Remote ATtestation ProcedureS
Internet-Draft
Intended status: Standards Track
Expires: 3 September 2026

H. Birkholz
Fraunhofer SIT
T. Fossati
Linaro
Y. Deshpande
arm
N. Smith
Independent
W. Pan
Huawei Technologies
2 March 2026

Concise Reference Integrity Manifest
draft-ietf-rats-corim-10

Abstract

Remote Attestation Procedures (RATS) enable Relying Parties to assess the trustworthiness of a remote Attester and therefore to decide whether or not 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.

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/ietf-rats-wg/draft-ietf-rats-corim.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The RATS Architecture Section 4 of [RFC9334] specifies several roles, including Endorsers and Reference Value Providers. These two roles are typically fulfilled by supply chain actors, such as manufacturers, distributors, or device owners. Endorsers and Reference Value Providers supply Endorsements (e.g., test results or certification data) and Reference Values (e.g., digest) relating to an Attester. This information is used by a Verifier to appraise

Evidence received from an Attester which describes Attester operational state.

In a complex supply chain, multiple actors will likely produce these values over several points in time. As such, one supply chain actor might only supply a portion of the Reference Values or Endorsements that otherwise fully characterizes an Attester. Ideally, only the supply chain actor who is the most knowledgable entity regarding a particular component will supply Reference Values or Endorsements for that component.

Attesters vary across vendors and even across products from a single vendor. 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.

In order to promote inter-operability, consistency and accuracy in the representation of Endorsements and Reference Values this document specifies a data model for Endorsements and Reference Values known as Concise Reference Integrity Manifests (CoRIM). The CoRIM data model is expressed in CDDL which is used to realize a CBOR [STD94] encoding suitable for cryptographic operations (e.g., hashing, signing, encryption) and transmission over computer networks. Additionally, this document describes multiple phases of a Verifier Appraisal and provides an example of a possible use of CoRIM messages from multiple supply chain actors to represent a homogeneous representation of Attester state. CoRIM is extensible to accommodate supply chain diversity while supporting a common representation for Endorsement and Reference Value inputs to Verifiers. See Section 2.

1.1. Terminology and Requirements Language

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.

This document uses terms and concepts defined by the RATS architecture. Specifically the terms Attester, Reference Value Provider, Endorser, Verifier Owner, and Relying Party are taken from Section 4 of [RFC9334].

For a complete glossary, see Section 4 of [RFC9334].

This document uses the terms _"actual state"_ and _"reference state"_ as defined in Section 2 of [I-D.ietf-rats-endorsements].

In this document, the term CoRIM message and CoRIM documents are used as synonyms. A CoRIM data structure can be at rest (e.g., residing in a file system as a document) or can be in flight (e.g., conveyed as a message in a protocol exchange). The bytes composing the CoRIM data structure are the same either way.

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]. Terms and concepts are always referenced as proper nouns, i.e., with Capital Letters.

1.1.1. Glossary

This document uses the following terms:

Appraisal Claims Set (ACS):
: A structure that holds Environment-Claim Tuples that have been appraised. The ACS contains Attester state that has been authorized by Verifier processing and Appraisal Policy.

Appraisal Policy:
: A description of the conditions that, if met, allow appraisal 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. See also "Appraisal Policy for Evidence" in [RFC9334].

Attestation Results Set (ARS):
: A structure that holds results of appraisal and Environment-Claim Tuples that are used to construct an Attestation Results message that is conveyed to a Relying Party.

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.

Claim:
: A piece of information, in the form of a key-value pair. See also Section 4.2 of [RFC9334] and Section 2 of [RFC7519].

Class ID:
: An identifier for an Environment that is shared among similar Environment instances, such as those with the same hardware assembly. See also Section 4.2.4 of [RFC9711].

Composite Attester:
A Composite Attester is either a Composite Device (Section 3.3 of [RFC9334]) or a Layered Attester (Section 3.2 of [RFC9334]) or any composition involving a combination of one or more Composite Devices or Layered Attesters.

Domain:
A domain is a hierarchical description of a Composite Attester in terms of its constituent Environments and their compositional relationships.

Endorsed values:
A set of characteristics of an Attester that do not appear in Evidence. For example, Endorsed Values may include testing or certification data related to a hardware or firmware module. Endorsed Values are said to be "conditional" when they apply if Attester's actual state matches Verifier's accepted Claims. See also Section 3 of [I-D.ietf-rats-endorsements].

Environment:
A logical partition within an Attester. The term "Target Environment" refers to the group of system security metrics that are reported through Evidence. The term "Attesting Environment" refers to the entity that collects and cryptographically signs such security metrics. See also Section 3.1 of [RFC9334].

Environment-Claim Tuple (ECT):
A structure containing a set of values that describe a Target Environment plus a set of Measurement / Claim values that describe properties of the Target Environment. The ECT also contains Authority which identifies the entity that authored the ECT.

Instance ID:
An identifier of an Environment that is unique to that Environment instance, such as the serial number of a hardware module. See also Section 4.2.1 of [RFC9711].

Measurement:
A value associated with specific security characteristics of an Attester that influences the trustworthiness of that Attester. The object of a Measurement could be the invariant part of a firmware component loaded into memory during startup, a run-time integrity check (RTIC), a file system object, or a CPU register. A measured object is part of the Attester's Target Environment. Expected, or "golden," Measurements are compiled as Reference Values, which are used by the Verifier to assess the trust state of the Attester. See also [TNC.Arch], and Section 9.5.5 of [TPM2.Part1].

Reference Values:
: A set of values that represent the desired or undesired state of an Attester. Reference Values are compared against Evidence to determine whether Attester state is corroborated by a Reference Value Provider. Reference Values with matching Evidence produce "acceptable Claims." See also Section 4.2 of [RFC9334], Section 8.3 of [RFC9334], and Section 2 of [I-D.ietf-rats-endorsements].

Triple:
: A term derived from the Resource Description Framework (RDF) to mean a statement expressing a relationship between a subject and an object resource. The nature of the relationship between subject and object is expressed via a predicate. In CoRIM, unlike RDF, the predicate of the triple is implicit and is encoded in the triple's name/codepoint. CoRIM triples typically represent assertions made by the CoRIM author regarding Attesting or Target Environments and their security features, such as Measurements and cryptographic key material. See also Section 3.1 of [W3C.rdf11-primer].

2. Verifier Reconciliation

This specification describes the CoRIM format and documents how a Verifier must process the CoRIM. This ensures that the behaviour of the CoRIM-based appraisal is predictable and consistent, in a word deterministic.

A Verifier needs to reconcile its various inputs, with CoRIM being one of them. In addition to the external CoRIM documents, the Verifier is expected to create an internal representation for each input and map each external representation to an internal one. By using the internal representation, the Verifier processes inputs as if they are part of a conversation, keeping track of who said what. The origin of the inputs is tracked as _authority_. The authority for the Claims in a CoRIM is the CoRIM issuer. To this effect, this specification defines one possible internal representation of the attester's actual state for use during the appraisal procedure, known as Appraisal Claims Set (ACS).

Effectively, Attesters, Reference Value Providers, Endorsers, Verifier Owners, Relying Parties, and even the Verifier potentially all contribute to the conversation. Each producer of corresponding RATS Conceptual Messages can assert Claims about an Attester's actual or allowed state. The Verifier's objective is to produce a list of Claims that describe the Attester's presumed actual state. Producers of RATS Conceptual Messages can assert contradictory assertions. For example, a compromised Attester may produce false claims that

conflict with the Reference Values provided by a Reference Value Provider (RVP). In essence, if Evidence is not corroborated by an RVP's Claims, then the RVP's Claims are not included in the ACS. Please see Figure 1.

A Verifier relies on input from appraisal policy to identify relevant assertions included in the ACS. For example, if a policy requires corroborated assertions issued by a particular RVP, then those assertions may be conveyed as Attestation Results. The Verifier may produce new assertions as a result of an applied appraisal policy. For example, if an appraisal procedure finds all of the components of a subsystem are configured correctly, the policy may direct the Verifier to produce new assertions, "Subsystem=X" has the Claim "TRUSTED=TRUE". Consequently, the internal ACS structure is a reconciled conversation between several producers of RATS Conceptual Messages that has mapped each message into a consistent internal representation, has associated the identity of the corresponding RATS role with each assertion (the authority), and has applied Conceptual Message constraints to the assertion.

The CoRIM data model specified in this document covers the RATS Conceptual Message types, "Reference Values" and "Endorsements". Reference values and Endorsements are required for Verifier reconciliation, and Evidence is required for corresponding internal ACS creation as illustrated in Section 2.2.

2.1. Internal Representation

In this document CDDL is used to specify both the CoRIM structure and to specify an internal representation for use in the appraisal procedure. The actual internal representation of a Verifier is implementation-specific and out-of-scope of this document. Requirements for an internal representation of Conceptual Messages are defined in Table 1, where each Conceptual Message type has a structure as depicted by the _Structure_ column. The internal representations used by this document are defined in Section 9.1.

2.2. Interacting with an ACS

Conceptual Messages interact with an ACS by specifying criteria that should be met by the ACS and by presenting the assertions that should be added to the ACS if the criteria are satisfied. The processing sequence of Conceptual Message interaction with ACS is guided by Section 8.1.

The internal representations of Conceptual Messages and ACS SHOULD satisfy the requirements in Table 1 for Verifier reconciliation and appraisal processing:

CM Type	Structure	Description
Evidence	List of Evidence claims	If the Attester is authenticated, add Evidence claims to the ACS with Attester authority
Reference Values	List of Reference Values claims	If a reference value in a CoRIM matches claims in the ACS, then the authority of the CoRIM issuer is added to those claims.
Endorsements	List of expected actual state claims, List of Endorsed Values claims	If the list of expected claims are in the ACS, then add the list of Endorsed Values claims to the ACS with Endorser authority
Series Endorsements	List of expected actual state claims and a series of selection-addition tuples	If the expected claims are in the ACS, and if the series selection condition is satisfied, then add the additional claims to the ACS with Endorser authority. See Section 9.1.5.1

Table 1: Conceptual Message Representation Requirements

3. Typographical Conventions for CDDL

The CDDL definitions in this document follows the naming conventions illustrated in Table 2.

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 2: Type Traits and Typographical Conventions

4. Concise Reference Integrity Manifest (CoRIM)

A CoRIM is a collection of tags and related metadata in a concise CBOR [STD94] encoding. A CoRIM can be digitally signed with a COSE [STD96] signature. A tag is a structured, machine-readable data format used to uniquely identify, describe, and manage modules or components of a system.

Tags can be of different types:

* Concise Module ID (CoMID) tags (Section 5) contain metadata and claims about the hardware and firmware modules.

* Concise Software ID (CoSWID) tags ([RFC9393]) are used to identify, describe and manage software components.

* Concise Tag List (CoTL) tags (Section 6) contain the list of CoMID and CoSWID tags that the Verifier should consider as "active" at a certain point in time.

CoRIM allows for new types of tags to be added in future specifications. 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.

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 CDDL definition to be customized to fit a specific Attester by augmenting the base CDDL data definition via the specified extension points or by constraining types defined. A profile MUST NOT change the base CoRIM CDDL definition's semantics, which includes not changing or overloading names and numbers registered at IANA registries used by this document. For more detail, see Section 4.1.4.

