. UUID

Used to tag a byte string as a binary UUID defined in Section 4.1.2. of [RFC4122].

```
uuid-type = bytes .size 16
tagged-uuid-type = #6.37(uuid-type)
```

1.3.5. UEID

Used to tag a byte string as Universal Entity ID Claim (UUID) defined in Section 4.2.1 of [I-D.ietf-rats-eat].

```
ueid-type = bytes .size 33
tagged-ueid-type = #6.550(ueid-type)
```

1.3.6. OID

Used to tag a byte string as the BER encoding [X.690] of an absolute object identifier [RFC9090].

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

1.3.7. Tagged Integer Type

Used as a class identifier for the environment. It is expected that the integer value is vendor specific rather than globally meaningful. Therefore, the sibling vendor field in the class-map MUST be populated to define the namespace under which the value must be understood.

```
tagged-int-type = #6.551(int)
```

1.3.8. Digest

A digest represents the value of a hashing operation together with the hash algorithm used. The type of the digest algorithm identifier can be either int or text. When carried as an integer value, it is interpreted according to the "Named Information Hash Algorithm Registry" [IANA.named-information]. When it is carried as text, there are no requirements with regards to its format. In general, the int encoding is RECOMMENDED. The text encoding should only be used when the digest type conveys reference value measurements that are matched verbatim with Evidence that uses the same convention - e.g., Section 4.4.1.5 of [I-D.tschofenig-rats-psa-token]).

```
digest = [
  alg: (int / text),
  val: bytes
]
```

2. Concise Reference Integrity Manifest (CoRIM)

A CoRIM is a collection of tags and related metadata as described below.

Tags can be of different types:

- Concise Module ID (CoMID) tags (Section 3) contain metadata and claims about the hardware and firmware modules.
- Concise Software ID (CoSWID) tags [I-D.ietf-sacm-coswid] describe software components.
- Concise Bill of Material (CoBOM) tags (Section 4) contain the list of CoMID and CoSWID tags that the Verifier should consider as "active" at a certain point in time.

The set of tags is extensible so that future specifications can add new kinds of information. For example, Concise Trust Anchor Stores (CoTS) [I-D.ietf-rats-concise-ta-stores] is currently being defined as a standard CoRIM extension.

Each CoRIM contains a unique identifier to distinguish a CoRIM from other CoRIMs. Tracked at: https://github.com/ietf-rats-wg/draft-ietf-rats-corim/issues/73

CoRIM can also carry the following optional metadata:

- A locator, which allows discovery of possibly related RIMs
- A profile identifier, which is used to interpret the information contained in the enclosed tags. A profile allows the base CoRIM

schema to be customised to fit a specific Attester. For example, see [I-D.fdb-rats-psa-endorsements].

*A validity period, which indicates the time period for which the CoRIM contents are valid.

*Information about the supply chain entities responsible for the contents of the CoRIM and their associated roles.

A CoRIM can be signed (Section 2.2) using COSE Sign1 to provide end-to-end security to the CoRIM contents. When CoRIM is signed, the protected header carries further identifying information about the CoRIM signer. Alternatively, CoRIM can be encoded as a CBOR-tagged payload (Section 2.1) and transported over a secure channel.

The following CDDL describes the top-level CoRIM.

```
corim = tagged-concise-rim-type-choice

$concise-rim-type-choice /= tagged-corim-map
$concise-rim-type-choice /= tagged-signed-corim
```

2.1. CoRIM Map

The CDDL specification for the corim-map is as follows and this rule and its constraints must be followed when creating or validating a CoRIM map.

```
corim-map = {
  &(id: 0) => $corim-id-type-choice
  &(tags: 1) => [ + $concise-tag-type-choice ]
  ? &(dependent-rims: 2) => [ + corim-locator-map ]
  ? &(profile: 3) => $profile-type-choice
  ? &(rim-validity: 4) => validity-map
  ? &(entities: 5) => [ + corim-entity-map ]
  * $$corim-map-extension
}
```

The following describes each child item of this map.

*id (index 0): A globally unique identifier to identify a CoRIM. Described in Section 2.1.1

*tags (index 1): An array of one or more CoMID or CoSWID tags. Described in Section 2.1.2

*dependent-rims (index 2): One or more services supplying additional, possibly dependent, manifests or related files. Described in Section 2.1.3

*profile (index 3): An optional profile identifier for the tags contained in this CoRIM. The profile MUST be understood by the CoRIM processor. Failure to recognize the profile identifier MUST result in the rejection of the entire CoRIM. If missing, the profile defaults to DICE. Described in Section 2.1.4

*rim-validity (index 4): Specifies the validity period of the CoRIM. Described in Section 1.3.3

*entities (index 5): A list of entities involved in a CoRIM life-cycle. Described in Section 2.1.5

*$$corim-map-extension: This CDDL socket is used to add new information structures to the corim-map. See Section 8.3.

```
tagged-corim-map = #6.501(corim-map)
```

2.1.1. Identity

A CoRIM Identifier uniquely identifies a CoRIM instance. The base schema allows UUID and text identifiers. Other types of identifiers could be defined as needed.

```
$corim-id-type-choice /= tstr
$corim-id-type-choice /= uuid-type
```

2.1.2. Tags

A $concise-tag-type-choice is a tagged CBOR payload that carries either a CoMID (Section 3), a CoSWID [I-D.ietf-sacm-coswid], or a CoBOM Section 4.

```
$concise-tag-type-choice /= tagged-concise-swid-tag
$concise-tag-type-choice /= tagged-concise-mid-tag
$concise-tag-type-choice /= tagged-concise-bom-tag
```

2.1.3. Locator Map

The locator map contains pointers to repositories where dependent manifests, certificates, or other relevant information can be retrieved by the Verifier.

```
corim-locator-map = {
  &(href: 0) => uri
  ? &(thumbprint: 1) => digest
}
```

The following describes each child element of this type.

- *href (index 0): URI identifying the additional resource that can be fetched

- *thumbprint (index 1): expected digest of the resource referenced by href. See Section 1.3.8.

2.1.4. Profile Types

Profiling is the mechanism that allows the base CoRIM schema to be customised to fit a specific Attester.

A profile defines which of the optional parts of a CoRIM are required, which are prohibited and which extension points are exercised and how. A profile MUST NOT alter the syntax or semantics of a standard CoRIM type. A profile MAY constrain the values of a given CoRIM type to a subset of the values. A profile MAY extend the set of a given CoRIM type using the defined extension points (see Section 3.2). Exercised extension points should preserve the intent of the original semantics.

CoRIM profiles SHOULD be specified in a publicly available document.

A CoRIM profile can use one of the base CoRIM media types defined in Section 8.6.1 and Section 8.6.2 with the profile parameter set to the appropriate value. Alternatively, it MAY define and register its own media type.

A profile identifier is either an OID [RFC9090] or a URL [STD66].

The profile identifier uniquely identifies a documented profile. Any changes to the profile, even the slightest deviation, is considered a different profile that MUST have a different identifier.

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

2.1.5. Entities

The CoRIM Entity is an instantiation of the Entity generic (Section 1.3.2) using a $corim-role-type-choice.

The only role defined in this specification for a CoRIM Entity is manifest-creator.

The $$corim-entity-map-extension extension socket is empty in this specification.

```
corim-entity-map =
  entity-map<$corim-role-type-choice, $$corim-entity-map-extension>

$corim-role-type-choice /= &(manifest-creator: 1)
```

2.2. Signed CoRIM

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

Signing a CoRIM follows the procedures defined in CBOR Object Signing and Encryption [STD96]. A CoRIM tag MUST be wrapped in a COSE_Sign1 structure. The CoRIM MUST be signed by the CoRIM creator.

The following CDDL specification defines a restrictive subset of COSE header parameters that MUST be used in the protected header alongside additional information about the CoRIM encoded in a corim-meta-map (Section 2.2.2).

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

The following describes each child element of this type.

- protected: A CBOR Encoded protected header which is protected by the COSE signature. Contains information as given by Protected Header Map below.

- unprotected: A COSE header that is not protected by COSE signature.

- payload: A CBOR encoded tagged CoRIM.

- signature: A COSE signature block which is the signature over the protected and payload components of the signed CoRIM.

2.2.1. Protected Header Map

```
protected-corim-header-map = {
  &(alg-id: 1) => int
  &(content-type: 3) => "application/corim-unsigned+cbor"
  &(issuer-key-id: 4) => bstr
  &(corim-meta: 8) => bstr .cbor corim-meta-map
  * cose-label => cose-value
}
```

The following describes each child item of this map.

*alg-id (index 1): An integer that identifies a signature algorithm.

*content-type (index 3): A string that represents the "MIME Content type" carried in the CoRIM payload.

*issuer-key-id (index 4): A bit string which is a key identity pertaining to the CoRIM Issuer.

*corim-meta (index 8): A map that contains metadata associated with a signed CoRIM. Described in Section 2.2.2.

Additional data can be included in the COSE header map as per Section 3 of [STD96].

2.2.2. Meta Map

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

```
corim-meta-map = {
  &(signer: 0) => corim-signer-map
  ? &(signature-validity: 1) => validity-map
}
```

The following describes each child item of this group.

*signer (index 0): Information about the entity that signs the CoRIM. Described in Section 2.2.2.1

*signature-validity (index 1): Validity period for the CoRIM. Described in Section 1.3.3

2.2.2.1. Signer Map

```
corim-signer-map = {
  &(signer-name: 0) => $entity-name-type-choice
  ? &(signer-uri: 1) => uri
  * $$corim-signer-map-extension
}
```

*signer-name (index 0): Name of the organization that performs the signer role

*signer-uri (index 1): A URI identifying the same organization

*$$corim-signer-map-extension: Extension point for future expansion of the Signer map.

2.2.3. Unprotected CoRIM Header Map

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

3. Concise Module Identifier (CoMID)

A CoMID tag contains information about hardware, firmware, or module composition.

Each CoMID has a unique ID that is used to unambigously identify CoMID instances when cross referencing CoMID tags, for example in typed link relations, or in a CoBOM tag.

A CoMID defines several types of Claims, using "triples" semantics.

At a high level, a triple is a statement that links a subject to an object via a predicate. CoMID triples typically encode assertions made by the CoRIM author about Attesting or Target Environments and their security features, for example measurements, cryptographic key material, etc.

The set of triples is extensible. The following triples are currently defined:

*Reference Values triples: containing Reference Values that are expected to match Evidence for a given Target Environment (Section 3.1.4.2).

*Endorsed Values triples: containing "Endorsed Values", i.e., features about an Environment that do not appear in Evidence. Specific examples include testing or certification data pertaining to a module (Section 3.1.4.3).

*Device Identity triples: containing cryptographic credentials - for example, an IDevID - uniquely identifying a device (Section 3.1.4.4).