* 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 4.2) 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 #6.501 CBOR-tagged payload (Section 4.1) and transported over a secure channel.

The following CDDL describes the top-level CoRIM.

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

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

4.1. 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
}
unsigned-corim-map = corim-map
```

The following describes each child item of this map.

* id (index 0): A globally unique identifier to identify a CoRIM. Described in Section 4.1.1.

* tags (index 1): An array of one or more CoMID, CoSWID or CoTL tags. Described in Section 4.1.2.

* dependent-rims (index 2): One or more services supplying additional, possibly dependent, manifests or related files. Described in Section 4.1.3.

* 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. See Section 4.1.4

* rim-validity (index 4): Specifies the validity period of the CoRIM. Described in Section 7.3.

* entities (index 5): A list of entities involved in a CoRIM life-cycle. Described in Section 4.1.5.

* $$corim-map-extension: This CDDL socket is used to add new information structures to the corim-map. Described in Section 12.3.

A corim-map is unsigned, and its tagged form is an entrypoint for parsing a CoRIM, so it is named tagged-unsigned-corim-map. ~~~ cddl
tagged-unsigned-corim-map = #6.501(unsigned-corim-map) ~~~

4.1.1. CoRIM Identifier

A CoRIM Identifier uniquely identifies a CoRIM instance within the context of a CoRIM issuer. In other words the CoRIM identifier is not guaranteed to be globally unique, but can be used to distinguish CoRIMs that come from the same issuer. The base CDDL definition 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
```

4.1.2. Tags

A $concise-tag-type-choice is a tagged CBOR payload that carries either a CoMID (Section 5), a CoSWID ([RFC9393]), or a CoTL (Section 6).

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

4.1.3. Locator Map

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

```
;# import measured-component as eatmc

corim-locator-map = {
  &(href: 0) => uri / [ + uri ]
  ? &(thumbprint: 1) => eatmc.digest / [ eatmc.digest ]
}
```

The following describes each child element of this type.

* href (index 0): a URI or array of alternative URIs identifying locations where the additional resource can be fetched.

* thumbprint (index 1): expected digest or an array of digests referenced by href or an array of hrefs. See sec-common-hash-entry}}.

4.1.4. Profile Types

Profiling is the mechanism that allows the base CoRIM CDDL definition to be customized 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 CoRIM types defined in this document.

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 (Section 5.2). 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 type defined in Section 12.12.1 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.

The CoRIM profile must describe at a minimum the following: (a) how cryptographic verification key material is represented (e.g., using Attestation Keys triples, or CoTS tags), and (b) how key material is associated with the Attesting Environment. The CoRIM profile should also specify whether CBOR deterministic encoding is required.

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

For an example profile definition, see [I-D.fdb-rats-psa-endorsements].

4.1.5. Entities

The CoRIM Entity is an instantiation of the Entity generic (Section 7.2) 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)
$corim-role-type-choice /= &(manifest-signer: 2)
```

The corim-entity-map MUST NOT contain two entities with the manifest-signer role.

4.2. 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 4.2.3) or alternatively in a CWT-Claims ([RFC9597]).

```
COSE-Sign1-corim = [
  protected: bstr .cbor protected-corim-header-map
  unprotected: unprotected-corim-header-map
  payload: bstr .cbor tagged-unsigned-corim-map / hash-envelope-digest / nil
  signature: bstr
]

hash-envelope-digest = 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: When the payload is signed directly, either a CBOR-encoded tagged CoRIM, or nil if it is detached. When the payload is signed indirectly, the digest of a CBOR-encoded tagged CoRIM.

* signature: A COSE signature block, as defined in Section 4 of [STD96].

4.2.1. Protected Header Map

```
   ; protected corim header map needs to contain at least one of corim-meta (8) or CWT-Cl
aims (15)
   protected-corim-header-map = protected-corim-header-map-inline / protected-corim-heade
r-map-hash-envelope

   protected-corim-header-map-inline =
     {
       &(alg: 1) => int,
       &(content-type: 3) => "application/rim+cbor",
       meta-group,
       * cose-label => cose-value
     }

   protected-corim-header-map-hash-envelope =
     {
       &(alg: 1) => int,
       &(payload_hash_alg: 258) => int
       &(payload_preimage_content_type: 259) => "application/rim+cbor",
       ? &(payload_location: 260) => tstr
       meta-group,
       * cose-label => cose-value
     }

   meta-group = ((
           corim-meta-identity,
           ? cwt-claims-identity,
         ) // cwt-claims-identity)

   corim-meta-identity = (&(corim-meta: 8) => bstr .cbor corim-meta-map)
   cwt-claims-identity = (&(CWT-Claims: 15) => cwt-claims)
```

The CoRIM protected header map uses some common COSE header parameters plus additional metadata. Additional metadata can either be carried in a CWT_Claims (index: 15) parameter as defined by [RFC9597], or in a corim-meta map as a legacy alternative, described in Section 4.2.3.

The following describes each child item of this map.

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

Either, when the payload is signed directly:

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

Or, when the payload is signed indirectly using a Hash Envelope ([I-D.ietf-cose-hash-envelope]):

* payload_hash_alg (index 258): The hash algorithm used to produce the payload.

* payload_preimage_content_type (index 259): A string that represents the "MIME Content type" of the CoRIM document used as the pre-image of the payload.

* payload_location (index 260): An optional identifier enabling retrieval of the original resource (preimage) identified by the payload.

At least one of:

* CWT-Claims (index 15): A map that contains metadata associated with a signed CoRIM. Described in [RFC9597].

* corim-meta (index 8): A map that contains metadata associated with a signed CoRIM. Described in Section 4.2.3.

Documents MAY include both CWT-Claims and corim-meta, in which case the signer MUST ensure that their contents are semantically identical: the CWT-Claims issuer (iss) MUST have the same value as signer-name in corim-meta, and the nbf and exp values in the CWT-Claims MUST match the signature-validity in corim-meta.

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

4.2.2. CWT Claims

The CWT Claims ([RFC9597]) map identifies the entity that created and signed the CoRIM. This ensures the consumer is able to identify credentials used to authenticate its signer. To avoid any possible ambiguity with the contents of the CoRIM tags, the CWT Claims map MUST NOT contain claims that have semantic overlap with the information contained in CoRIM tags.

The following describes each child item of this group.

* iss (index 1): Issuer or signer for the CoRIM, formerly signer-name or signer-uri in Section 4.2.3.1.

* sub (index 2): Optional - identifies the CoRIM document, equivalent to a string representation of $corim-id-type-choice

Additional data can be included in the CWT Claims, as per [RFC8392], such as:

* exp (index 4): Expiration time, formerly signature-validity in Section 7.3.

* nbf (index 5): Not before time, formerly signature-validity in Section 7.3.

4.2.3. 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 4.2.3.1.

* signature-validity (index 1): Validity period for the CoRIM. Described in Section 7.3.

4.2.3.1. 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.

4.2.4. Unprotected CoRIM Header Map

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

4.3. Signer authority of securely conveyed unsigned CoRIM

An unsigned (#6.501-tagged) CoRIM may be a payload in an enveloping signed document, or it may be conveyed unsigned within the protection scope of a secure channel. The CoRIM signer authority is taken from the authenticated credential (e.g., OAUTH token) of the entity that originates the CoRIM. For example, this entity could be the sending peer in a secure channel. A CoRIM role entry expressing the origin of the unsigned CoRIM (i.e., the enveloping signed document or the origin endpoint of the secure channel) via the manifest-signer role MUST be added to corim-entity-map. If the authority cannot be expressed directly via the existing authority types, the receiver SHOULD establish a local authority in one of the supported authority formats (e.g., if an unsigned CoRIM is received over a secure channel where authentication is token- or password-based). If it is impossible to assert the authority of the origin, the Verifier's appraisal policy MAY assert the Verifier' s authority as the CoRIM origin.

It is out of scope of this document to specify a method of delegating the signer role in the case that an unsigned CoRIM is conveyed through multiple secured links with different notions of authenticity without end-to-end integrity protection.

4.3.1. CoRIM collections

Several CoRIMs may share the same signer (e.g., as collection payload in a different signed message) and use locally-resolvable references to each other, for example using a RATS Conceptual Message Wrapper (CMW) [I-D.ietf-rats-msg-wrap]. The Collection CMW type is similar to a profile in its way of restricting the shape of the CMW collection. The Collection CMW type for a CoRIM collection SHALL be tag:{{&SELF}}:corim.

A COSE_Sign1-signed CoRIM Collection CMW has a similar requirement to a signed CoRIM. The signing operation MUST include either a CWT-Claims or a corim-meta and MAY contain both, in the COSE_Sign1 protected-header parameter. These metadata containers ensure that each CoRIM in the collection has an identified signer. The COSE protected header can include a Collection CMW type name by using the cmwc_t content type parameter for the &(content-type: 3) COSE header, or &(payload_preimage_content_type: 259) in the case of hash envelopes.

If using other signing envelope formats, the CoRIM signing authority MUST be specified. For example, this can be accomplished by adding the manifest-signer role to every CoRIM, or by using a protected header analogous to corim-meta.

```
corim-cbor-collection = {
  "__cmwc_t" => "tag:{{&SELF}}:corim",
  + cmw-label => [TBD1, bytes .cbor tagged-corim-map]
}
cmw-label = text / int
```

The Collection CMW MAY use any label for its CoRIMs. If there is a hierarchical structure to the CoRIM Collection CMW, the base entry point SHOULD be labeled 0 in CBOR or "base" in JSON. It is RECOMMENDED to label a CoRIM with its tag-id in string format, where uuid-type string format is specified by [RFC9562]. CoRIMs distributed in a CoRIM Collection CMW MAY declare their interdependence dependent-rims with local resource indicators. It is RECOMMENDED that a CoRIM with a uuid-type tag-id be referenced with URI urn:uuid:_tag-id-uuid-string_. It is RECOMMENDED that a CoRIM with a tstr tag-id be referenced with tag:{{&SELF}}:local,_tag-id-tstr_. It is RECOMMENDED for a corim-locator-map containing local URIs to afterwards list a nonzero number of reachable URLs as remote references.

The following example demonstrates these recommendations for bundling CoRIMs with a common signer but have different profiles.

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

/ cbor-collection / {
  "__cmwc_t": "tag:{{&SELF}}:corim",
  "adee8cd4-4e47-461f-b341-2aba3ae4cbda":
    / cbor-cmw / [TBD1, << / tagged-corim-map / 501({
      / id / 0: h'adee8cd44e47461fb3412aba3ae4cbda',
      / tags / 1: [/ tagged-concise-mid-tag / 506(<<{
        / tag-identity / 1: {
          / tag-id / 0: h'315cfc0208d548ee9c89906df96e7fb8'
        },
        / triples / 4: {
          / reference-triples / 0: [[
            / ref-env / {/ class / 0: {/ class-id / 0: 560(h'c0de')}\
                                                                  },
            / ref-claims / {
              / mval / 1: {
                / profile-foo / -404:
                  h'\
                  6094685f8bb9b67daaf35707bd2c391f536a1df2ce5152c3cc'
              }}]],
          / coswid-triples / 6: [[
            {/ class / 0: {/ class-id / 0: 560(h'c0de')}},
            [h'369a6688451240b9b74905b1dd5fae9f']]]}}>>)],
      / profile / 3: "tag:example.com,2025/example-profile",
      / dependent-rims / 2: [{
        / href / 0: [
          "urn:uuid:51a5f633-71c0-45f5-855e-43e8254c1806",
          "https://example.com/369a6688451240b9b74905b1dd5fae9f.\
                                                            corim"
      ]}]}) >>],
  "51a5f633-71c0-45f5-855e-43e8254c1806":
    / cbor-cmw / [TBD1, << / tagged-corim-map / 501({
      / id: / 0: h'51a5f63371c045f5855e43e8254c1806',
      / tags / 1: [/ tagged-concise-swid-tag / 505(<<{
          / tag-id / 0: h'369a6688451240b9b74905b1dd5fae9f',
          / tag-version / 12: 2,
          / software-name / 1: "Gadget Firmware",
          / entity / 2: {
            / entity-name / 31: "ACME Firmware",
            / role / 33: / software-creator / 2
          },
          / profile-bar / 4294967295: {
           / profile-baz / 0: 5,
           / profile-qux / 1: h'f76e41f3462a62e8'}}>>)],
      / profile / 3: "tag:example.com,2025/software-profile"})>>]
}
```

5. 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 unambiguously identify CoMID instances when cross referencing CoMID tags, for example in typed link relations, or in a CoTL 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.

This specification defines two classes of triples, the Mandatory To Implement (MTI) and the Optional To Implement (OTI). The MTI triples are essential to basic appraisal processing as illustrated in [RFC9334] and [I-D.ietf-rats-endorsements]. Every CoRIM Verifier MUST implement the MTI triples. The OTI class of triples are generally useful across profiles. A CoRIM Verifier SHOULD implement OTI triples. Verifiers may be constrained in various ways that may make implementation of the OTI class infeasible or unnecessary. For example, deployment environments may have constrained resources, limited code size, or limited scope Attesters.

MTI Triples:

* Reference Values triples: containing Reference Values that are expected to match Evidence for a given Target Environment (Section 5.1.5).

* 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 5.1.6).

* Conditional Endorsement triples: describing one or more conditions that, once matched, result in augmenting the Attester’s actual state with the supplied Endorsed Values (Section 5.1.7).

OTI Triples:

* Conditional Endorsement Series triples: describing conditional endorsements that are evaluated using a special matching algorithm (Section 5.1.7).

* Device Identity triples: containing cryptographic credentials - for example, an IDevID - uniquely identifying a device (Section 5.1.9).

* Attestation Key triples: containing cryptographic keys that are used to verify the integrity protection on the Evidence received from the Attester (Section 5.1.10).

* Domain dependency triples: describing trust relationships between domains, i.e., collection of related environments and their measurements (Section 5.1.11.2).

* 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 5.1.11.1).

* CoMID-CoSWID linking triples: associating a Target Environment with existing CoSWID Payload tags (Section 5.1.12).

CoMID triples are extensible (Section 5.1.4). Triples added via the extensibility feature MUST be OTI class triples. This document specifies profiles (see Section 5.2). OTI triples MAY be reclassified as MTI using a profile. Conversely, profiles can choose not to _use_ certain MTI triples. Profiles MUST NOT reclassify MTI triples as OTI.

5.1. 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 5.1.1.

* entities (index 2): Provides information about one or more organizations responsible for producing the CoMID tag. Described in Section 5.1.2.

* linked-tags (index 3): A list of one or more linked-tag-map providing typed relationships between this and other CoMIDs. Described in Section 5.1.3).

* 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 5.1.4.

5.1.1. 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 5.1.1.1.

* tag-version (index 1): Optional versioning information for the tag-id. Described in Section 5.1.1.2.

5.1.1.1. 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 [RFC9562]. There are no strict guidelines on how the identifier is structured, but examples include a 16-byte GUID (e.g., class 4 UUID) [RFC9562], or a URI [STD66].

5.1.1.2. 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.

5.1.2. Entities

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

The CoMID Entity is an instantiation of entity-map (Section 7.2) 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.

5.1.3. 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. See Section 5.1.1.1.

* 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.

5.1.4. 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.

In most cases, the supply chain entity that is responsible for providing a triple (i.e., Reference Values or Endorsed Values) is by default the CoRIM signer. The signer of a triple is said to be its _authority_. However, multiple authorities may be involved in signing triples. See [STD96]. Consequently, authority may differ for search criteria. See Section 5.1.4.5.

```
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: 10) =>
    [ + 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 5.1.5.

* endorsed-triples (index 1): Triples containing endorsed values. Described in Section 5.1.6.

* identity-triples (index 2): Triples containing identity credentials. Described in Section 5.1.9.

* attest-key-triples (index 3): Triples containing verification keys associated with attesting environments. Described in Section 5.1.10.

* dependency-triples (index 4): Triples describing trust relationships between domains. Described in Section 5.1.11.2.

* membership-triples (index 5): Triples describing topological relationships between (sub-)modules. Described in Section 5.1.11.1.

* coswid-triples (index 6): Triples associating modules with existing CoSWID tags. Described in Section 5.1.12.

* conditional-endorsement-series-triples (index 8): Triples describing a series of Endorsements that are applicable based on the acceptance of a condition. Described in Section 5.1.8.

* conditional-endorsement-triples (index 10): Triples describing a series of conditional Endorsements based on the acceptance of a stateful environment. Described in Section 5.1.7.

5.1.4.1. Environments

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).

An environment MUST be globally unique. The combination of values within class-map MUST combine to form a globally unique identifier.

```
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 5.1.4.2.

* instance (index 1): Contains a unique identifier of a module’s instance. Described in Section 5.1.4.3.

* group (index 2): identifier for a group of instances, e.g., if an anonymization scheme is used. Described in Section 5.1.4.4.

5.1.4.2. Environment 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-bytes
```

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) variable-length opaque byte string (Section 7.8) 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.

5.1.4.3. Environment Instance

An instance-id is a unique value that identifies a Target Environment instance. The identifier is reliably bound to the Target Environment. For example, if an X.509 certificate’s subject public key is unique for each instance of a target environment, the instance-id might be created from that subject public key. See Section 4.1 of [RFC5280]. Alternatively, if the certificate’s subject public key is large, the instance-id might be a key identifier that is a digest of that public key. See Section 4.2.1.2 of [RFC5280]. The key identifier is reliably bound to the subject public key because the identifier is a digest of the key.

The types defined for an instance identifier are CBOR tagged expressions of UEID, UUID, variable-length opaque byte string (Section 7.8), cryptographic keys, or cryptographic key identifiers.

```
$instance-id-type-choice /= tagged-ueid-type
$instance-id-type-choice /= tagged-uuid-type
$instance-id-type-choice /= tagged-bytes
$instance-id-type-choice /= tagged-pkix-base64-key-type
$instance-id-type-choice /= tagged-pkix-base64-cert-type
$instance-id-type-choice /= tagged-cose-key-type
$instance-id-type-choice /= tagged-key-thumbprint-type
$instance-id-type-choice /= tagged-cert-thumbprint-type
$instance-id-type-choice /= tagged-pkix-asn1der-cert-type
```

5.1.4.4. Environment 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 types defined for a group identified are UUID and variable-length opaque byte string (Section 7.8).

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

5.1.4.5. 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 a 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.

An environment may have multiple measured elements. Measured elements are distinguished from each other by measurement keys. Measurement keys may be used to disambiguate measurements of the same type originating from different elements.

Triples that have search conditions may specify authority as matching criteria by populating authorized-by.

```
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 measurement key. Described in Section 5.1.4.5.1. A measurement-map without an mkey is said to be anonymous.

* mval (index 1): The measurements associated with the environment. Described in Section 5.1.4.5.2.

* authorized-by (index 2): The cryptographic identity of the entity (individual or organization) that is the designated authority for measurement Claims. For example, the signer of a CoMID triple. See Section 5.1.4.6. An entity is authoritative when it makes Claims that are inside its area of competence.

5.1.4.5.1. Measurement Keys

Measurement keys SHALL be unique within the scope of the environment-map they are associated with. The initial types defined are OID, UUID, uint, and tstr. mkey may be necessary to disambiguate multiple measurements of the same type or to distinguish multiple measured elements within the same environment. A single anonymous measurement-map is allowed within the same environment. Two or more measurement-map entries within the same environment MUST populate mkey.

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

5.1.4.5.2. 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) => digests-type
  ? &(flags: 3) => flags-map
  ? (
      &(raw-value: 4) => $raw-value-type-choice,
      ? &(raw-value-mask-DEPRECATED: 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: 13) => [ + $crypto-key-type-choice ]
  ? &(integrity-registers: 14) => integrity-registers
  ? &(int-range: 15) => int-range-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 5.1.4.5.3.

* svn (index 1): The security version number typically changes only when a security relevant change is made to the measured environment. Described in Section 5.1.4.5.4.

* digests (index 2): Contains the digest(s) of the measured environment together with the respective hash algorithm used in the process. It uses the digests-type. Described in Section 7.7.

* 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 5.1.4.5.5.

* raw-value (index 4): Contains the actual (not hashed) value of the element. The vendor determines the encoding of raw-value. When used for comparison, the tagged-masked-raw-value variant includes a mask indicating which bits in the value to compare. Described in Section 5.1.4.5.6

* raw-value-mask-DEPRECATED (index 5): Is an obsolete method of indicating which bits in a raw value to compare. New CoMID files should use the tagged-masked-raw-value on index 4 instead of using index 5.

* mac-addr (index 6): An EUI-48 (Extended Unique Identifier 48) or EUI-64 MAC address [IEEE-802.OandA] associated with the measured environment. Described in Section 5.1.4.5.7.

* ip-addr (index 7): An IPv4 or IPv6 address associated with the measured environment. Described in Section 5.1.4.5.7.

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

* ueid (index 9): UEID associated with the measured environment. Described in Section 7.5.

* uuid (index 10): UUID associated with the measured environment. Described in Section 7.4.

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

* cryptokeys (index 13): identifies cryptographic keys that are protected by the Target Environment See Section 5.1.4.6 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 9.4.6.1.5.

* integrity-registers (index 14): A group of one or more named measurements associated with the environment. Described in Section 5.1.4.7.

5.1.4.5.3. Version

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

```
;# import rfc9393 as coswid

version-map = {
  &(version: 0) => text
  ? &(version-scheme: 1) => coswid.$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 [RFC9393]. 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
```

5.1.4.5.4. 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 cannot be decremented. If a security relevant flaw is discovered in the Target Environment and is subsequently fixed, 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 = svn / tagged-svn / tagged-min-svn
```

5.1.4.5.5. 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. Note that, while the fields may not be completely independent of one another, this specification imposes no restrictions on combinations of the flags-map booleans. However, a profile may restrict the possible flags-map booleans and their valid combinations.

```
flags-map = non-empty<{
  ? &(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 rollback to previous software images.

* 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.

5.1.4.5.6. 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. A profile may choose to define more specific semantic meaning to a raw value.

A raw-value measurement, or an Endorsement, is a tagged value of type bytes. This specification defines tag #6.560. The default raw value measurement is of type tagged-bytes (Section 7.8).

Additional value types can be added to $raw-value-type-choice. These additional values MUST be CBOR tagged bstrs. Constraining all raw value types to be bstr lets Verifiers compare raw values without understanding their contents.

A raw value intended for comparison can include a mask value, which selects the bits to compare during appraisal. 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.

The raw-value-mask-DEPRECATED in measurement-values-map is deprecated, but retained for backwards compatibility. This code point may be removed in a future revision of this specification.