*Attestation Key triples: containing cryptographic keys that are used to verify the integrity protection on the Evidence received from the Attester (Section 3.1.4.5).

*Domain dependency triples: describing trust relationships between domains, i.e., collection of related environments and their measurements (Section 3.1.4.6).

- Domain membership triples: describing topological relationships between (sub-)modules. For example, in a composite Attester comprising multiple sub-Attesters (sub-modules), this triple can be used to define the topological relationship between lead- and sub- Attester environments (Section 3.1.4.7).

- CoMID-CoSWID linking triples: associating a Target Environment with existing CoSWID tags (Section 3.1.4.8).

3.1. Structure

The CDDL specification for the concise-mid-tag map is as follows and this rule and its constraints MUST be followed when creating or validating a CoMID tag:

```
concise-mid-tag = {
  ? &(language: 0) => text
  &(tag-identity: 1) => tag-identity-map
  ? &(entities: 2) => [ + comid-entity-map ]
  ? &(linked-tags: 3) => [ + linked-tag-map ]
  &(triples: 4) => triples-map
  * $$concise-mid-tag-extension
}
```

The following describes each member of the concise-mid-tag map.

- lang (index 0): A textual language tag that conforms with IANA "Language Subtag Registry" [IANA.language-subtag-registry]. The context of the specified language applies to all sibling and descendant textual values, unless a descendant object has defined a different language tag. Thus, a new context is established when a descendant object redefines a new language tag. All textual values within a given context MUST be considered expressed in the specified language.

- tag-identity (index 1): A tag-identity-map containing unique identification information for the CoMID. Described in Section 3.1.1.

- entities (index 2): Provides information about one or more organizations responsible for producing the CoMID tag. Described in Section 3.1.2.

- linked-tags (index 3): A list of one or more linked-tag-map (described in Section 3.1.3), providing typed relationships between this and other CoMIDs.

- triples (index 4): One or more triples providing information specific to the described module, e.g.: reference or endorsed values, cryptographic material, or structural relationship between

the described module and other modules. Described in (Section 3.1.4).

3.1.1. Tag Identity

```
tag-identity-map = {
  &(tag-id: 0) => $tag-id-type-choice
  ? &(tag-version: 1) => tag-version-type
}
```

The following describes each member of the tag-identity-map.

- tag-id (index 0): A universally unique identifier for the CoMID. Described in Section 3.1.1.1.

- tag-version (index 1): Optional versioning information for the tag-id . Described in Section 3.1.1.2.

3.1.1.1. Tag ID

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

A Tag ID is either a 16-byte binary string, or a textual identifier, uniquely referencing the CoMID. The tag identifier MUST be globally unique. Failure to ensure global uniqueness can create ambiguity in tag use since the tag-id serves as the global key for matching, lookups and linking. If represented as a 16-byte binary string, the identifier MUST be a valid universally unique identifier as defined by [RFC4122]. There are no strict guidelines on how the identifier is structured, but examples include a 16-byte GUID (e.g., class 4 UUID) [RFC4122], or a URI [STD66].

3.1.1.2. Tag Version

```
tag-version-type = uint .default 0
```

Tag Version is an integer value that indicates the specific release revision of the tag. Typically, the initial value of this field is set to 0 and the value is increased for subsequent tags produced for the same module release. This value allows a CoMID tag producer to correct an incorrect tag previously released without indicating a change to the underlying module the tag represents. For example, the tag version could be changed to add new metadata, to correct a broken link, to add a missing reference value, etc. When producing a revised tag, the new tag-version value MUST be greater than the old tag-version value.

3.1.2. Entities

```
comid-entity-map =
  entity-map<$comid-role-type-choice, $$comid-entity-map-extension>
```

The CoMID Entity is an instantiation of the Entity generic (Section 1.3.2) using a $comid-role-type-choice.

The $$comid-entity-map-extension extension socket is empty in this specification.

```
$comid-role-type-choice /= &(tag-creator: 0)
$comid-role-type-choice /= &(creator: 1)
$comid-role-type-choice /= &(maintainer: 2)
```

The roles defined for a CoMID entity are:

- *tag-creator (value 0): creator of the CoMID tag.

- *creator (value 1): original maker of the module described by the CoMID tag.

- *maintainer (value 2): an entity making changes to the module described by the CoMID tag.

3.1.3. Linked Tag

The linked tag map represents a typed relationship between the embedding CoMID tag (the source) and another CoMID tag (the target).

```
linked-tag-map = {
  &(linked-tag-id: 0) => $tag-id-type-choice
  &(tag-rel: 1) => $tag-rel-type-choice
}
```

The following describes each member of the tag-identity-map.

- *linked-tag-id (index 0): Unique identifier for the target tag. For the definition see Section 3.1.1.1.

- *tag-rel (index 1): the kind of relation linking the source tag to the target identified by linked-tag-id.

```
$tag-rel-type-choice /= &(supplements: 0)
$tag-rel-type-choice /= &(replaces: 1)
```

The relations defined in this specification are:

- *supplements (value 0): the source tag provides additional information about the module described in the target tag.

*replaces (value 1): the source tag corrects erroneous information contained in the target tag. The information in the target MUST be disregarded.

3.1.4. Triples

The triples-map contains all the CoMID triples broken down per category. Not all category need to be present but at least one category MUST be present and contain at least one entry.

```
triples-map = non-empty<{
  ? &(reference-triples: 0) =>
    [ + reference-triple-record ]
  ? &(endorsed-triples: 1) =>
    [ + endorsed-triple-record ]
  ? &(identity-triples: 2) =>
    [ + identity-triple-record ]
  ? &(attest-key-triples: 3) =>
    [ + attest-key-triple-record ]
  ? &(dependency-triples: 4) =>
    [ + domain-dependency-triple-record ]
  ? &(membership-triples: 5) =>
    [ + domain-membership-triple-record ]
  ? &(coswid-triples: 6) =>
    [ + coswid-triple-record ]
  ? &(conditional-endorsement-series-triples: 8) =>
    [ + conditional-endorsement-series-triple-record ]
  ? &(conditional-endorsement-triples: 9) =>
    [ + conditional-endorsement-triple-record ]
  * $$triples-map-extension
}>
```

The following describes each member of the triples-map:

*reference-triples (index 0): Triples containing reference values. Described in Section 3.1.4.2.

*endorsed-triples (index 1): Triples containing endorsed values. Described in Section 3.1.4.3.

*identity-triples (index 2): Triples containing identity credentials. Described in Section 3.1.4.4.

*attest-key-triples (index 3): Triples containing verification keys associated with attesting environments. Described in Section 3.1.4.5.

*dependency-triples (index 4): Triples describing trust relationships between domains. Described in Section 3.1.4.6.

*membership-triples (index 5): Triples describing topological relationships between (sub-)modules. Described in Section 3.1.4.7.

*coswid-triples (index 6): Triples associating modules with existing CoSWID tags. Described in Section 3.1.4.8.

*conditional-endorsement-series-triples (index 8) Triples describing a series of conditional Endorsements based on the acceptance of a stateful environment. Described in Section 3.1.4.9.

*conditional-endorsement-triples (index 9) Triples describing conditional Endorsement based on the acceptance of a stateful environment. Described in Section 3.1.4.10.

3.1.4.1. Common Types

3.1.4.1.1. Environment

An environment-map may be used to represent a whole Attester, an Attesting Environment, or a Target Environment. The exact semantic depends on the context (triple) in which the environment is used.

An environment is named after a class, instance or group identifier (or a combination thereof).

```
environment-map = non-empty<{
  ? &(class: 0) => class-map
  ? &(instance: 1) => $instance-id-type-choice
  ? &(group: 2) => $group-id-type-choice
}>
```

The following describes each member of the environment-map:

*class (index 0): Contains "class" attributes associated with the module. Described in Section 3.1.4.1.2.

*instance (index 1): Contains a unique identifier of a module's instance. See Section 3.1.4.1.3.

*group (index 2): identifier for a group of instances, e.g., if an anonymization scheme is used.

3.1.4.1.2. Class

The Class name consists of class attributes that distinguish the class of environment from other classes. The class attributes include class-id, vendor, model, layer, and index. The CoMID author determines which attributes are needed.

```
class-map = non-empty<{
  ? &(class-id: 0) => $class-id-type-choice
  ? &(vendor: 1) => tstr
  ? &(model: 2) => tstr
  ? &(layer: 3) => uint
  ? &(index: 4) => uint
}>

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

The following describes each member of the class-map:

- class-id (index 0): Identifies the environment via a well-known identifier. Typically, class-id is an object identifier (OID) or universally unique identifier (UUID). Use of this attribute is preferred.

- vendor (index 1): Identifies the entity responsible for choosing values for the other class attributes that do not already have naming authority.

- model (index 2): Describes a product, generation, and family. If populated, vendor MUST also be populated.

- layer (index 3): Is used to capture where in a sequence the environment exists. For example, the order in which bootstrap code is executed may have security relevance.

- index (index 4): Is used when there are clones (i.e., multiple instances) of the same class of environment. Each clone is given a different index value to disambiguate it from the other clones. For example, given a chassis with several network interface controllers (NIC), each NIC can be given a different index value.

3.1.4.1.3. Instance

An instance carries a unique identifier that is reliably bound to a Target Environment that is an instance of the Attester.

The types defined for an instance identifier are CBOR tagged expressions of UEID, UUID, or cryptographic key identifier.

```
$instance-id-type-choice /= tagged-ueid-type
$instance-id-type-choice /= tagged-uuid-type
$instance-id-type-choice /= $crypto-key-type-choice
```

3.1.4.1.4. Group

A group carries a unique identifier that is reliably bound to a group of Attesters, for example when a number of Attester are hidden in the same anonymity set.

The type defined for a group identified is UUID.

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

3.1.4.1.5. Measurements

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

An environment has one or more measurable elements. Each element can have a dedicated measurement or multiple elements could be combined into a single measurement. Measurements can have class, instance or group scope. This is typically determined by the triple's environment.

Class measurements apply generally to all the Attesters in the given class. Instance measurements apply to a specific Attester instance. Environments identified by a class identifier have measurements that are common to the class. Environments identified by an instance identifier have measurements that are specific to that instance.

The supply chain entity that is responsible for providing the the measurements (i.e. Reference Values or Endorsed Values) is by default the CoRIM signer. If a different entity is authorized to provide measurement values, the authorized-by statement can be supplied in the measurement-map.

```
measurement-map = {
  ? &(mkey: 0) => $measured-element-type-choice
  &(mval: 1) => measurement-values-map
  ? &(authorized-by: 2) => [ + $crypto-key-type-choice ]
}
```

The following describes each member of the measurement-map:

- *mkey (index 0): An optional unique identifier of the measured (sub-)environment. See Section 3.1.4.1.5.1.

- *mval (index 1): The measurements associated with the (sub-)environment. Described in Section 3.1.4.1.5.2.

*authorized-by (index 2): The cryptographic identity of the individual or organization that is the designated authority for this measurement. For example, producer of the measurement or a delegated supplier.

3.1.4.1.5.1. Measurement Keys

The types defined for a measurement identifier are OID, UUID or uint.

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

3.1.4.1.5.2. Measurement Values

A measurement-values-map contains measurements associated with a certain environment. Depending on the context (triple) in which they are found, elements in a measurement-values-map can represent class or instance measurements. Note that some of the elements have instance scope only.

Measurement values may support use cases beyond Verifier appraisal. Typically, a Relying Party determines if additional processing is desirable and whether the processing is applied by the Verifier or the Relying Party.

```
measurement-values-map = non-empty<{
  ? &(version: 0) => version-map
  ? &(svn: 1) => svn-type-choice
  ? &(digests: 2) => [ + digest ]
  ? &(flags: 3) => flags-map
  ? (
      &(raw-value: 4) => $raw-value-type-choice,
      ? &(raw-value-mask: 5) => raw-value-mask-type
    )
  ? &(mac-addr: 6) => mac-addr-type-choice
  ? &(ip-addr: 7) =>  ip-addr-type-choice
  ? &(serial-number: 8) => text
  ? &(ueid: 9) => ueid-type
  ? &(uuid: 10) => uuid-type
  ? &(name: 11) => text
  ? &(cryptokeys: 12) => [ + $crypto-key-type-choice ]
  * $$measurement-values-map-extension
}>
```

The following describes each member of the measurement-values-map.

*version (index 0): Typically changes whenever the measured environment is updated. Described in Section 3.1.4.1.5.3.

*svn (index 1): The security version number typically changes only when a security relevant change is made to the measured environment. Described in Section 3.1.4.1.5.4.

*digests (index 2): Contains the digest(s) of the measured environment together with the respective hash algorithm used in the process. See Section 1.3.8.

*flags (index 3): Describes security relevant operational modes. For example, whether the environment is in a debug mode, recovery mode, not fully configured, not secure, not replay protected or not integrity protected. The flags field indicates which operational modes are currently associated with measured environment. Described in Section 3.1.4.1.5.5.

*raw-value (index 4): Contains the actual (not hashed) value of the element. An optional raw-value-mask (index 5) indicates which bits in the raw-value field are relevant for verification. A mask of all ones ("1") means all bits in the raw-value field are relevant. Multiple values could be combined to create a single raw-value attribute. The vendor determines how to pack multiple values into a single raw-value structure. The same packing format is used when collecting Evidence so that Reference Values and collected values are bit-wise comparable. The vendor determines the encoding of raw-value and the corresponding raw-value-mask.

*mac-addr (index 6): A EUI-48 or EUI-64 MAC address associated with the measured environment. Described in Section 3.1.4.1.5.7.

*ip-addr (index 7): An IPv4 or IPv6 address associated with the measured environment. Described in Section 3.1.4.1.5.7.

*serial-number (index 8): A text string representing the product serial number.

*ueid (index 9): UEID associated with the measured environment. See Section 1.3.5.

*uuid (index 10): UUID associated with the measured environment. See Section 1.3.4.

*name (index 11): a name associated with the measured environment.

*cryptokeys (index 12): identifies cryptographic keys that are protected by the Target Environment See Section 3.1.4.1.6 for the supported formats. An Attesting Environment determines that keys are protected as part of Claims collection. Appraisal verifies that, for each value in cryptokeys, there is a matching Reference Value entry. Matching is described in Section 5.4.2.3.4.

3.1.4.1.5.3. Version

A version-map contains details about the versioning of a measured environment.

```
version-map = {
  &(version: 0) => text
  ? &(version-scheme: 1) => $version-scheme
}
```

The following describes each member of the version-map:

- *version (index 0): the version string

- *version-scheme (index 1): an optional indicator of the versioning convention used in the version attribute. Defined in Section 4.1 of [I-D.ietf-sacm-coswid]. The CDDL is copied below for convenience.

```
$version-scheme /= &(multipartnumeric: 1)
$version-scheme /= &(multipartnumeric-suffix: 2)
$version-scheme /= &(alphanumeric: 3)
$version-scheme /= &(decimal: 4)
$version-scheme /= &(semver: 16384)
$version-scheme /= int / text
```

3.1.4.1.5.4. Security Version Number

The following details the security version number (svn) and the minimum security version number (min-svn) statements. A security version number is used to track changes to an object (e.g., a secure enclave, a boot loader executable, a configuration file, etc.) that are security relevant. Rollback of a security relevant change is considered to be an attack vector, as such, security version numbers can't be decremented. If a security relevant flaw is discovered in the Target Environment and subsequently fiexed, the svn value is typically incremented.

There may be several revisions to a Target Environment that are in use at the same time. If there are multiple revisions with different svn values, the revision with a lower svn value may or may not be in a security critical condition. The Endorser may provide a minimum security version number using min-svn to specify the lowest svn value that is acceptable. svn values that are equal to or greater than min-svn do not signal a security critical condition. svn values that are below min-svn are in a security critical condition that is unsafe for normal use.

The svn-type-choice measurement consists of a tagged-svn or tagged-min-svn value. The tagged-svn and tagged-min-svn tags are CBOR tags with the values #6.552 and #6.553 respectively.

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

3.1.4.1.5.5. Flags

The flags-map measurement describes a number of boolean operational modes. If a flags-map value is not specified, then the operational mode is unknown.

```
flags-map = {
  ? &(is-configured: 0) => bool
  ? &(is-secure: 1) => bool
  ? &(is-recovery: 2) => bool
  ? &(is-debug: 3) => bool
  ? &(is-replay-protected: 4) => bool
  ? &(is-integrity-protected: 5) => bool
  ? &(is-runtime-meas: 6) => bool
  ? &(is-immutable: 7) => bool
  ? &(is-tcb: 8) => bool
  ? &(is-confidentiality-protected: 9) => bool
  * $$flags-map-extension
}
```

The following describes each member of the flags-map:

- is-configured (index 0): If the flag is true, the measured environment is fully configured for normal operation.

- is-secure (index 1): If the flag is true, the measured environment's configurable security settings are fully enabled.

- is-recovery (index 2): If the flag is true, the measured environment is in recovery mode.

- is-debug (index 3): If the flag is true, the measured environment is in a debug enabled mode.

- is-replay-protected (index 4): If the flag is true, the measured environment is protected from replay by a previous image that differs from the current image.

*is-integrity-protected (index 5): If the flag is true, the measured environment is protected from unauthorized update.

*is-runtime-meas (index 6): If the flag is true, the measured environment is measured after being loaded into memory.

*is-immutable (index 7): If the flag is true, the measured environment is immutable.

*is-tcb (index 8): If the flag is true, the measured environment is a trusted computing base.

*is-confidentiality-protected (index 9): If the flag is true, the measured environment is confidentiality protected. For example, if the measured environment consists of memory, the sensitive values in memory are encrypted.

3.1.4.1.5.6. Raw Values Types

Raw value measurements are typically vendor defined values that are checked by Verifiers for consistency only, since the security relevance is opaque to Verifiers.

There are two parts to a raw-value-group, a measurement and an optional mask. The default raw value measurement is a CBOR tagged bstr. Additional raw value types can be defined, but must be CBOR tagged so that parsers can distinguish between the various semantics of type values.

The mask is applied by the Verifier as part of appraisal. Only the raw value bits with corresponding TRUE mask bits are compared during appraisal.

When a new raw value type is defined, the convention for applying the mask is also defined. Typically, a CoRIM profile is used to define new raw values and mask semantics.

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

raw-value-mask-type = bytes
```

3.1.4.1.5.7. Address Types

The types or associating addressing information to a measured environment are:

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

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

3.1.4.1.6. Crypto Keys

A cryptographic key can be one of the following formats:

- *tagged-pkix-base64-key-type: PEM encoded SubjectPublicKeyInfo. Defined in Section 13 of [RFC7468].

- *tagged-pkix-base64-cert-type: PEM encoded X.509 public key certificate. Defined in Section 5 of [RFC7468].

- *tagged-pkix-base64-cert-path-type: X.509 certificate chain created by the concatenation of as many PEM encoded X.509 certificates as needed. The certificates MUST be concatenated in order so that each directly certifies the one preceding.

- *tagged-cose-key-type: CBOR encoded COSE_Key or COSE_KeySet. Defined in Section 7 of [STD96]

A cryptographic key digest can be one of the following formats:

- *tagged-thumbprint-type: a digest of a raw public key. The digest value may be used to find the public key if contained in a lookup table.

- *tagged-cert-thumbprint-type: a digest of a certificate. The digest value may be used to find the certificate if contained in a lookup table.

- *tagged-cert-path-thumbprint-type: a digest of a certification path. The digest value may be used to find the certificate path if contained in a lookup table.

In a split Verifier scenario, a first Verifier may verify the signature of a cryptographic key then compute a digest of the key that is forwarded to a second Verifier. The second Verifier completes the signature verification by performing certificate path validation, revocation checks, and trust anchor checks.

```
$crypto-key-type-choice /= tagged-pkix-base64-key-type
$crypto-key-type-choice /= tagged-pkix-base64-cert-type
$crypto-key-type-choice /= tagged-pkix-base64-cert-path-type
$crypto-key-type-choice /= tagged-cose-key-type
$crypto-key-type-choice /= tagged-thumbprint-type
$crypto-key-type-choice /= tagged-cert-thumbprint-type
$crypto-key-type-choice /= tagged-cert-path-thumbprint-type

tagged-pkix-base64-key-type = #6.554(tstr)
tagged-pkix-base64-cert-type = #6.555(tstr)
tagged-pkix-base64-cert-path-type = #6.556(tstr)
tagged-thumbprint-type = #6.557(digest)
tagged-cose-key-type = #6.558(COSE_KeySet / COSE_Key)
tagged-cert-thumbprint-type = #6.559(digest)
tagged-cert-path-thumbprint-type = #6.561(digest)
```

3.1.4.1.7. Domain Types

A domain is a context for bundling a collection of related environments and their measurements.