```
$raw-value-type-choice /= tagged-bytes
$raw-value-type-choice /= tagged-masked-raw-value

raw-value-mask-type = bytes
tagged-masked-raw-value = #6.563([
  value: bytes
  mask : bytes
])
```

5.1.4.5.7. Address Types

This specification defines types for 48-bit and 64-bit MAC identifiers. For IP addresses, it reuses the "Address Format" types defined in [RFC9164] with the CBOR tag removed.

All the types represent a single address.

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

;# import rfc9164 as cbor-ip

ip-addr-type-choice /= cbor-ip.ipv4-address
ip-addr-type-choice /= cbor-ip.ipv6-address
```

5.1.4.6. 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].

* tagged-pkix-asn1der-cert-type: a bstr of ASN.1 DER encoded X.509 public key certificate. Defined in Section 4 of [RFC5280].

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

* tagged-key-thumbprint-type: a digest (e.g., the SHA-2 hash) of a raw public key. For example, the digest value can be used to locate a public key contained in a lookup table. Ultimately, the discovered keys have to be successfully byte-by-byte compared with the corresponding keys.

* 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.

* tagged-bytes: a key identifier with no prescribed construction method.

```
;# import measured-component as eatmc

$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-pkix-asn1der-cert-type
$crypto-key-type-choice /= tagged-key-thumbprint-type
$crypto-key-type-choice /= tagged-cert-thumbprint-type
$crypto-key-type-choice /= tagged-cert-path-thumbprint-type
$crypto-key-type-choice /= tagged-bytes
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-key-thumbprint-type = #6.557(eatmc.digest)
tagged-cose-key-type = #6.558(COSE_Key)
tagged-cert-thumbprint-type = #6.559(eatmc.digest)
tagged-cert-path-thumbprint-type = #6.561(eatmc.digest)
tagged-pkix-asn1der-cert-type = #6.562(bstr)
```

5.1.4.7. Integrity Registers

An Integrity Registers map groups together one or more measured "objects". Each measured object has a unique identifier and one or more associated digests. Identifiers are either unsigned integers or text strings and their type matters, e.g., unsigned integer 5 is distinct from the text string "5". The digests use digests-type semantics (Section 7.7).

```
integrity-register-id-type-choice = uint / text

integrity-registers = {
  + integrity-register-id-type-choice => digests-type
}
```

All the measured objects in an Integrity Registers map are explicitly named and the order in which they appear in the map is irrelevant. Any digests associated with a measured object represent an acceptable state for the object. Therefore, if multiple digests are provided, the acceptable state is their cross-product. For example, given the following Integrity Registers:

```
{
  0: [ [ 0, h'00' ] ],
  1: [ [ 0, h'11' ], [ 1, h'12' ] ]
}
```

then both

```
{
  0: [ 0, h'00' ],
  1: [ 0, h'11' ]
}
```

and

```
{
  0: [ 0, h'00' ],
  1: [ 1, h'12' ]
}
```

are acceptable states.

Integrity Registers can be used to model the PCRs in a TPM or vTPM, in which case the identifier is the register index, or other kinds of vendor-specific measured objects.

5.1.4.8. Int Range

An int range describes an integer value that can be compared with linear order in the target environment. An int range is represented with either major type 0 or major type 1 ints.

```
int-range-type-choice = int / tagged-int-range
int-range = [min: int / negative-inf, max: int / positive-inf]
tagged-int-range = #6.564(int-range)
positive-inf = null
negative-inf = null
```

The signed integer range representation is an inclusive range unless either min or max are infinite as represented by null, in which case, each infinity is necessarily exclusive.

5.1.5. Reference Values Triple

Reference Values Triples describe the possible intended states of an Attester. At any given point in time, an Attester is expected to match only one of these states.

A Reference Values Triple provides reference values pertaining to a Target Environment. In a Reference Value triple, the subject identifies a Target Environment, the object contains reference measurements associated with one or more measured elements of the Environment, and the predicate asserts that these represent the expected state of the Target Environment.

The Reference Values Triple has the following structure:

```
reference-triple-record = [
  ref-env: environment-map
  ref-claims: [ + measurement-map ]
]
```

The reference-triple-record has the following parameters:

* ref-env: Identifies the Target Environment

* ref-claims: Contains one or more reference measurements for the Target Environment

CoMID triples (Section 5.1.4) may contain multiple reference-triple-record entries, each of which describes one or more possible states for a particular Target Environment.

The ref-claims in a reference-triple-record can contain one or more entries. This multiplicity can have different meanings:

1. Each ref-claims entry can represent a different possible state of the Environment.

2. Each ref-claims entry can represent a possible state of a different measured element (identified by its mkey) within the Environment.

Note that the same semantics can be expressed using multiple Reference Value Triples.

Note also that a measurement key-value pair could be defined to have multiple values or use "wild carding" to describe a range of acceptable values, for example when using int-range and min-svn.

Any of these multiplicities could be used in the context of Reference Values Triples.

To process a reference-triple-record, the ref-env and ref-claims criteria are compared with Evidence entries. First, ref-env is used as search criteria to locate matching Evidence environments. Then, the ref-claims from this triple are used to match against the Evidence measurements for a matched environment. If the search criteria are satisfied, the matching entry is added to the body of Attester state, except these Claims are asserted with the Reference Value Provider’s authority. By re-asserting Evidence matched with Reference Values using the RVP’s authority, the Verifier avoids confusing Reference Values (reference / possible state) with Evidence (actual state). See [I-D.ietf-rats-endorsements]. Evidence Claims that are re-asserted using RVP authority are said to be "corroborated Evidence" because the actual state in Evidence was found within the corpus of the RVP’s possible state.

5.1.6. Endorsed Values Triple

An Endorsed Values triple provides additional Endorsements - i.e., claims reflecting the actual state - for an existing Target Environment. For Endorsed Values Claims, the subject is a Target Environment, the object contains Endorsement Claims for the Environment, and the predicate defines semantics for how the object relates to the subject.

The Endorsed Values Triple has the following structure:

```
endorsed-triple-record = [
  condition: environment-map
  endorsement: [ + measurement-map ]
]
```

The endorsed-triple-record has the following parameters:

* condition: Search criterion that locates an Evidence, corroborated Evidence, or Endorsements environment.

* endorsement: Additional Endorsement Claims.

To process a endorsed-triple-record the condition is compared with existing Evidence, corroborated Evidence, and Endorsements. If the search criterion is satisfied, the endorsement Claims are combined with the condition environment-map to form a new (actual state) entry. The new entry is added to the existing set of entries using the Endorser's authority.

5.1.7. Conditional Endorsement Triple

A Conditional Endorsement Triple declares one or more conditions that, once matched, results in augmenting the Attester's actual state with the Endorsement Claims. The conditions are expressed via stateful-environment-records, which match Target Environments from Evidence in certain reference state.

The Conditional Endorsement Triple has the following structure:

```
conditional-endorsement-triple-record = [
  conditions: [ + stateful-environment-record ]
  endorsements: [ + endorsed-triple-record ]
]

stateful-environment-record = [
  environment: environment-map,
  claims-list: [ + measurement-map ]
]
```

The conditional-endorsement-triple-record has the following parameters:

* conditions: Search criteria that locates Evidence, corroborated Evidence, or Endorsements.

* endorsements: Additional Endorsements.

To process a conditional-endorsement-triple-record the conditions are compared with existing Evidence, corroborated Evidence, and Endorsements. If the search criteria are satisfied, the endorsements entries are asserted with the Endorser's authority as new Endorsements.

5.1.8. Conditional Endorsement Series Triple

The Conditional Endorsement Series Triple employs a 2-stage matching convention to assert endorsed values based on an initial condition match followed by a series selection match. If both the condition and selection criteria are satisfied, a set of endorsed values are added to the matching triple records. The condition match identifies the set of Claims to which the selection criteria are applied. The selection specifies a pattern of measurements that, if present, controls when a focused set of endorsed values are to be asserted. The 2-stage approach enables Endorsement authors the ability to craft powerful search criteria while avoiding probelmatic repetition of search criteria.

The Conditional Endorsement Series Triple has the following structure:

```
conditional-endorsement-series-triple-record = [
  condition: [
    environment: environment-map
    claims-list: [ * measurement-map ]
    ? authorized-by: [ + $crypto-key-type-choice ]
  ]
  series: [ + conditional-series-record ]
]

conditional-series-record = [
  selection: [ + measurement-map]
  addition: [ + measurement-map ]
]
```

The conditional-endorsement-series-triple-record has the following parameters:

* condition: Initial selection criteria that locates Evidence, corroborated Evidence, or Endorsements from the current set of accepted Claims. The condition consists of an environment-map, a (possibly empty) claims-list, and an optional authorized-by.

* series: A sequence of selection-addition tuples.

The conditional-series-record has the following parameters:

* selection: Secondary selection criteria that locates Evidence, corroborated Evidence, or Endorsements from the initial selection criteria’s condition result.

* addition: Endorsements that are added if the selection criteria are satisfied.

5.1.8.1. Condition Matching

The condition matching criteria is applied to the set of Claims the Verifier has previously accepted. The criteria is expressed in terms of environments (i.e., environment-map) and optionally measurements (i.e., claims-list) or authority (i.e., authorized-by). Condition matching is intended to powerfully enable broad or narrow searches that serve as staging for subsequent selection matching.

Note that measurement-map can also specify authority criteria. To avoid conflicting criteria, the authorized-by in condition takes precedence over the authorized-by in measurement-map.

5.1.8.1.1. Selection Matching

Every conditional-series-record selection MUST select the same mkeys where every selected mkey's corresponding set of code points represented as mval.key MUST be the same across each conditional-series-record. For example, if a selection matches on 3 measurement-map statements; mkey is the same for all 3 statements and mval contains only A= variable-X, B= variable-Y, and C= variable-Z (exactly the set of code points A, B, and C) respectively for every conditional-series-record in the series.

These restrictions ensure that evaluation order does not change the meaning of the triple during the appraisal process. Series entries are ordered such that the most precise match is evaluated first and least precise match is evaluated last. The first series condition that matches terminates series matching and the endorsement values are added to the Attester's actual state.

5.1.8.1.2. Processing the Addition

To process a conditional-endorsement-series-record the selection criteria in condition entries are matched with existing Evidence, corroborated Evidence, and Endorsements. If the selection criteria are satisfied, the series tuples are processed.

The series array contains an ordered list of conditional-series-record entries. Evaluation order begins at list position 0.

For each series entry, if the selection criteria matches an entry found in the condition result, the series addition is combined with the environment-map from the condition result to form a new Endorsement entry. The new entry is added to the existing set of Endorsements.

The first series entry that successfully matches the selection criteria terminates series processing.