Three types are defined: uint and text for local scope, UUID for global scope.

```
$domain-type-choice /= uint
$domain-type-choice /= text
$domain-type-choice /= tagged-uuid-type
$domain-type-choice /= tagged-oid-type
```

3.1.4.2. Reference Values Triple

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

```
reference-triple-record = [
  environment-map
  measurement-map
]
```

3.1.4.3. Endorsed Values Triple

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

```
endorsed-triple-record = [
  environment-map
  measurement-map
]
```

3.1.4.4. Device Identity Triple

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

```
identity-triple-record = [
  environment-map
  [ + $crypto-key-type-choice ]
]
```

3.1.4.5. Attestation Keys Triple

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

```
attest-key-triple-record = [
  environment-map
  [ + $crypto-key-type-choice ]
]
```

3.1.4.6. Domain Dependency Triple

A Domain Dependency triple defines trust dependencies between measurement sources. The subject identifies a domain (Section 3.1.4.1.7) that has a predicate relationship to the object containing one or more dependent domains. Dependency means the subject domain's trustworthiness properties rely on the object domain(s) trustworthiness having been established before the trustworthiness properties of the subject domain exists.

```
domain-dependency-triple-record = [
  $domain-type-choice
  [ + $domain-type-choice ]
]
```

3.1.4.7. Domain Membership Triple

A Domain Membership triple assigns domain membership to environments. The subject identifies a domain (Section 3.1.4.1.7) that has a

predicate relationship to the object containing one or more environments. Endorsed environments (Section 3.1.4.3) membership is conditional upon successful matching of Reference Values (Section 3.1.4.2) to Evidence.

```
domain-membership-triple-record = [
  $domain-type-choice
  [ + environment-map ]
]
```

3.1.4.8. CoMID-CoSWID Linking Triple

A CoSWID triple relates reference measurements contained in one or more CoSWIDs to a Target Environment. The subject identifies a Target Environment, the object one or more unique tag identifiers of existing CoSWIDs, and the predicate asserts that these contain the expected (i.e., reference) measurements for the Target Environment.

```
coswid-triple-record = [
  environment-map
  [ + concise-swid-tag-id ]
]

concise-swid-tag-id = text / bstr .size 16
```

3.1.4.9. Conditional Endorsement Series Triple

A Conditional Endorsement Series triple uses a stateful environment, (i.e., stateful-environment-record), that identifies a Target Environment based on an environment-map plus the measurement-map measurements that have matching Evidence.

The stateful Target Environment is a triple subject that MUST be satisfied before the series triple object is matched.

```
; an environment with a set of measurements that must match evidence
stateful-environment-record = [
  environment-map,
  measurement-map
]
```

The series object is an array of conditional-series-record that has both Reference and Endorsed Values. Each conditional-series-record record is evaluated in the order it appears in the series array. The Endorsed Values are accepted if the Reference Values in a conditional-series-record matches Evidence. The first conditional-series-record that successfully matches Evidence terminates the series and the matching Reference Values as well as the Endorsed Values are accepted. If none of the Reference Values in the series

match Evidence, the triple is not matched, and no Claims are accepted.

The authorized-by value in measurement-map in the stateful environment, if present, applies to all measurements in the triple, including conditional-series-record records.

```
conditional-endorsement-series-triple-record = [
  stateful-environment-record
  ; order matters: the first matching record wins and halts matching
  [ + conditional-series-record ]
]

conditional-series-record = [
  ; reference values to be matched against evidence
  refv: measurement-values-map
  ; endorsed values that apply in case revf matches
  endv: measurement-values-map
]
```

3.1.4.10. Conditional Endorsement Triple

A Conditional Endorsement triple uses a stateful environment, (i.e., stateful-environment-record), that identifies a Target Environment based on an environment-map plus the measurement-map measurements that have matching Evidence.

The stateful Target Environment is a triple subject that MUST be satisfied before the Endorsed Values in the triple object are accepted.

```
; an environment with a set of measurements that must match evidence
stateful-environment-record = [
  environment-map,
  measurement-map
]
```

The authorized-by value in measurement-map in the stateful environment, if present, applies to all measurements in the triple, including those in measurement-values-map.

```
conditional-endorsement-triple-record = [
  stateful-environment-record,
  ; endorsed values
  measurement-values-map
]
```

3.2. Extensibility

The base CORIM schema is described using CDDL [RFC8610] that can be extended only at specific allowed points known as "extension points"

The following types of extensions are supported in CoRIM

3.2.1. Map Extensions

Map Extensions provides extensibility support to CoRIM Map structures. CDDL map extensibility enables a CoRIM profile to extend the base CoRIM definition. CDDL map extension points have the form ($$NAME-extension) where "NAME" is the name of the map and '$$' signifies map extensibility. Typically, map extension requires a convention for code point naming that avoids code-point reuse. Well-known code points may be in a registry, such as CoSWID [IANA.concise-software-identifier]. Additionally, a range of code points may be reserved for vendor-specific use such as negative integers.

3.2.2. Data Type Extensions

Data type extensibility has the form ($NAME-type-choice) where "NAME" is the type name and '$' signifies type extensibility.

Schema extensions (Map or Data Type) should be documented to facilitate interoperability. CoRIM profiles are best used to document vendor or industry defined extensions.

4. CoBOM

A Concise Bill of Material (CoBOM) object represents the signal for the Verifier to activate the listed tags. Verifier policy determines whether CoBOMs are required.

When CoBOMs are required, each tag MUST be activated by a CoBOM before being processed. All the tags listed in the CoBOM MUST be activated atomically. If any tag activated by a CoBOM is not available to the Verifier, the entire CoBOM is rejected.

The number of CoBOMs required in a given supply chain ecosystem is dependent on Verifier Owner's Appraisal Policy for Evidence. Corresponding policies are often driven by the complexity and nature of the use case.

If a Verifier Owner has a policy that does not require CoBOM, tags within a CoRIM received by a Verifier are activated immediately and treated valid for appraisal.

There may be cases when Verifier receives CoRIMs from multiple Reference Value providers and Endorsers. In such cases, a supplier (or other authorities, such as integrators) may be designated to issue a single CoBOM to activate all the tags submitted to the Verifier in these CoRIMs.

In a more complex case, there may be multiple authorities that issue CoBOMs at different points in time. An Appraisal Policy for Evidence may dictate how multiple CoBOMs are to be processed within the Verifier.

4.1. Structure

The CDDL specification for the concise-bom-tag map is as follows and this rule and its constraints MUST be followed when creating or validating a CoBOM tag:

```
concise-bom-tag = {
  &(tag-identity: 0) => tag-identity-map
  &(tags-list: 1) => [ + tag-identity-map ],
  &(bom-validity: 2) => validity-map
  * $$concise-bom-tag-extension
}
```

The following describes each member of the concise-bom-tag map.

- tag-identity (index 0): A tag-identity-map containing unique identification information for the CoBOM. Described in Section 3.1.1.

- tags-list (index 1): A list of one or more tag-identity-maps identifying the CoMID and CoSWID tags that constitute the "bill of material", i.e., a complete set of verification-related information. The tags-list behaves like a signaling mechanism from the supply chain (e.g., a product vendor) to a Verifier that activates the tags in tags-list for use in the Evidence appraisal process. The activation is atomic: all tags listed in tags-list MUST be activated or no tags are activated.

- bom-validity (index 2): Specifies the validity period of the CoBOM. Described in Section 1.3.3

- $$concise-bom-tag-extension: This CDDL socket is used to add new information structures to the concise-bom-tag. See Section 8.5. The $$concise-bom-tag-extension extension socket is empty in this specification.

5. CoRIM-based Appraisal of Evidence

The verification procedure is divided into three separate phases:

- *Appraisal Context initialisation
- *Evidence collection
- *Evidence appraisal

At a few well-defined points in the procedure, the Verifier behaviour will depend on the specific CoRIM profile. Each CoRIM profile MUST provide a description of the expected Verifier behavior for each of those well-defined points.

Note that what follows describes a simplified and standard algorithm. Verifiers claiming compliance with this specification MUST exhibit the same externally visible behavior as described here, they are not required to use the same internal data structures. For example, it is expected that the resources used during the initialisation phase can be amortised across multiple appraisals.

5.1. Verifier Abstraction

This document assumes Verifier implementations may differ. To facilitate description of normative Verifier behavior, this document uses abstract representation of Verifier internals.

- *Claim: A piece of information, in the form of a key-value pair.
- *Environment Measurement Tuple (EMT): A structure containing a set of environment Claims that describe a Target Environment and a set of measurement Claims that describe attributes of the Target Environment.
- *reference state: Claims that describe various alternative states of a Target Environment. Reference Values Claims typically describe various possible states due to versioning, manufactruing practices, or supplier configuration options.
- *actual state: Claims that describe a Target Environment instance at a given point in time. Endorsed Values and Evidence typically are Claims about actual state.
- *Group: A set of Evidence, Reference Values, Endorsed Values and Appraisal Policies which are processed together. An Attester may be composed of multiple components, where each component may represent a scope of appraisal.

*Authority: The entity asserting that a claim is true. Typically, a Claim is asserted using a cryptographic key to digitally sign the Claim. A cryptographic key can be a proxy for a human or organizational entity.

*Accepted Claims Set (ACS): A structure that holds EMT Claims that have been vetted following the appraisal process. The ACS describes the actual state of an Attester that has been vetted by Appraisal Policy. The ACS also keeps track of a Claim's authority.

*Appraisal Policy: A description of the conditions that, if met, allow acceptance of Claims. Typically, the entity asserting a Claim should have knowledge, expertise, or context that gives credibility to the assertion. Appraisal Policy resolves which entities are credible and under what conditions.

5.2. Appraisal Context initialisation

The goal of the initialisation phase is to load the CoRIM Appraisal Context with objects such as tags (CoMID, CoSWID, etc.) from CoRIM files, cryptographic validation key material (e.g., raw public keys, root certificates, intermediate CA certificate chains), etc. that will be used in the subsequent Evidence Appraisal phase.

5.2.1. CoRIM Selection

All available CoRIMs are collected. A Verifier may be pre-configured with a large number of CoRIMs describing many types of device. All CoRIMs are loaded at this stage, later stages will select the CoRIMs appropriate to the Evidence Appraisal step.

CoRIMs that are not within their validity period, or that cannot be associated with an authenticated and authorised source MUST be discarded.

CoRIM that are secured by a cryptographic mechanism such as a signature which does not pass validation MUST be discarded.

Other selection criteria MAY be applied.

For example, if the Evidence format is known in advance, CoRIMs using a profile that is not understood by a Verifier can be readily discarded.

The selection process MUST yield at least one usable tag.

5.2.2. CoBOM Extraction

This section is not applicable if the Verifier policy does not require CoBOMs.

All the available Concise Bill Of Material (CoBOMs) tags are then collected from the selected CoRIMs.

CoBOMs which are not within their validity period, or which reference tags not available to the verifier, are discarded.

The Verifier processes all CoBOMs that are valid at the point in time of Evidence Appraisal, and activates all tags referenced therein.

A Verifier may decide to discard some of the available and valid CoBOMs depending on any locally configured authorization policies. (Such policies model the trust relationships between the Verifier Owner and the relevant suppliers, and are out of scope of the present document.)

For example, a composite device (Section 3.3 of [RFC9334]) is likely to be fully described by multiple CoRIMs, each signed by a different supplier. In such case, the Verifier Owner may instruct the Verifier to discard tags activated by supplier CoBOMs that are not activated by the trusted integrator.

After the Verifier has processed all CoBOMs it MUST discard any tags which have not been activated by a CoBOM.

5.2.3. Tags Identification and Validation

The Verifier chooses tags -- including Concise Module ID Tags (CoMID, Section 3), Concise Software ID Tags (CoSWID, [I-D.ietf-sacm-coswid]), and/or Concise Trust Anchor Stores (CoTS, [I-D.ietf-rats-concise-ta-stores]) -- from the selected CoRIMs.

The Verifier MUST discard all tags which are not syntactically and semantically valid. In particular, any cross-referenced triples (e.g., CoMID-CoSWID linking triples) MUST be successfully resolved.

5.2.4. Appraisal Context Construction

All of the validated and potentially useful tags are loaded into the Appraisal Context.

This concludes the initialisation phase.

5.3. Evidence Collection

In the Evidence collection phase the Verifier communicates with attesters to collect Evidence.

The first part of the Evidence collection phase does not perform any cryptographic validation. This allows Verifiers to use untrusted code for their initial Evidence collection.

The results of the evidence collection are protocol specific data and transcripts which are used as input to appraisal by the Verifier.

5.3.1. Cryptographic validation of Evidence

If the authenticity of Evidence is secured by a cryptographic mechanism such as a signature, the first step in the Evidence Appraisal is to perform cryptographic validation of the Evidence.

The exact cryptographic signature validation mechanics depend on the specific Evidence collection protocol.

For example: In DICE, a proof of liveness is performed on the final key in the certificate chain. If this passes then a suitable certification path anchored on a trusted root certificate is looked up -- e.g., based on linking information obtained from the DeviceID certificate (see Section 9.2.1 of [DICE.Layer]) -- in the Appraisal Context. If found, then usual X.509 certificate validation is performed. In PSA, the verification public key is looked up in the appraisal context using the ueid claim found in the PSA claims-set (see Section 4.2.1 of [I-D.tschofenig-rats-psa-token]). If found, COSE Sign1 verification is performed accordingly.

Independent of the specific integrity protection method used, the integrity of Evidence MUST be successfully verified.

A CoRIM profile MUST describe:

-How cryptographic verification key material is represented (e.g., using Attestation Keys triples, or CoTS tags)

-How key material is associated with the Attesting Environment

-How the Attesting Environment is identified in Evidence

5.3.2. The Accepted Claims Set

At the end of the Evidence collection process Evidence has been converted into a format suitable for appraisal. To this end, this document describes an accepted-claims-set format and the algorithms used to compare it against CoMID Reference Values.

```
accepted-claims-set = {
  &(state-triples: 0) => [ + endorsed-triple-record ]
  ? &(identity-triples: 1) => [ + identity-triple-record ]
  ? &(coswid-triples: 2) => [ + ev-coswid-triple-record ]
  * $$accepted-claims-set-extension
}
```

Verifiers are not required to use this as their internal state, for the purposes of this document a sample Verifier is discussed which uses this format.

The Accepted Claims Set (ACS) contains the actual state of Target Environments (TEs). The state-triples field contains Evidence (from Attesters) and Endorsements (e.g. from endorsed-triple-record).

CoMID Reference Values will be matched against the Accepted Claims Set, as per the appraisal policy of the Verifier. This document describes an example evidence structure which can be easily matched against these Reference Values.

Each entry within state-triples uses the syntax of endorsed-triple-record. When an endorsed-triple-record appears within state-triples it indicates that the authority named by measurement-map/authorized-by asserts that the actual state of one or more Claims within the Target Environment, as identified by environment-map, have the measurement values in measurement-map/mval.

In authorized-by, authority is represented by cryptographic keys. Authority is asserted by digitally signing a Claim using the key. Hence, Claims are added to the ACS under the authority of a key.

Each Claim is encoded as an Environment Measurement Tuple (a contraction of environment-map, measurement-map tuple). The environment-map and a key within measurement-values-map encode the name of the Claim. The value matching that key within measurement-values-map is the actual state of the Claim.

This specification does not assign special meanings to any Claim name, it only specifies rules for determining when two Claim names are the same.

If two Claims have the same environment-map encoding then this does not trigger special encoding in the Verifier. The Verifier follows instructions in the CoRIM file which tell it how claims are related.

If Evidence or Endorsements from different sources has the same environment-map and authorized-by then the measurement-values-maps are merged.

The ACS must maintain the authority information for each EMT. There can be multiple entries in state-triples which have the same environment-map and a different authorized-by field (see Section 5.3.2.2).

If the merged measurement-value-map contains duplicate codepoints and the measurement values are equivalent, then duplicate claims SHOULD

be omitted. Equivalence typically means values MUST be binary identical.

If the merged measurement-value-map contains duplicate codepoints and the measurement values are not equivalent then the verifier SHALL report an error and stop validation processing.

5.3.2.1. Accepted Claims Set Initialization

The Accepted Claims Set is initialized by copying Evidence claims describing authenticated Attester's Target Environments into the Verifier's Accepted Claims Set.

Evidence formats may require format translation before being added to the Accepted Claims Set. If format translation is required, a CoRIM profile, see Section 2.1.4, defines an Evidence translation function.

Section 5.5 provides information on how DICE and SPDM Evidence is reformatted into CoMID schema compliant expressions before being added to the Accepted Claims Set.

5.3.2.2. The authorized-by field in Accepted Claims Set

The authorized-by field in an Accepted Claims Set entry indicates the entity whose authority backs the claim.

An entity is authoritative when it makes Claims that are inside its area of competence. The Verifier keeps track of the authorities that assert Claims so that it can filter out claims from entities that do not satisfy appraisal policies.

When adding an Evidence Claim to the Accepted Claims Set, the Verifier SHALL set the authorized-by field in that Claim to the trusted authority keys at the head of each key chain which signed that Evidence. This key is often the subject of a self-signed certificate. The Verifier has already verified the certificate chain (see Section 5.3.1).

If multiple authorities approve the same Claim, for example if multiple key chains are available, then the authorized-by field SHALL be set to include the trusted authority keys used by each of those authorities.

When adding Endorsement Claims to the Accepted Claims Set that resulted from CoRIM processing (see Section 5.4.3) the Verifier SHALL set the authorized-by field in that Evidence to the trusted authority key that is at the head of the key chain that signed the CoRIM.

When searching the Accepted Claims Set for an entry which matches a Reference Value containing an authorized-by field, the Verifier SHALL

ignore ACS entries if none of the keys present in the Reference Value authorized-by field are also present in the ACS authorized-by field.

The Verifier SHOULD set the authorized-by field in Accepted Claims Set entries to a format which contains only a key, for example the tagged-cose-key-type format. Using a common format makes it easier to compare the field.

5.4. Accepted Claims Set extension using CoMID tags or triples

In the Accepted Claims Set extension phase, a CoRIM Appraisal Context and an Evidence Appraisal Policy are used by the Verifier to find CoMID tags or triples which match the Attester. Tags/triples which match are accepted, and the Accepted Claims Set is extended using Endorsements etc. from the accepted tags.

5.4.1. Comparing and processing CoMID tags or triples

5.4.2. Matching Evidence against Reference Values

An Endorser may use CoMID tags to publish Conditional Endorsements, which are added to the Accepted Claims Set only if specified conditions apply. This section describes the process performed by the Verifier to determine which Conditional Endorsements from the candidate CoMIDs should be added to the Accepted Claims Set.

The verifier checks whether Conditional Endorsements are applicable by comparing Evidence in the Accepted Claims Set against Reference Values from the CoMID. These Reference Values may be provided as Reference Value Triples or may be combined with the Endorsements, for example as the Conditional Endorsement Series Triple.

The following subsections describe how the CoRIM tells the verifier which Reference Values and Endorsed Values are grouped together (Section 5.4.2.1) and how the verifier matches a Reference Value against the Accepted Claims Set (Section 5.4.2.2).

5.4.2.1. Grouping Reference Values and Endorsements

> This paragraph will be replaced by a description of how the CoRIM tells the verifier which Reference Values and Endorsed Values are grouped together.

5.4.2.2. Matching all Reference Values in a group against the Accepted Claims Set

If all Reference Values in a group match entries in the Accepted Claims Set then all Endorsements in the group are added to the Accepted Claims Set (see Section 5.4.3). Section 5.4.2.3 describes how one Reference Value is matched against the Accepted Claims Set.

If any Reference Value in a group does not match the Accepted Claims Set then all Endorsements in the group are silently ignored.

Each group is processed independently of other groups. If a group fails to match the Accepted Claims Set then this does not affect the processing of other groups.

5.4.2.3. Matching a Reference Value against the Accepted Claims Set

This section describes how a Reference Value is matched against Evidence in the Accepted Claims Set. If any part of the processing indicates that the Reference Value does not match then the remaining steps in this section are skipped for that group.

A Reference Value consists of an environment-map plus a measurement-map. In the reference-triple-record these are encoded together. In other triples multiple Reference Values are represented more compactly by letting one environment-map apply to multiple measurement-maps.

The Verifier first looks for entries in the Accepted Claims Set with the same environment-map as the Reference Value. These are the candidate claims. If there are no candidate claims then the Reference Value does not match.

A Verifier SHALL compare two environment-maps using a binary comparison of the CBOR encoded objects.

A Verifier SHOULD convert environment-map into a form which meets CBOR Core Deterministic Encoding Requirements [STD94] before performing the binary comparison.

If the Reference Value contains an authorized-by field then the Verifier SHALL modify the candidate claims set to remove Claims whose authorized-by field does not contain one of the keys listed in the Reference Value authorized-by field (see Section 5.3.2.2 for more details). If all candidate claim entries are discarded by this step then the Reference Value does not match.

The Verifier SHALL iterate over the codepoints which are present in the measurement-values-map field within the Reference Value measurement-values-map. The Reference Value entry is compared against each of the candidate claims. If none of the candidate claims matches the Reference Value entry then the Reference Value does not match.

The algorithm used to match the measurement-values-map entries is described below. The comparison performed depends on the type of field being compared.

If the Reference Value measurement-values-map value is tagged with a CBOR tag [STD94] then the Verifier MUST use the comparison algorithm associated with that tag.

If the Reference Value is not tagged then the Verifier MUST use the comparison algorithm associated with the measurement-values-map codepoint for the entry.

This specification defines the matching algorithm for some CBOR tagged reference values, which is described in sub-sections below.

A CoRIM profile may define additional tags and their matching algorithms.