5.1.9. Device Identity Triple

Device Identity triples (see identity-triples in Section 5.1.4) endorse that the keys were securely provisioned to the named Target Environment. A single Target Environment (as identified by environment and mkey) may contain one or more cryptographic keys. The existence of these keys is asserted in Evidence, Reference Values, or Endorsements.

The device identity keys may have been used to authenticate the Attester device or may be held in reserve for later use.

Device Identity triples instruct a Verifier to perform key validation checks, such as revocation, certificate path construction & verification, or proof of possession. The Verifier SHOULD verify keys contained in Device Identity triples.

Additional details about how a key was provisioned or is protected may be asserted using Endorsements such as endorsed-triples.

Depending on key formatting, as defined by $crypto-key-type-choice, the Verifier may take different steps to locate and verify the key.

If a key has usage restrictions that limit its use to device identity challenges, the Verifier SHOULD enforce key use restrictions.

Each successful verification of a key in key-list SHALL produce Endorsement Claims that are added to the Attester’s Claim set. Claims are asserted with the joint authority of the Endorser (CoRIM signer) and the Verifier. The Verifier MAY report key verification results as part of an error reporting function.

```
identity-triple-record = [
  environment: environment-map
  key-list: [ + $crypto-key-type-choice ]
  ? conditions: non-empty<{
    ? &(mkey: 0) => $measured-element-type-choice,
    ? &(authorized-by: 1) => [ + $crypto-key-type-choice ]
  }>
]
```

* environment: An environment-map condition used to identify the target Evidence or Reference Value. See Section 5.1.4.1.

* key-list: A list of $crypto-key-type-choice keys that identifies which keys are to be verified. See Section 5.1.4.6.

* mkey: An optional $measured-element-type-choice condition used to identify the element within the target Evidence or Reference Value. See Section 5.1.4.5.1.

* authorized-by: An optional list of $crypto-key-type-choice keys that identifies the authorities that asserted the key-list in the target Evidence or Reference Values.

5.1.10. Attest Key Triple

Attest Key triples (see attest-key-triples in Section 5.1.4) endorse that the keys were securely provisioned to the named Attesting Environment. An Attesting Environment (as identified by environment and mkey) may contain one or more cryptographic keys. The existence of these keys is asserted in Evidence, Reference Values, or Endorsements.

The attestation keys may have been used to sign Evidence or may be held in reserve for later use.

Attest Key triples instruct a Verifier to perform key validation checks, such as revocation, certification path construction and validation, or proof of possession. The Verifier SHOULD verify keys contained in Attest Key triples.

Additional details about how a key was provisioned or is protected may be asserted using Endorsements such as endorsed-triples.

Depending on key formatting, as defined by $crypto-key-type-choice, the Verifier may take different steps to locate and verify the key. If a key has usage restrictions that limits its use to Evidence signing (e.g., see Section 5.1.5.3 in [DICE.cert]). The Verifier SHOULD enforce key use restrictions.

Each successful verification of a key in key-list SHALL produce Endorsement Claims that are added to the Attester's Claim set. Claims are asserted with the joint authority of the Endorser (CoRIM signer) and the Verifier. The Verifier MAY report key verification results as part of an error reporting function.

```
attest-key-triple-record = [
  environment: environment-map
  key-list: [ + $crypto-key-type-choice ]
  ? conditions: non-empty< {
    ? &(mkey: 0) => $measured-element-type-choice,
    ? &(authorized-by: 1) => [ + $crypto-key-type-choice ]
  }>
]
```

See Section 5.1.9 for additional details.

5.1.11. Triples for domain definitions

A domain is a hierarchical description of a Composite Attester in terms of its constituent Environments and their compositional relationships.

The following CDDL describes domain type.

```
domain-type = environment-map
```

Domain structure is defined with the following types of triples.

5.1.11.1. Domain Membership Triple

A Domain Membership Triple (DMT) links a domain identifier to its member Environments. The triple's subject is the domain identifier while the triple' s object lists all the member Environments within the domain.

The Domain Membership Triple allows an Endorser (for example, an Integrator) to issue an authoritative statement about the composition of an Attester as a collection of Environments. This allows a topological description of an Attester to be expressed by linking a parent Environment (e.g., a lead Attester) to its child Environments (e.g., one or more sub-Attesters).

If the Verifier Appraisal policy requires Domain Membership, the Domain Membership Triple is used to match an Attester's reference composition with the actual composition represented in Evidence.

Representing members of a DMT as domains enables the recursive construction of an entity's topology, such as a Composite Device (see Section 3.3 of [RFC9334]), where multiple lower-level domains can be aggregated into a higher-level domain.

```
domain-membership-triple-record = [
  domain-id: domain-type
  members: [ +  domain-type ]
]
```

5.1.11.2. Domain Dependency Triple

A Domain Dependency Triple (DDT) links a domain to a set of _trustee_ domains. A domain dependency triple is used by an Endorser to assert that a trust dependency exists between various components. A DDT specifies which component (identified by domain-id) depends on which other components (identified by trustees) for proper operation. A series of DDTs can be used to describe the trust dependencies of a system of components as a graph. CoRIM uses environment-map to identify components and groupings of components (i.e., domains).

Trust dependency means that an environment can only be fully trusted if one or more trustee environments have been appraised and found to be trustworthy. A candidate environment can only be trusted if the trustee environments it depends on exist, have been appraised and are found to be trustworthy.

The first four phases of appraisal (see Section 8.1) might not determine whether a component is trustworthy. Subsequent Verifier stages or Relying Party processing might be needed to finalize trustworthiness. Therefore, the trustworthiness of trustee domains MUST be appraised before the trustworthiness of the subject domain can be finalized. Consequently, trust dependency semantics may need to be represented in Attestation Results if Relying Parties play a role in finalizing which components are trustworthy.

There are a variety of use cases where trust dependency might exist. For example, trust in an operating system (OS) might depend on trustworthy loading of the OS loader image. Consequently, the OS loader is a trustee domain of the OS. Alternatively, trust in a peripheral device might depend on trustworthy operation of a perpheral device's bus controller. The bus controller is therefore a trustee domain of the peripheral device.

DDTs cannot create domains. Instead, DDT processing first checks that a domain-id has already been accepted into the ACS before adding trust dependencies.

The domain dependency triple subject (domain-id) identifies the member domain (see Section 5.1.11.1) that has trustees. The triple object trustees lists the domains that are trustees of the subject domain. The triple predicate asserts that a trust appraisal of domain-id is not complete without appraisal of the trustees.

```
domain-dependency-triple-record = [
  domain-id: domain-type
  trustees: [ + domain-type ]
]
```

All of the DDT subjects (domain-id) and objects (trustees) MUST also be domain members for the DDT expression to be processed.

Trust dependency graphs are acyclic, meaning a domain-id MUST NOT appear in the trustees list or within a trustee’s subtree.

A terminating "leaf" trustee is a "root of trust" for that subtree. Leaf trustees SHOULD have a corresponding Endorsement triple. Verifiers MAY use DDTs with appraisal policies to assess the veracity of domain-to-trustee linkages.

Trust dependency typically exists if any of the following are true:

* A trustee performs any Attesting Environment functions relating to a Target Environment (TE), such as Claims collection, Claims signing, loading or initialization of the TE, provisioning TE secrets - including cryptographic keys or other security-relevant material.

* A trustee executes security-relevant code in response to an execution thread that originates from the domain-id environment.

* A trustee is a component embedded within another component identified by domain-id.

Trust dependency processing is described in Section 9.1.8.

5.1.12. 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.

```
;# import rfc9393 as coswid

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

5.2. Extensibility

The base CoRIM document definition 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.

5.2.1. Map Extensions

Map extensions provide extensibility support to CoRIM map structures. CDDL map extensibility enables a CoRIM profile to extend the base CoRIM CDDL 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.coswid]. Non-negative integers are reserved for IANA to assign meaning globally.

5.2.2. Data Type Extensions

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

New data type extensions SHOULD be documented to facilitate interoperability. CoRIM profiles are best used to document vendor or industry defined extensions.

6. CoTL

A Concise Tag List (CoTL) object represents the signal for the Verifier to activate the listed tags. Verifier policy determines whether CoTLs are required.

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

The number of CoTLs 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 CoTL, 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 CoTL to activate all the tags submitted to the Verifier in these CoRIMs.

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

6.1. Structure

The CDDL specification for the concise-tl-tag map and additional grammatical requirements specified in the text of this Section MUST be followed when creating or validating a CoTL tag are given below:

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

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

* tag-identity (index 0): A tag-identity-map containing unique identification information for the CoTL. Described in Section 5.1.1.

* tags-list (index 1): One or more tag-identity-maps identifying the CoMID and CoSWID tags that constitute the list, 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, and the activation is atomic. All tags listed in tags-list MUST be activated or no tags are activated.

* tl-validity (index 2): Specifies the validity period of the CoTL. Described in Section 7.3.

7. Common Types

The following CDDL types may be shared by CoRIM, CoMID, and CoTL.

7.1. 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 })
```

7.2. 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 12.6.

* 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 4.1.5) and (Section 5.1.2).

7.3. 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

7.4. UUID

Used to tag a byte string as a binary UUID. Defined in Section 4 of [RFC9562].

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

7.5. UEID

Used to tag a byte string as Universal Entity ID Claim (UEID). Defined in Section 4.2.1 of [RFC9711].

```
ueid-type = bytes .size (7..33)
tagged-ueid-type = #6.550(ueid-type)
```

7.6. 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)
```

7.7. Digest

A digest represents the value of a hashing operation together with the hash algorithm used. This specification reuses the digest type defined in Section 4.2 of [I-D.ietf-rats-eat-measured-component]. Only the CBOR serialization is used.

```
;# import measured-component as eatmc

digests-type = [ + eatmc.digest ]
```

A measurement can be obtained using different hash algorithms. A digests-type can be used to collect multiple digest values obtained by applying different hash algorithms on the same input. Each entry in the digests-type MUST have a unique alg value.

7.8. Tagged Bytes Type

An opaque, variable-length byte string. It can be used in different contexts: as an instance, class or group identifier in an environment-map; as a raw value measurement in a measurement-values-map. Its semantics are defined by the context in which it is found, and by the overarching CoRIM profile. When used as an identifier the responsible allocator entity SHOULD ensure uniqueness within the context that it is used.

```
tagged-bytes = #6.560(bytes)
```

8. Appraisal of CoRIM-based Inputs

Inputs to a Verifier are mapped from their external representation to an internal representation. CoRIM defines CBOR structures and content media types for Conceptual Messages that include Endorsements and Reference Values. CoRIM data structures may also be used by Evidence and Attestation Results that wish to describe overlapping structure. CoRIM-based data structures define an external representation of Conceptual Messages that are mapped to an internal representation. Appraisal processing describes both mapping transformations and Verifier reconciliation (Section 2). Non-CoRIM-based data structures require mapping transformation, but these are

out of scope for this document.

If a CoRIM profile is specified, there are a few well-defined points in the procedure where Verifier behaviour depends on the profile. The CoRIM profile MUST provide a description of the expected Verifier behavior for each of those well-defined points.

Verifier implementations MUST provide the specified information model of the Appraisal Context at the end of phase 4 as described in this specification. They are not required to use the same internal representation or evaluation order described by this specification.

8.1. Appraisal Procedure

The appraisal procedure is divided into several logical phases for clarity.

* *Phase 1*: Input Validation and Transformation

During Phase 1, all available Conceptual Messages are processed for validation. This involves checking digital signatures to verify their integrity and authenticity, ensuring they are not outdated, and confirming their relevance to the current appraisal. If validation fails, the input Conceptual Message is discarded. If validation succeeds, the input Conceptual Message is transformed from its external representation into an internal one. These internal representations are then collected in an implementation-specific "staging area", which acts as a database for subsequent appraisal processing.

* *Phase 2*: Evidence Augmentation

During Phase 2, Evidence inputs are added to a list that describes the Attester’s actual state. These inputs are added with the Attester’s authority.

* *Phase 3*: Reference Values Corroboration and Augmentation

During Phase 3, Reference Values inputs are compared with Evidence inputs. Reference Values inputs describe possible states of Attesters. If the actual state of the Attester is described by the possible Attester states, then the overlapping (corroborated) actual states are added to the Attester’s actual state. These inputs are added with the Reference Value Provider’s authority.

* *Phase 4*: Endorsed Values Augmentation

During Phase 4, Endorsed Values inputs containing conditions that describe expected Attester state are processed. If the comparison is satisfied, then additional Claims about the Attester are added to the ACS. These inputs are added with the Endorser's authority.

* *Subsequent Phases*: Before producing an Attestation Result, a Verifier may go through subsequent phases of the appraisal procedure.

For example, the Verifier may perform consistency, integrity, or additional validity checks. These checks may result in additional Claims about the Attester that are added to the ACS. These Claims are added with the Verifier's authority.

Typically, a Verifier applies Appraisal Policy for Evidence on the ACS that describes desirable or undesirable Attester states. If these conditions exist, the policy may add additional Claims about the Attester, to the ACS. These Claims are added with the policy author's authority.

Finally, the outcome of Appraisal and the set of Attester Claims of interest to a Relying Party are copied from the Attester state to an output staging area. The Claims in the output staging area and other Verifier-related metadata are transformed into an external representation, suitable for consumption by a Relying Party. This external representation is the Attestation Result message Section 8.4 of [RFC9334].

Please note, a detail description of Subsequent Phases is out of scope of this document. They are mentioned here to provide an overall context to the Appraisal procedure.

9. Reference Verifier Sequence

This document presumes that Verifier implementations will differ. To facilitate the description of normative Verifier behavior, this document describes the internal representation for a reference Verifier and demonstrates how the data is used in the appraisal phases outlined in Section 8.1. If the Verifier operates on CoRIM documents, it is RECOMMENDED that it follows this algorithm.

The terms Claim, Environment-Claim Tuple (ECT), Authority, Appraisal Claims Set (ACS), Appraisal Policy, and Attestation Results Set (ARS) are used with the meaning defined in Section 1.1.1.

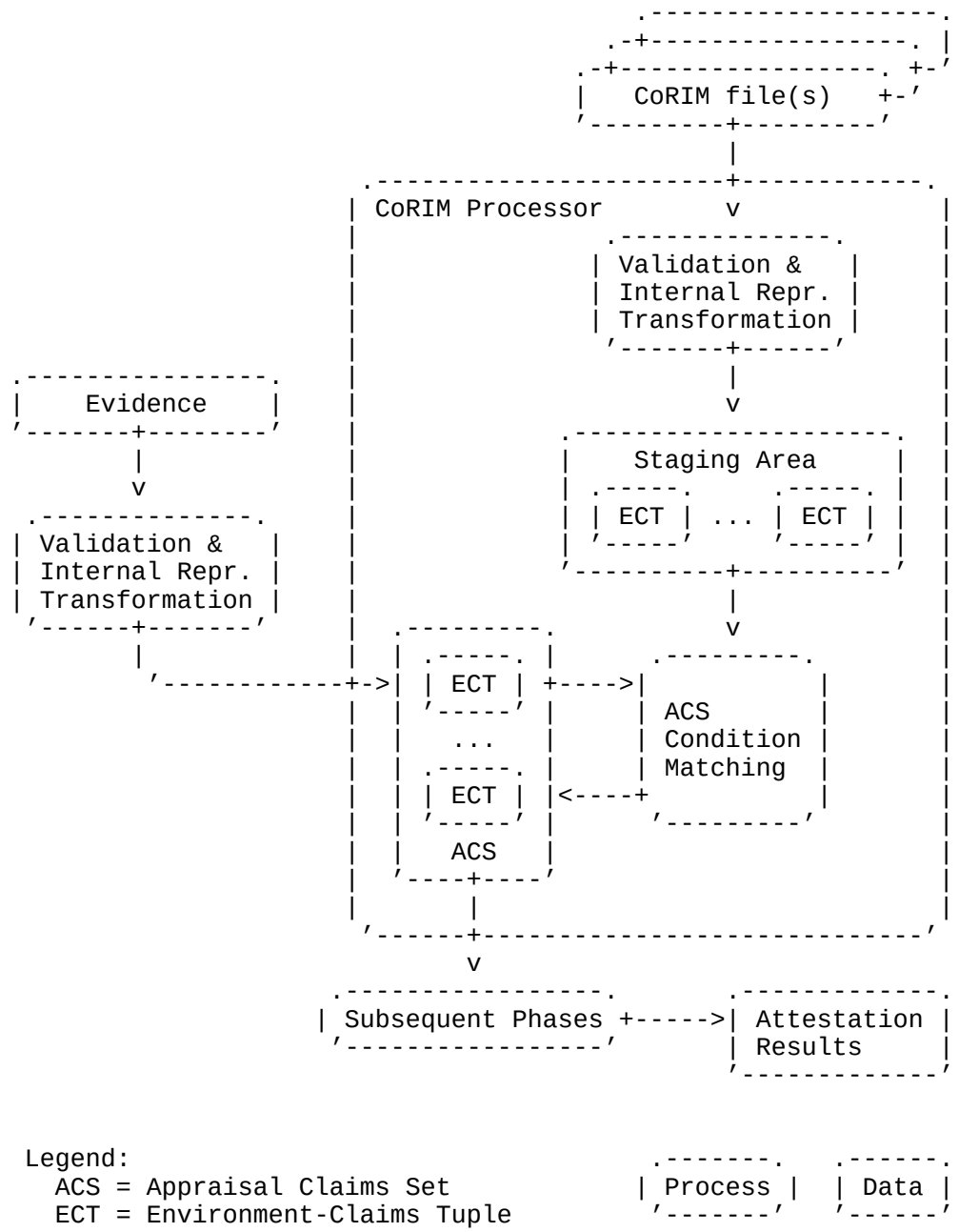

Figure 1: CoRIM Processing Flow

9.1. Processing of Conceptual Messages

Conceptual Messages are Verifier input and output values such as Evidence, Reference Values, Endorsed Values, Appraisal Policy, and Attestation Results.

All the Conceptual Messages once fully processed are transformed into a common internal representation known as Environment Claims Tuple(ECT). This section describes the internal structure of ECT and explains the process of transformation of external conceptual messages into ECTs.

9.1.1. Internal structure of ECT

Environment-Claims Tuples (ECT) have six attributes:

1. Environment : Identifies the Target Environment. Environments are identified using instance, class, or group identifiers. Environments may be composed of elements, each having an element identifier (mkey). If the Conceptual Message Type is domain-member, this field contains the domain identifier (domain-id) of the domain triple.

2. Elements : Identifies the set of elements contained within a Target Environment and their trustworthiness Claims.

3. Authority : Identifies the entity that issued the tuple. A certain type of key material by which the authority (and corresponding provenance) of the tuple can be determined, such as the public key of an asymmetric key pair that is associated with an authority’s PKIX certificate.

4. Members : Identifies the set of Environments that act as members when a Domain Membership is expressed in an ECT

5. Conceptual Message Type : Identifies the type of Conceptual Message that originated the tuple.

6. Profile : The profile that defines this tuple. If no profile is used, this attribute is omitted.

The following CDDL describes the ECT structure in more detail.

```
ECT = {
  ? environment: environment-map
  ? element-list: [ + element-map ]
  ? authority: [ + $crypto-key-type-choice ]
  ? members: [ + environment-map ]
  ? trustees: [ + environment-map ]
  ? cmtype: cm-type
  ? profile: $profile-type-choice
}

element-map = {
  ? element-id: $measured-element-type-choice
  element-claims: measurement-values-map
}

cm-type =  &(
  reference-values: 0
  endorsements: 1
  evidence: 2
  attestation-results: 3
  verifier: 4
  policy: 5
  domain-member: 6
  trustee: 7
)
```

The Conceptual Message type (cmtype) determines which attributes are mandatory.

9.1.2. Common Conventions for Input Transformation

The following mapping conventions apply to all forms of input transformation:

- The environment field is populated with a Target Environment identifier.
- The element-list field is populated with the measurements collected by an Attesting Environment.
- The authority field is populated with the identity of the entity that asserted (e.g., signed) the Conceptual Message.
- The cmtype field is set based on the type of Conceptual Message inputted or to be output.
- The profile field is set based on the corim-map profile value.

9.1.3. Processing of Evidence

9.1.3.1. Internal Representation of Evidence

An internal representation of Attestation Evidence uses the ae relation. ae implies Attestation Evidence and refers to a collection of evidence ECTs.

```
ae = [
  addition: [ + ECT ]
]
```

The addition is a list of ECTs with Evidence to be appraised.

A Verifier may maintain multiple simultaneous sessions to different Attesters. Each Attester has a different ACS. The Verifier ensures the Evidence inputs are associated with the correct ACS. The addition is added to the ACS for a specific Attester.

Table 3 contains the requirements for the ECT fields of the Evidence tuple:

ECT type	ECT Field	Requirement
addition	environment	Mandatory
	element-list	Mandatory
	authority	Mandatory
	cmtype	Mandatory
	profile	Optional
	members	n/a
	trustees	n/a

Table 3: Evidence tuple requirements

9.1.3.2. Evidence Transformation

Evidence is transformed from an external representation to an internal representation as specified in Section 9.1.3.1. The Evidence is mapped into one or more addition ECTs. If the Evidence does not have a value for the mandatory ae fields, the Verifier MUST NOT process the Evidence.

Evidence transformation algorithms may be well-known, defined by a CoRIM profile (Section 4.1.4), or supplied dynamically. The handling of Evidence transformation algorithms is out of scope for this document.

9.1.4. Processing of Reference Value Triples

Upon processing of CoRIMs containing Reference Value Triples, the RV Triples are transformed to an internal representation.

9.1.4.1. Internal Representation of Reference Values

An internal representation of Reference Values uses the rv relation, which is a list of ECTs that contains possible states and a list of ECTs that contain actual states asserted with RVP authority.