If the Verifier does not recognize the Reference Value CBOR tag value then the Reference Value does not match.

If the Reference Value is not tagged and the measurement-value-map key is a value with handling described in the sub-sections below, then the algorithm appropriate to that key is used to match the entries.

If the Reference Value is not tagged, and the measurement-values-map key is not a value described below, then the entries are compared using binary comparison of their CBOR encoded values. If the values are not binary identical then the Reference Value does not match.

Note that while specifications may extend the matching semantics using CBOR tags, there is no way to extend the matching semantics of keys. Any new keys requiring non-default comparison must add a CBOR tag to the Reference Value describing the desired behaviour.

If all checks above have been performed successfully then the Reference Value matches.

5.4.2.3.1. Comparison for svn entries

The value stored under measurement-values-map key 1 is an SVN, which must have type UINT.

If the Reference value for measurement-values-map key 1 is an untagged UINT or a UINT tagged with #6.552 then an equality comparison is performed. If the value of the SVN in Accepted Claims Set is not equal to the value in the Reference Value then the Reference Value does not match.

If the Reference value for measurement-values-map key 1 is a UINT tagged with #6.553 then a minimum comparison is performed. If the value of the SVN in Accepted Claims Set less than the value in the Reference Value then the Reference Value does not match.

5.4.2.3.2. Comparison for digests entries

The value stored under measurement-values-map key 2, or a value tagged with #6.TBD is a digest entry. It contains one or more digests, each measuring the same object. A Reference Value may contain multiple digests, each with a different algorithm acceptable to the Reference Value provider. If the digest in Evidence contains a single value with an algorithm and value matching one of the algorithms and values in the Reference Value then it matches.

To prevent downgrade attacks, if there are multiple algorithms which are in both the Evidence and Reference Value then the digests calculated using all shared algorithms must match.

If the CBOR encoding of the digests entry in the Reference Value or the Accepted Claim Set value with the same key is incorrect (for example if fields are missing or the wrong type) then the Reference Value does not match.

The Verifier MUST iterate over the Reference Value digests array, locating hash algorithm identifiers that are present in the Reference Value and in the Accepted Claims Set entry.

If the hash algorithm identifier which is present in the Reference Value differs from the hash algorithm identifier in the Accepted Claims Set entry then the Reference Value does not match.

If a hash algorithm identifier is present in both the Reference Value and the Accepted Claims Set, but the value of the hash is not binary identical between the Reference Value and the Accepted Claims Set entry then the Reference Value does not match.

5.4.2.3.3. Comparison for raw-value entries

> I think this comparison method only works if the entry is at key 4 (because there needs to be a mask at key 5). Should we have a Reference Value of this which stores [expect-raw-value raw-value-mask] in an array?

5.4.2.3.4. Comparison for cryptokeys entries

The value stored under measurement-values-map key 12 is an array of $crypto-key-type-choice entries. $crypto-key-type-choice entries are CBOR tagged values. The array contains one or more entries in sequence.

The CBOR tag of the first entry of the Reference Value cryptokeys array is compared with the CBOR tag of the first entry of the Accepted Claims Set cryptokeys value. If the CBOR tags match, then the bytes following the CBOR tag from the Reference Value entry are

compared with the bytes following the CBOR tag from the Accepted Claims Set entry. If the byte strings match, and there is another array entry, then the next entry from the Reference Values array is likewise compared with the next entry of the Accepted Claims Set array. If all entries of the Reference Values array match a corresponding entry in the Accepted Claims Set array, then the cryptokeys Reference Value matches. Otherwise, cryptokeys does not match.

5.4.2.3.5. Handling of new tags

A profile may specify handling for new CBOR tagged Reference Values. The profile must specify how to compare the CBOR tagged Reference Value against the Accepted Claims Set.

Note that the verifier may compare Reference Values in any order, so the comparison should not be stateful.

5.4.3. Adding CoMID Endorsed Values to the Accepted Claims Set

5.5. Adding DICE/SPDM Evidence to the Accepted Claims Set

This section defines how Evidence from DICE [DICE.AA] and/or SPDM [SPDM] is transformed into a format where it can be added to an accepted claims set. A Verifier supporting DICE/SPDM format Evidence should implement this section.

5.5.1. Transforming SPDM Evidence to a format usable for matching

The TCG DICE Concise Evidence Binding for SPDM specification [CE.SPDM] describes the process by which measurements in an SPDM Measurement Block are converted to Evidence suitable for matching using the rules below. The SPDM measurements are converted to concise-evidence which has a format that is similar to CoRIM triples-map (their semantics follows the matching rules described above).

5.5.2. Transforming DICE Evidence to a format usable for matching

DICE Evidence appears in certificates in the TcbInfo or MultiTcbInfo extension. Each TcbInfo, and each entry in the MultiTcbInfo, is converted to an endorsed-triple-record using the rules in this section. In a MultiTcbInfo each entry in the sequence is treated as independent and translated into a separate Evidence object.

The Verifier SHALL translate each field in the TcbInfo into a field in the created endorsed-triple-record

* The TcbInfo type field SHALL be copied to the field named environment-map / class / class-id and tagged with tag #6.111

* The TcbInfo vendor field SHALL be copied to the field named environment-map / class / vendor

* The TcbInfo model field SHALL be copied to the field named environment-map / class / model

* The TcbInfo layer field SHALL be copied to the field named environment-map / class / layer

* The TcbInfo index field SHALL be copied to the field named environment-map / class / index

* The TcbInfo version field SHALL be translated to the field named measurement-map / mval / version / version

* The TcbInfo svn field SHALL be copied to the field named measurement-map / mval / svn

* The TcbInfo fwids field SHALL be translated to the field named measurement-map / mval / digests

  - Each digest within fwids is translated to a CoMID digest object, with an appropriate algorithm identifier

* The TcbInfo flags field SHALL be translated to the field named measurement-map / mval / flags

  - Each flag is translated independently

* The TcbInfo vendorInfo SHALL shall be copied to the field named measurement-map / mval / raw-value

If there are multiple endorsed-triple-records with the same environment-map then they MUST be merged into a single entry. If the measurement-values-map fields in Evidence triples have conflicting values then the Verifier MUST fail validation.

6. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalogue of available implementations or their

features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as Evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

6.1. Veraison

* Organization responsible for the implementation: Veraison Project, Linux Foundation

* Implementation's web page: https://github.com/veraison/corim/README.md

* Brief general description: The corim/corim and corim/comid packages provide a golang API for low-level manipulation of Concise Reference Integrity Manifest (CoRIM) and Concise Module Identifier (CoMID) tags respectively. The corim/cocli package uses the API above (as well as the API from the veraison/swid package) to provide a user command line interface for working with CoRIM, CoMID and CoSWID. Specifically, it allows creating, signing, verifying, displaying, uploading, and more. See https://github.com/cocli/README.md for further details.

* Implementation's level of maturity: alpha.

* Coverage: the whole protocol is implemented, including PSA-specific extensions [I-D.fdb-rats-psa-endorsements].

* Version compatibility: Version -02 of the draft

* Licensing: Apache 2.0 https://github.com/veraison/corim/blob/main/LICENSE

* Implementation experience: n/a

* Contact information: https://veraison.zulipchat.com

* Last updated: https://github.com/veraison/corim/commits/main

7. Security and Privacy Considerations

Content missing. Tracked at: https://github.com/ietf-rats-wg/draft-ietf-rats-corim/issues/11

8. IANA Considerations

8.1. New COSE Header Parameters

Content missing. Tracked at: https://github.com/ietf-rats-wg/draft-ietf-rats-corim/issues/12

8.2. New CBOR Tags

IANA is requested to allocate the following tags in the "CBOR Tags" registry [IANA.cbor-tags], preferably with the specific CBOR tag value requested:

Tag	Data Item	Semantics	Reference
500	tag	A tagged-concise-rim-type-choice, see Section 2.1.2	RFCthis
501	map	A tagged-corim-map, see Section 2.1	RFCthis
502	tag	A tagged-signed-corim, see Section 2.2	RFCthis
503-504	any	Earmarked for CoRIM	RFCthis
505	bytes	A tagged-concise-swid-tag, see Section 2.1.2	RFCthis
506	bytes	A tagged-concise-mid-tag, see Section 2.1.2	RFCthis
507	any	Earmarked for CoRIM	RFCthis
508	bytes	A tagged-concise-bom-tag, see Section 2.1.2	RFCthis
509-549	any	Earmarked for CoRIM	RFCthis
550	bytes .size 33	tagged-ueid-type, see Section 1.3.5	RFCthis
551	int	tagged-int-type, see Section 1.3.7	RFCthis
552	uint	tagged-svn, see Section 3.1.4.1.5.4	RFCthis
553	uint	tagged-min-svn, see Section 3.1.4.1.5.4	RFCthis
554	text	tagged-pkix-base64-key-type, see Section 3.1.4.1.6	RFCthis
555	text	tagged-pkix-base64-cert-type, see Section 3.1.4.1.6	RFCthis
556	text	tagged-pkix-base64-cert-path-type, see Section 3.1.4.1.6	RFCthis
557	[int/text, bytes]	tagged-thumbprint-type, see Section 1.3.8	RFCthis
558	COSE_Key/ COSE_KeySet	tagged-cose-key-type, see Section 3.1.4.1.6	RFCthis
559	digest		RFCthis

Tag	Data Item	Semantics	Reference
		tagged-cert-thumbprint-type, see Section 3.1.4.1.6	
560	bytes	tagged-bytes, see Section 3.1.4.1.5.6	RFCthis
561	digest	tagged-cert-path-thumbprint-type, see Section 3.1.4.1.6	RFCthis
562-599	any	Earmarked for CoRIM	RFCthis

Table 2

Tags designated as "Earmarked for CoRIM" can be reassigned by IANA based on advice from the designated expert for the CBOR Tags registry.

8.3. New CoRIM Registries

Content missing. Tracked at: https://github.com/ietf-rats-wg/draft-ietf-rats-corim/issues/14

8.4. New CoMID Registries

Content missing. Tracked at: https://github.com/ietf-rats-wg/draft-ietf-rats-corim/issues/15

8.5. New CoBOM Registries

Content missing. Tracked at: https://github.com/ietf-rats-wg/draft-ietf-rats-corim/issues/45

8.6. New Media Types

IANA is requested to add the following media types to the "Media Types" registry [IANA.media-types].

Name	Template	Reference
corim-signed+cbor	application/corim-signed+cbor	RFCthis, Section 8.6.1
corim-unsigned+cbor	application/corim-unsigned+cbor	RFCthis, Section 8.6.2

Table 3: New Media Types

8.6.1. corim-signed+cbor

Type name: application
Subtype name: corim-signed+cbor
Required parameters: n/a
Optional parameters: "profile" (CoRIM profile in string format. OIDs MUST use the dotted-decimal notation.)

Encoding considerations: binary
Security considerations: Section 7 of RFCthis
Interoperability considerations: n/a
Published specification: RFCthis
Applications that use this media type: Attestation Verifiers, Endorsers and Reference-Value providers that need to transfer COSE Sign1 wrapped CoRIM payloads over HTTP(S), CoAP(S), and other transports.
Fragment identifier considerations: n/a
Magic number(s): D9 01 F6 D2, D9 01 F4 D9 01 F6 D2
File extension(s): n/a
Macintosh file type code(s): n/a
Person & email address to contact for further information: RATS WG mailing list (rats@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author/Change controller: IETF
Provisional registration? Maybe

8.6.2. corim-unsigned+cbor

Type name: application
Subtype name: corim-unsigned+cbor
Required parameters: n/a
Optional parameters: "profile" (CoRIM profile in string format. OIDs MUST use the dotted-decimal notation.)
Encoding considerations: binary
Security considerations: Section 7 of RFCthis
Interoperability considerations: n/a
Published specification: RFCthis
Applications that use this media type: Attestation Verifiers, Endorsers and Reference-Value providers that need to transfer unprotected CoRIM payloads over HTTP(S), CoAP(S), and other transports.
Fragment identifier considerations: n/a
Magic number(s): D9 01 F5, D9 01 F4 D9 01 F5
File extension(s): n/a
Macintosh file type code(s): n/a
Person & email address to contact for further information: RATS WG mailing list (rats@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author/Change controller: IETF
Provisional registration? Maybe

8.7. CoAP Content-Formats Registration

IANA is requested to register the two following Content-Format numbers in the "CoAP Content-Formats" sub-registry, within the

"Constrained RESTful Environments (CoRE) Parameters" Registry [IANA.core-parameters]:

Content-Type	Content Coding	ID	Reference
application/corim-signed+cbor	-	TBD1	RFCthis
application/corim-unsigned+cbor	-	TBD2	RFCthis

Table 4: New Content-Formats

9. References

9.1. Normative References

[I-D.ietf-rats-concise-ta-stores] Wallace, C., Housley, R., Fossati, T., and Y. Deshpande, "Concise TA Stores (CoTS)", Work in Progress, Internet-Draft, draft-ietf-rats-concise-ta-stores-01, 5 June 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-concise-ta-stores-01>.

[I-D.ietf-rats-eat] Lundblade, L., Mandyam, G., O'Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-22, 14 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-22>.

[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-tags] IANA, "Concise Binary Object Representation (CBOR) Tags", <https://www.iana.org/assignments/cbor-tags>.

[IANA.core-parameters] IANA, "Constrained RESTful Environments (CoRE) Parameters", <https://www.iana.org/assignments/core-parameters>.

[IANA.language-subtag-registry] IANA, "Language Subtag Registry", <https://www.iana.org/assignments/language-subtag-registry>.

[IANA.media-types] IANA, "Media Types", <https://www.iana.org/assignments/media-types>.

[IANA.named-information] IANA, "Named Information", <https://www.iana.org/assignments/named-information>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://doi.org/10.17487/RFC2119>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://doi.org/10.17487/RFC4122>.

[RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <https://doi.org/10.17487/RFC7468>.

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

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://doi.org/10.17487/RFC8610>.

[RFC9090] Bormann, C., "Concise Binary Object Representation (CBOR) Tags for Object Identifiers", RFC 9090, DOI 10.17487/RFC9090, July 2021, <https://doi.org/10.17487/RFC9090>.

[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://doi.org/10.17487/RFC9334>.

[STD66] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://doi.org/10.17487/RFC3986>.

[STD94] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://doi.org/10.17487/RFC8949>.

[STD96] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://doi.org/10.17487/RFC9052>.

[X.690] International Telecommunications Union, "Information technology -- ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, August 2015, <https://www.itu.int/rec/T-REC-X.690>.

9.2. Informative References

[CE.SPDM] Trusted Computing Group, "TCG DICE Concise Evidence Binding for SPDM", Version 1.00, Revision 0.53, public review , June 2023, <https://trustedcomputinggroup.org/wp-content/uploads/TCG-DICE-Concise-Evidence-Binding-for-SPDM-Version-1.0-Revision-53_1August2023.pdf>.

[DICE.AA] Trusted Computing Group, "DICE Attestation Architecture", Version 1.1, Revision 0.17, public review , May 2023, <https://trustedcomputinggroup.org/wp-content/uploads/DICE-Attestation-Architecture-Version-1.1-Revision-17_1August2023.pdf>.

[DICE.Layer] Trusted Computing Group, "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.fdb-rats-psa-endorsements] Fossati, T., Deshpande, Y., and H. Birkholz, "Arm's Platform Security Architecture (PSA) Attestation Verifier Endorsements", Work in Progress, Internet-Draft, draft-fdb-rats-psa-endorsements-03, 10

September 2023, <https://datatracker.ietf.org/doc/html/draft-fdb-rats-psa-endorsements-03>.

[I-D.tschofenig-rats-psa-token] Tschofenig, H., Frost, S., Brossard, M., Shaw, A. L., and T. Fossati, "Arm's Platform Security Architecture (PSA) Attestation Token", Work in Progress, Internet-Draft, draft-tschofenig-rats-psa-token-14, 23 October 2023, <https://datatracker.ietf.org/doc/html/draft-tschofenig-rats-psa-token-14>.

[IANA.concise-software-identifier] "*** BROKEN REFERENCE ***".

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://doi.org/10.17487/RFC7942>.

[SPDM] Distributed Management Task Force, "Security Protocol and Data Model (SPDM)", Version 1.3.0 , May 2023, <https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.3.0.pdf>.

Appendix A. Full CoRIM CDDL

```
corim = tagged-concise-rim-type-choice

$concise-rim-type-choice /= tagged-corim-map
$concise-rim-type-choice /= tagged-signed-corim

concise-bom-tag = {
  &(tag-identity: 0) => tag-identity-map
  &(tags-list: 1) => [ + tag-identity-map ],
  &(bom-validity: 2) => validity-map
  * $$concise-bom-tag-extension
}

$concise-tag-type-choice /= tagged-concise-swid-tag
$concise-tag-type-choice /= tagged-concise-mid-tag
$concise-tag-type-choice /= tagged-concise-bom-tag

corim-entity-map =
  entity-map<$corim-role-type-choice, $$corim-entity-map-extension>

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

corim-locator-map = {
  &(href: 0) => uri
  ? &(thumbprint: 1) => digest
}

corim-map = {
  &(id: 0) => $corim-id-type-choice
  &(tags: 1) => [ + $concise-tag-type-choice ]
  ? &(dependent-rims: 2) => [ + corim-locator-map ]
  ? &(profile: 3) => $profile-type-choice
  ? &(rim-validity: 4) => validity-map
  ? &(entities: 5) => [ + corim-entity-map ]
  * $$corim-map-extension
}

corim-meta-map = {
  &(signer: 0) => corim-signer-map
  ? &(signature-validity: 1) => validity-map
}

$corim-role-type-choice /= &(manifest-creator: 1)

corim-signer-map = {
  &(signer-name: 0) => $entity-name-type-choice
  ? &(signer-uri: 1) => uri
  * $$corim-signer-map-extension
}
```

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

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

protected-corim-header-map = {
  &(alg-id: 1) => int
  &(content-type: 3) => "application/corim-unsigned+cbor"
  &(issuer-key-id: 4) => bstr
  &(corim-meta: 8) => bstr .cbor corim-meta-map
  * cose-label => cose-value
}

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

tagged-corim-map = #6.501(corim-map)


tagged-concise-rim-type-choice = #6.500($concise-rim-type-choice)

tagged-signed-corim = #6.502(signed-corim)

tagged-concise-swid-tag = #6.505(bytes .cbor concise-swid-tag)

tagged-concise-mid-tag = #6.506(bytes .cbor concise-mid-tag)

tagged-concise-bom-tag = #6.508(bytes .cbor concise-bom-tag)

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

validity-map = {
  ? &(not-before: 0) => time
  &(not-after: 1) => time
}

concise-mid-tag = {
  ? &(language: 0) => text
  &(tag-identity: 1) => tag-identity-map
  ? &(entities: 2) => [ + comid-entity-map ]
  ? &(linked-tags: 3) => [ + linked-tag-map ]
  &(triples: 4) => triples-map
  * $$concise-mid-tag-extension
}
```

```
accepted-claims-set = {
  &(state-triples: 0) => [ + endorsed-triple-record ]
  ? &(identity-triples: 1) => [ + identity-triple-record ]
  ? &(coswid-triples: 2) => [ + ev-coswid-triple-record ]
  * $$accepted-claims-set-extension
}

attest-key-triple-record = [
  environment-map
  [ + $crypto-key-type-choice ]
]

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

class-map = non-empty<{
  ? &(class-id: 0) => $class-id-type-choice
  ? &(vendor: 1) => tstr
  ? &(model: 2) => tstr
  ? &(layer: 3) => uint
  ? &(index: 4) => uint
}>

comid-entity-map =
  entity-map<$comid-role-type-choice, $$comid-entity-map-extension>

$comid-role-type-choice /= &(tag-creator: 0)
$comid-role-type-choice /= &(creator: 1)
$comid-role-type-choice /= &(maintainer: 2)

conditional-endorsement-series-triple-record = [
  stateful-environment-record
  ; order matters: the first matching record wins and halts matching
  [ + conditional-series-record ]
]

conditional-endorsement-triple-record = [
  stateful-environment-record,
  ; endorsed values
  measurement-values-map
]

conditional-series-record = [
  ; reference values to be matched against evidence
  refv: measurement-values-map
  ; endorsed values that apply in case revf matches
  endv: measurement-values-map
]
```

```
COSE_KeySet = [ + COSE_Key ]

COSE_Key = {
    1 => tstr / int
    ? 2 => bstr
    ? 3 => tstr / int
    ? 4 => [+ (tstr / int) ]
    ? 5 => bstr
    * cose-label => cose-value
}

cose-label = int / tstr
cose-value = any

coswid-triple-record = [
  environment-map
  [ + concise-swid-tag-id ]
]

concise-swid-tag-id = text / bstr .size 16

$crypto-key-type-choice /= tagged-pkix-base64-key-type
$crypto-key-type-choice /= tagged-pkix-base64-cert-type
$crypto-key-type-choice /= tagged-pkix-base64-cert-path-type
$crypto-key-type-choice /= tagged-cose-key-type
$crypto-key-type-choice /= tagged-thumbprint-type
$crypto-key-type-choice /= tagged-cert-thumbprint-type
$crypto-key-type-choice /= tagged-cert-path-thumbprint-type

tagged-pkix-base64-key-type = #6.554(tstr)
tagged-pkix-base64-cert-type = #6.555(tstr)
tagged-pkix-base64-cert-path-type = #6.556(tstr)
tagged-thumbprint-type = #6.557(digest)
tagged-cose-key-type = #6.558(COSE_KeySet / COSE_Key)
tagged-cert-thumbprint-type = #6.559(digest)
tagged-cert-path-thumbprint-type = #6.561(digest)

domain-dependency-triple-record = [
  $domain-type-choice
  [ + $domain-type-choice ]
]

domain-membership-triple-record = [
  $domain-type-choice
  [ + environment-map ]
]