```
rv = [ + {
  condition: ECT
  addition: ECT
} ]
```

The rv relation is a list of condition-addition pairings where each pairing is evaluated together. If the condition containing reference ECTs matches Evidence ECTs then the Evidence ECTs are re-asserted, but with RVP authority as contained in the addition and cmtype set to reference-values.

The reference ECTs define the matching conditions that are applied to Evidence ECTs. If the matching condition is satisfied, then the re-asserted ECTs are added to the ACS. Refer to Section 9.3.3 for how the rv entries are processed.

Table 4 contains the requirements for the ECT fields of the Reference Values tuple:

ECT type	ECT Field	Requirement
condition	environment	Mandatory
	element-list	Mandatory
	authority	Optional
	cmtype	n/a
	profile	n/a
	members	n/a
	trustees	n/a
addition	environment	Mandatory
	element-list	Mandatory
	authority	Mandatory
	cmtype	Mandatory
	profile	Optional
	members	n/a
	trustees	n/a

Table 4: Reference Values tuple requirements

9.1.4.2. Reference Triples Transformation

Step 1. An rv list entry (Section 9.1.4.1) is allocated.

Step 2. The cmtype of the addition ECT in the rv entry is set to reference-values.

Step 3. The Reference Values Triple (RVT) (Section 5.1.5) populates the rv ECTs.

i RVT.ref-env

copy(environment-map, rv.condition.environment.environment-map)

copy(environment-map, rv.addition.environment.environment-map)

ii For each e in RVT.ref-claims:

copy(e.measurement-map, rv.condition.element-list.element-map)

Step 4. The signer of the Reference Values conceptual message is copied to the rv.addition.authority field.

Step 5. If the Reference Values conceptual message has a profile, the profile identifier is copied to the rv.addition.profile field.

9.1.5. Processing of Endorsed Value Triple, Conditional Endorsement Triple & Conditional Endorsement Series Triples

9.1.5.1. Internal Representation of Endorsed Values

An internal representation of Endorsed Values uses the ev and evs relations, which are lists of ECTs that describe matching conditions and the additions that are added if the conditions are satisfied. The ev relation is applicable to Endorsed Values (EV) and Conditional Endorsement (CE) Triple. The evs relation is applicable to the Conditional Endorsement Series Triples.

```
ev = [
  condition: [ + ECT ]
  addition: [ + ECT ]
]

evs = [
  condition: [ + ECT ]
  series: [ + {
    selection: [ + ECT ]
    addition: [ + ECT ]
  } ]
]
```

The ev relation compares the condition ECTs to the ACS and if all of the ECTs are found in the ACS then the addition ECTs are added to the ACS. Note, when ev relation is for EV Triple, then the element-list inside condition is Optional while it is Mandatory for CE Triple.

The evs relation compares the condition ECTs to the ACS and if all of the ECTs are found in the ACS then each entry in the series list is evaluated. The selection ECTs are compared with the ACS and if the selection criteria is satisfied, then the addition ECTs are added to the ACS and evaluation of the series ends. If the selection criteria is not satisfied, then evaluation procedes to the next series list entry.

Table 5 contains the requirements for the ECT fields of the Endorsed Values and Endorsed Values Series tuples:

ECT type	ECT Field	Requirement
condition	environment	Mandatory
	element-list	Mandatory
	authority	Optional
	cmtype	n/a
	profile	n/a
	members	n/a
	trustees	n/a
selection	environment	Mandatory
	element-list	Mandatory
	authority	Optional
	cmtype	n/a
	profile	n/a
	members	n/a
	trustees	n/a
addition	environment	Mandatory
	element-list	Mandatory
	authority	Mandatory

	cmtype	Mandatory
	profile	Optional
	members	n/a
	trustees	n/a

Table 5: Endorsed Values and Endorsed Values Series tuples requirements

9.1.5.2. Endorsement Triples Transformations

9.1.5.2.1. Endorsed Values Triple Transformation

Step 1. An ev entry (Section 9.1.5.1) is allocated.

Step 2. The cmtype of the ev entry’s addition ECT is set to endorsements.

Step 3. The Endorsed Values Triple (EVT) (Section 5.1.6) populates the ev ECTs.

i EVT.condition

copy(environment-map, ev.condition.environment.environment-map)

copy(environment-map, ev.addition.environment.environment-map)

ii For each e in EVT.endorsement:

copy(e.endorsement.measurement-map, ev.addition.element-list.element-map)

Step 4. The signer of the Endorsement conceptual message is copied to the ev.addition.authority field.

Step 5. If the Endorsement conceptual message has a profile, the profile is copied to the ev.addition.profile field.

9.1.5.2.2. Conditional Endorsement Triple Transformation

Step 1. An ev entry (Section 9.1.5.1) is allocated.

Step 2. The cmtype of the ev entry’s addition ECT is set to endorsements.

Step 3. Entries in the Conditional Endorsement Triple (CET) (Section 5.1.7) conditions list are copied to a suitable ECT in the internal representation.

i For each e in CET.conditions:

copy(e.stateful-environment-record.environment.environment-map, ev.condition.environment.environment-map)

copy(e.stateful-environment-record.claims-list.measurement-map, ev.condition.element-list.element-map)

ii For each e in CET.endorsements:

copy(e.endorsed-triple-record.condition.environment-map, ev.addition.environment.environment-map)

copy(e.endorsed-triple-record.endorsement.measurement-map, ev.addition.element-list.element-map)

Step 4. The signer of the Conditional Endorsement conceptual message is copied to the ev.addition.authority field.

Step 5. If the Conditional Endorsement conceptual message has a profile, the profile is copied to the ev.addition.profile field.

9.1.5.2.3. Conditional Endorsement Series Triple Transformation

Step 1. An evs entry (Section 9.1.5.1) is allocated.

Step 2. The cmtype of the evs entry’s addition ECT is set to endorsements.

Step 3. Populate the evs ECTs using the Conditional Endorsement Series Triple (CEST) (Section 5.1.8).

i. For c in CEST.condition:

copy(c.environment, evs.condition.environment)

If the c.claims-list is not empty then *copy*(c.claims-list, evs.condition.element-list)

If c.authorized-by is present then *copy*(c.authorized-by, evs.condition.authority)

ii. For each s in CEST.series:

copy(evs.condition.environment, evs.series[s].selection.environment)

copy(s.selection, evs.series[s].selection.element-list)

copy(evs.condition.environment, evs.series[s].addition.environment)

copy(s.addition, evs.series[s].addition.element-list)

Step 4. The signer of the Conditional Endorsement Series conceptual message is copied to the evs.series.addition.authority field for each addition.

Step 5. If the Conditional Endorsement Series conceptual message has a profile, the profile is copied to the evs.series.addition.profile field for each addition.

9.1.6. Processing of Attest Key and Device Identity Key Triples

9.1.6.1. Internal Representation of Cryptographic Keys

The internal representation for keys use the extension slot within measurement-values-map with the intrep-keys claim that consists of a list of typed-crypto-key. typed-crypto-key consists of a key and an optional key-type. There are two types of keys attest-key and identity-key.

```
$$measurement-values-map-extension //= (
  &(intrep-keys: 65534) => [ + typed-crypto-key ]
)

typed-crypto-key = {
  key: $crypto-key-type-choice
  ? key-type: uint .bits key-type
}

key-type = &(
  attest-key: 0
  identity-key: 1
)
```

9.1.6.2. Key Verification Triples Transformation

The following transformation steps are applied for both the identity-triples and attest-key-triples with noted exceptions:

Step 1. An ev entry (Section 9.1.5.1) is allocated.

Step 2. The cmtype of the ev entry’s addition ECT is set to endorsements.

Step 3. Populate the ev condition ECT using either the identity-triple-record or attest-key-triple-record (Section 5.1.9) as follows:

i. *copy*(environment-map, ev.condition.environment.environment-map).

ii. Foreach _key_ in keylist.$crypto-key-type-choice, *copy*(_key_, ev.condition.element-list.element-map.element-claims.measurement-values-map.intrep-keys.key).

iii. If key-list originated from attest-key-triples, *set*(ev.condition.element-list.element-map.element-claims.measurement-values-map.intrep-keys.key-type = attest-key).

iv. Else if key-list originated from identity-triples, *set*(ev.condition.element-list.element-map.element-claims.measurement-values-map.intrep-keys.key-type = identity-key).

v. If populated, *copy*(mkey, ev.condition.element-list.element-map.element-id).

vi. If populated, *copy*(authorized-by, ev.condition.authority).

Step 4. The signer of the Identity or Attest Key Endorsement conceptual message is copied to the ev.addition.authority field.

Step 5. If the Endorsement conceptual message has a profile, the profile is copied to the ev.addition.profile field.

9.1.7. Processing of Domain Membership Triples

9.1.7.1. Internal Representation of Domain Membership

An internal representation of Domain Membership is expressed in a single ECT, where the domain identifier is set in the environment field of the ECT, and the domain members are expressed in the members field. The cmtype is set to domain-member.