$domain-type-choice /= uint
$domain-type-choice /= text
```

```
$domain-type-choice /= tagged-uuid-type
$domain-type-choice /= tagged-oid-type

endorsed-triple-record = [
  environment-map
  measurement-map
]

entity-map<role-type-choice, extension-socket> = {
  &(entity-name: 0) => $entity-name-type-choice
  ? &(reg-id: 1) => uri
  &(role: 2) => [ + role-type-choice ]
  * extension-socket
}

$entity-name-type-choice /= text

environment-map = non-empty<{
  ? &(class: 0) => class-map
  ? &(instance: 1) => $instance-id-type-choice
  ? &(group: 2) => $group-id-type-choice
}>

flags-map = {
  ? &(is-configured: 0) => bool
  ? &(is-secure: 1) => bool
  ? &(is-recovery: 2) => bool
  ? &(is-debug: 3) => bool
  ? &(is-replay-protected: 4) => bool
  ? &(is-integrity-protected: 5) => bool
  ? &(is-runtime-meas: 6) => bool
  ? &(is-immutable: 7) => bool
  ? &(is-tcb: 8) => bool
  ? &(is-confidentiality-protected: 9) => bool
  * $$flags-map-extension
}

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

identity-triple-record = [
  environment-map
  [ + $crypto-key-type-choice ]
]

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

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

```
ip6-addr-type = bytes .size 16

linked-tag-map = {
  &(linked-tag-id: 0) => $tag-id-type-choice
  &(tag-rel: 1) => $tag-rel-type-choice
}

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

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

measurement-map = {
  ? &(mkey: 0) => $measured-element-type-choice
  &(mval: 1) => measurement-values-map
  ? &(authorized-by: 2) => [ + $crypto-key-type-choice ]
}

measurement-values-map = non-empty<{
  ? &(version: 0) => version-map
  ? &(svn: 1) => svn-type-choice
  ? &(digests: 2) => [ + digest ]
  ? &(flags: 3) => flags-map
  ? (
      &(raw-value: 4) => $raw-value-type-choice,
      ? &(raw-value-mask: 5) => raw-value-mask-type
    )
  ? &(mac-addr: 6) => mac-addr-type-choice
  ? &(ip-addr: 7) =>  ip-addr-type-choice
  ? &(serial-number: 8) => text
  ? &(ueid: 9) => ueid-type
  ? &(uuid: 10) => uuid-type
  ? &(name: 11) => text
  ? &(cryptokeys: 12) => [ + $crypto-key-type-choice ]
  * $$measurement-values-map-extension
}>

non-empty<M> = (M) .and ({ + any => any })

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

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

raw-value-mask-type = bytes
```

```
reference-triple-record = [
  environment-map
  measurement-map
]

stateful-environment-record = [
  environment-map,
  measurement-map
]

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

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

tag-identity-map = {
  &(tag-id: 0) => $tag-id-type-choice
  ? &(tag-version: 1) => tag-version-type
}

$tag-rel-type-choice /= &(supplements: 0)
$tag-rel-type-choice /= &(replaces: 1)

tag-version-type = uint .default 0

tagged-int-type = #6.551(int)

triples-map = non-empty<{
  ? &(reference-triples: 0) =>
    [ + reference-triple-record ]
  ? &(endorsed-triples: 1) =>
    [ + endorsed-triple-record ]
  ? &(identity-triples: 2) =>
    [ + identity-triple-record ]
  ? &(attest-key-triples: 3) =>
    [ + attest-key-triple-record ]
  ? &(dependency-triples: 4) =>
    [ + domain-dependency-triple-record ]
  ? &(membership-triples: 5) =>
    [ + domain-membership-triple-record ]
  ? &(coswid-triples: 6) =>
    [ + coswid-triple-record ]
  ? &(conditional-endorsement-series-triples: 8) =>
    [ + conditional-endorsement-series-triple-record ]
  ? &(conditional-endorsement-triples: 9) =>
```

```
    [ + conditional-endorsement-triple-record ]
  * $$triples-map-extension
}>

ueid-type = bytes .size 33
tagged-ueid-type = #6.550(ueid-type)

uuid-type = bytes .size 16
tagged-uuid-type = #6.37(uuid-type)

version-map = {
  &(version: 0) => text
  ? &(version-scheme: 1) => $version-scheme
}

digest = [
  alg: (int / text),
  val: bytes
]

concise-swid-tag = {
  tag-id => text / bstr .size 16,
  tag-version => integer,
  ? corpus => bool,
  ? patch => bool,
  ? supplemental => bool,
  software-name => text,
  ? software-version => text,
  ? version-scheme => $version-scheme,
  ? media => text,
  ? software-meta => one-or-more<software-meta-entry>,
  entity => one-or-more<entity-entry>,
  ? link => one-or-more<link-entry>,
  ? payload-or-evidence,
  * $$coswid-extension,
  global-attributes,
}

payload-or-evidence //= ( payload => payload-entry )
payload-or-evidence //= ( evidence => evidence-entry )

any-uri = uri
label = text / int

$version-scheme /= multipartnumeric
$version-scheme /= multipartnumeric-suffix
$version-scheme /= alphanumeric
$version-scheme /= decimal
$version-scheme /= semver
$version-scheme /= int / text
```

```
any-attribute = (
  label => one-or-more<text> / one-or-more<int>
)

one-or-more<T> = T / [ 2* T ]

global-attributes = (
  ? lang => text,
  * any-attribute,
)

hash-entry = [
  hash-alg-id: int,
  hash-value: bytes,
]

entity-entry = {
  entity-name => text,
  ? reg-id => any-uri,
  role => one-or-more<$role>,
  ? thumbprint => hash-entry,
  * $$entity-extension,
  global-attributes,
}

$role /= tag-creator
$role /= software-creator
$role /= aggregator
$role /= distributor
$role /= licensor
$role /= maintainer
$role /= int / text

link-entry = {
  ? artifact => text,
  href => any-uri,
  ? media => text,
  ? ownership => $ownership,
  rel => $rel,
  ? media-type => text,
  ? use => $use,
  * $$link-extension,
  global-attributes,
}

$ownership /= shared
$ownership /= private
$ownership /= abandon
$ownership /= int / text
```

```
$rel /= ancestor
$rel /= component
$rel /= feature
$rel /= installationmedia
$rel /= packageinstaller
$rel /= parent
$rel /= patches
$rel /= requires
$rel /= see-also
$rel /= supersedes
$rel /= supplemental
$rel /= -256..64436 / text

$use /= optional
$use /= required
$use /= recommended
$use /= int / text

software-meta-entry = {
  ? activation-status => text,
  ? channel-type => text,
  ? colloquial-version => text,
  ? description => text,
  ? edition => text,
  ? entitlement-data-required => bool,
  ? entitlement-key => text,
  ? generator =>  text / bstr .size 16,
  ? persistent-id => text,
  ? product => text,
  ? product-family => text,
  ? revision => text,
  ? summary => text,
  ? unspsc-code => text,
  ? unspsc-version => text,
  * $$software-meta-extension,
  global-attributes,
}

path-elements-group = ( ? directory => one-or-more<directory-entry>,
                        ? file => one-or-more<file-entry>,
                      )

resource-collection = (
  path-elements-group,
  ? process => one-or-more<process-entry>,
  ? resource => one-or-more<resource-entry>,
  * $$resource-collection-extension,
)
```

```
file-entry = {
  filesystem-item,
  ? size => uint,
  ? file-version => text,
  ? hash => hash-entry,
  * $$file-extension,
  global-attributes,
}

directory-entry = {
  filesystem-item,
  ? path-elements => { path-elements-group },
  * $$directory-extension,
  global-attributes,
}

process-entry = {
  process-name => text,
  ? pid => integer,
  * $$process-extension,
  global-attributes,
}

resource-entry = {
  type => text,
  * $$resource-extension,
  global-attributes,
}

filesystem-item = (
  ? key => bool,
  ? location => text,
  fs-name => text,
  ? root => text,
)

payload-entry = {
  resource-collection,
  * $$payload-extension,
  global-attributes,
}

evidence-entry = {
  resource-collection,
  ? date => integer-time,
  ? device-id => text,
  ? location => text,
  * $$evidence-extension,
```

```
  global-attributes,
}

integer-time = #6.1(int)

tag-id = 0
software-name = 1
entity = 2
evidence = 3
link = 4
software-meta = 5
payload = 6
hash = 7
corpus = 8
patch = 9
media = 10
supplemental = 11
tag-version = 12
software-version = 13
version-scheme = 14
lang = 15
directory = 16
file = 17
process = 18
resource = 19
size = 20
file-version = 21
key = 22
location = 23
fs-name = 24
root = 25
path-elements = 26
process-name = 27
pid = 28
type = 29
entity-name = 31
reg-id = 32
role = 33
thumbprint = 34
date = 35
device-id = 36
artifact = 37
href = 38
ownership = 39
rel = 40
media-type = 41
use = 42
activation-status = 43
channel-type = 44
```

```
colloquial-version = 45
description = 46
edition = 47
entitlement-data-required = 48
entitlement-key = 49
generator = 50
persistent-id = 51
product = 52
product-family = 53
revision = 54
summary = 55
unspsc-code = 56
unspsc-version = 57

multipartnumeric = 1
multipartnumeric-suffix = 2
alphanumeric = 3
decimal = 4
semver = 16384

tag-creator=1
software-creator=2
aggregator=3
distributor=4
licensor=5
maintainer=6

abandon=1
private=2
shared=3

ancestor=1
component=2
feature=3
installationmedia=4
packageinstaller=5
parent=6
patches=7
requires=8
see-also=9
supersedes=10

optional=1
required=2
recommended=3
```

Acknowledgments

Carl Wallace for review and comments on this document.

Contributors

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Carsten Bormann contributed to the CDDL specifications and the IANA considerations.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT

Email: henk.birkholz@sit.fraunhofer.de

Thomas Fossati
arm

Email: Thomas.Fossati@arm.com

Yogesh Deshpande
arm

Email: yogesh.deshpande@arm.com

Ned Smith
Intel

Email: ned.smith@intel.com

Wei Pan
Huawei Technologies

Email: william.panwei@huawei.com