```
dm = [ + ( domain: ECT ) ]
```

Table 6 contains the requirements for the ECT fields of the Domain Membership tuple:

ECT type	ECT Field	Requirement
domain	environment	Mandatory
	element-list	Optional
	authority	Mandatory
	cmtype	Mandatory
	profile	Optional
	members	Mandatory
	trustees	Optional

Table 6: Domain Membership tuple requirements

9.1.7.2. Domain Membership Triples Transformation

This section describes how the external representation of a Domain Membership Triple (DMT) (Section 5.1.11.1) is transformed into its CoRIM internal representation dm (see Section 9.1.7.1).

Step 1. Allocate a domain ECT entry.

Step 2. Set the conceptual message type for the domain ECT to 6 (domain-member).

i *copy*(domain-member, domain.cmtype)

Step 3. Set the authority for the domain ECT to the DMT signer (Section 4.2.3.1).

i *copy*(DMT.signer, domain.authority)

Step 4. Use the DMT to populate the dm internal representation.

i *copy*(DMT.domain-id, domain.environment)

ii For each environment *e* in DMT.members:

copy(DMT.members[_e_].environment,
domain.members[_e_].environment)

Step 5. If the conceptual message containing the DMT has a profile, it is used to populate the profile for the domain ECT.

i *copy*(DMT.profile, domain.profile)

9.1.8. Processing of Domain Dependency Triples

9.1.8.1. Internal Representation of Domain Dependency

An internal representation of trust dependency is a directed acyclic graph where each node in the graph identifies a member domain and contains edges to dependent, or "trustee" domains. An ECT structure environment field contains the domain identifier, and the ECT trustees list are the edges. The cmtype is inclusive of trustee to indicate the ECT is being used to model a trust dependency graph.

ddg = [+ (dde: ECT)]

Table 7 contains the requirements for the ECT fields of the Domain Dependency tuple:

ECT type	ECT Field	Requirement
domain	environment	Mandatory
	element-list	Optional
	authority	Mandatory
	cmtype	Mandatory
	profile	Optional
	members	Mandatory
	trustees	Mandatory

Table 7: Domain Dependency tuple requirements

9.1.8.2. Domain Dependency Triples Transformation

This section describes how the external representation of a Domain Dependency Triple (DDT) (Section 5.1.11.2) is transformed into its CoRIM internal representation of a domain dependency graph (ddg) (see Section 9.1.8.1).

For each domain-dependency-triple-record (ddtr) in the DDT list, perform the following steps:

Step 1. Allocate a domain dependency edge dde ECT entry.

Step 2. Set the conceptual message type cmtype for the dde ECT to trustee.

i *assign*(trustee, dde.cmtype)

Step 3. Set the authority for the trust domain ECT to the ddt signer (Section 4.2.3.1).

i *copy*(ddtr.signer, dde.authority)

Step 4. Populate the ECT environment using the domain identifier.

i *copy*(ddtr.domain-id, dde.environment)

Step 5. Populate the ECT trustees.

i For each environment _e_ in ddtr.trustees:

copy([_e_].environment, dde.trustees[_e_].environment)

Step 6. If the conceptual message containing the DDT has a profile, it is used to populate the profile for the dde ECT.

i *copy*(ddtr.profile, dde.profile)

Append the domain dependency edge (dde) to the domain dependency graph (ddg).

Process each domain dependency triple record (ddtr) in the DDT list until every entry has been transformed to the internal representation and is contained in the domain dependency graph.

The domain dependency graph becomes input to domain dependency processing steps in Section 9.3.4.6.

9.2. Input Validation and Transformation (Phase 1)

During the initialization phase, the CoRIM Appraisal Context is loaded with various conceptual message inputs such as CoMID tags (Section 5), CoSWID tags [RFC9393], CoTL tags, and cryptographic validation key material (including raw public keys, root certificates, intermediate CA certificate chains), and Concise Trust Anchor Stores (CoTS) [I-D.ietf-rats-concise-ta-stores]. These objects will be utilized in the Evidence Appraisal phase that follows. The primary goal of this phase is to ensure that all necessary information is available for subsequent processing.

After context initialization, additional inputs are held back until appraisal processing has completed.

9.2.1. Input Validation

9.2.1.1. CoRIM Selection

All available CoRIMs are collected.

CoRIM tags MUST be discarded if they are expired, or if they are not associated with an authenticated and authorized source, or if they have been revoked by an authorized source.

Any CoRIM that has been secured by a cryptographic mechanism that fails validation MUST be discarded. An example of such a mechanism is a digital signature.

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.

Later stages will further select the CoRIMs appropriate to the Evidence Appraisal stage.

9.2.1.2. CoRIM Trust Anchors

If CoRIM tags are signed, the signatures MUST be validated using the appropriate trust anchors (certification paths) available to the Verifier. The Verifier is expected to have a trust anchor store. The way in which these trust anchors (i.e., root certificates) are provisioned in the Verifier is beyond the scope of this specification. If signed, the CoRIM itself should include at least one certificate (e.g., as part of the x5chain in the COSE header), which corresponds to the key pair used for signing. This certificate (the "leaf certificate") must be linked to one of the Verifier trust anchors.

9.2.1.3. Tags Extraction and Validation

The Verifier chooses tags from the selected CoRIMs - including CoMID, CoSWID, CoTL, and CoTS.

The Verifier MUST discard all tags which are not syntactically and semantically valid. Cross-referenced triples MUST be successfully resolved. An example of a cross-referenced triple is a CoMID-CoSWID linking triple.

9.2.1.4. CoTL Extraction

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

CoTLs which are not within their validity period MUST be discarded.

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

The Verifier MAY decide to discard some of the available and valid CoTLs 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 the 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 a case, the Verifier Owner may instruct the Verifier to discard tags activated by supplier CoTLs that are not also activated by the trusted integrator.

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

9.2.2. Evidence Collection

During the Evidence collection phase, the Verifier communicates with Attesters to gather Evidence. Discovery of Evidence sources is untrusted. Verifiers may rely on conveyance protocol specific context to identify an Evidence source, which is the Evidence input oracle for appraisal.

The collected Evidence is then transformed to an internal representation, making it suitable for appraisal processing.

The exact protocol used to collect Evidence is out of scope of this specification.

9.2.2.1. Cryptographic Validation of Evidence

If Evidence is cryptographically signed, its validation is applied before transforming Evidence to an internal representation.

If Evidence is not cryptographically signed, the underlying conveyance protocol that collected it, should provide the required security. In such cases, the cryptographic validation of Evidence is determined by the security offered by the conveyance protocol.

The way cryptographic signature validation works depends on the specific Evidence collection method used. For example, in DICE, a proof of liveness is carried out on the final key in the certificate chain (a.k.a., the alias certificate). If this is successful, a suitable certification path is looked up in the Appraisal Context, based on linking information obtained from the DeviceID certificate. See Section 9.2.1 of [DICE.Layer]. If a trusted root certificate is found, X.509 certificate validation is performed.

As a second example, in PSA [RFC9783] the verification public key is looked up in the appraisal context using the ueid claim found in the PSA claims-set. If found, COSE Sign1 verification is performed accordingly.

Regardless of the specific integrity protection method used, the Verifier MUST NOT process Evidence which is not successfully validated.

Once Evidence is validated it is transformed into an internal representation as given in Section 9.1.3.

9.2.3. Input Transformation

Input Conceptual Messages, whether Evidence, Reference Values, Endorsements, or Policies, are transformed to an internal representation that is based on ECTs (Section 9.1).

9.2.3.1. Appraisal Context Construction

All of the extracted and validated tags are loaded into an _appraisal context_. The Appraisal Context contains an internal representation of the inputted Conceptual Messages. The selected tags are mapped to an internal representation, making them suitable for appraisal processing. As the Conceptual Messages are processed during Appraisal, the _appraisal context_ starts to get populated using a data structure as given below. See Section 9.2.4

9.2.4. Internal Representation of Appraisal Claims Set (ACS)

An ACS is a list of ECTs that describe an Attester’s actual state.

For ECTs present in the ACS, the cmtype field is mandatory. Table Table 8 shows the minimum required mandatory fields applicable to all ECTs in an ACS.

ECT type	ECT Field	Requirement
n/a	environment	Mandatory
	authority	Mandatory
	cmtype	Mandatory

Table 8: ACS tuple minimum requirements

ACS = [+ ECT]

9.3. ACS Augmentation - Phases 2, 3, and 4

In the ACS augmentation phase, a CoRIM Appraisal Context and an Evidence Appraisal Policy are used by the Verifier to find CoMID triples which match the ACS. Triples that specify an ACS matching condition will augment the ACS with Endorsements if the condition is met.

Each triple is processed independently of other triples. However, the ACS state may change as a result of processing a triple. If a triple condition does not match, then the Verifier continues to process other triples.

9.3.1. ACS Requirements

At the end of the Evidence collection process, the Evidence has been converted into an internal representation suitable for appraisal. See Section 9.1.

Verifiers are not required to use this as their internal representation. For the purposes of this document, appraisal is described in terms of the above cited internal representation.

9.3.1.1. ACS Processing Requirements

The ACS contains the actual state of Attester’s Target Environments (TEs). The ACS contains Evidence ECTs (from Attesters) and Endorsement ECTs (e.g. from endorsed-triple-record).

CoMID Reference Values will be matched against the ACS following the comparison rules in Section 9.4.

Each Endorsement ECT contains the environment and internal representation of measurement-maps as extracted from an endorsed-triple-record. When an endorsed-triple-record is transformed to Endorsements ECTs 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.

ECT 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 cryptographic key.

Each Claim is encoded as an ECT. The environment-map, the mkey or element-id, 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 ECT. There can be multiple entries in state-triples which have the same environment-map and a different authority. See Section 9.3.2.2.

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

9.3.1.1.1. Ordering of triple processing

Triples interface with the ACS by either adding new ACS entries or by matching existing ACS entries before updating the ACS. Most triples use an environment-map field to select the ACS entries to match or modify. This field may be contained in an explicit matching condition, such as stateful-environment-record.

The order of triples processing is important. Processing a triple may result in ACS modifications that affect matching behavior of other triples.

The Verifier MUST ensure that a triple including a matching condition is processed after any other triple that modifies or adds an ACS entry with an environment-map that is in the matching condition.

This can be acheived by sorting the triples before processing, by repeating processing of some triples after ACS modifications or by other algorithms.

9.3.1.2. ACS Augmentation Requirements

The ordering of ECTs in the ACS is not significant. Logically, the ACS represents the conjunction of all claims, so adding an ECT entry to the existing ACS at the end is equivalent to inserting it anywhere else. Implementations may optimize ECT order to achieve better performance. Additions to the ACS MUST be atomic.

9.3.2. Evidence Augmentation (Phase 2)

9.3.2.1. Appraisal Claims Set Initialization

The ACS is initialized by copying the internal representation of Evidence claims to the ACS. See Section 9.3.

9.3.2.2. The authority field in the ACS

The authority field in an ACS ECT indicates the entity whose authority backs the Claims.

The Verifier keeps track of authority so that it can satisfy appraisal policy that specifies authority.

When adding an Evidence entry to the ACS, the Verifier SHALL set the authority field using a $crypto-keys-type-choice representation of the entity that signed the Evidence.

If multiple authorities approve the same Claim, for example if multiple key chains are available, then the authority field SHALL be set to include the $crypto-keys-type-choice representation for each key chain.

When adding Endorsement or Reference Values Claims to the ACS that resulted from CoRIM processing, the Verifier SHALL set the authority field using a $crypto-keys-type-choice representation of the entity that signed the CoRIM.

When searching the ACS for an entry which matches a triple condition containing an authorized-by field, the Verifier SHALL ignore ACS entries if none of the entries present in the condition authorized-by field are present in the ACS authority field. The Verifier SHALL match ACS entries if all of the entries present in the condition authorized-by field are present in the ACS authority field.

9.3.2.3. ACS augmentation using CoMID triples

In the ACS augmentation phase, a CoRIM Appraisal Context and an Evidence Appraisal Policy are used by the Verifier to find CoMID triples which match the ACS. Triples that specify an ACS matching condition will augment the ACS with Endorsements if the condition is met.

Each triple is processed independently of other triples. However, the ACS state may change as a result of processing a triple. If a triple condition does not match, then the Verifier continues to process other triples.

9.3.3. Reference Values Corroboration and Augmentation (Phase 3)

Reference Value Providers (RVP) publish Reference Values using the Reference Values Triple (Section 5.1.5) which are transformed (Section 9.1.4.2) into an internal representation (Section 9.1.4.1). Each Reference Value Triple describes a single possible Attester state.

Corroboration is the process of determining whether actual Attester state (as contained in the ACS) can be satisfied by Reference Values.

Reference Values are matched with ACS entries by iterating through the rv list. For each rv entry, the condition ECT is compared with an ACS ECT, where the ACS ECT cmtype contains evidence.

If satisfied, for the rv entry, the following three steps are performed:

1. The addition ECT is moved to the ACS, with cmtype set to reference-values

2. The claims, i.e., the element-list from the ACS ECT with cmtype set to evidence is copied to the element-list of the addition ECT

3. The authority field of the addition ECT has been confirmed as being set correctly to the RVP authority

9.3.4. Endorsed Values Augmentation (Phase 4)

Endorsers publish Endorsements using endorsement triples (see Section 5.1.6), Section 5.1.7, and Section 5.1.8) which are transformed (Section 9.1.5.2) into an internal representation (Section 9.1.5.1). Endorsements describe actual Attester state. Endorsements are added to the ACS if the Endorsement condition is satisifed by the ACS.

9.3.4.1. Processing Endorsements

Endorsed Values Triple and Conditional Endorsement Triple share the same internal representation.

After transformation into an ev entry, the processing steps of both triples are the same, as described below. Each ev entry is processed independently of other evs.

Endorsements are matched with ACS entries by iterating through the ev list. For each ev entry, the condition ECT is compared with an ACS ECT, where the ACS ECT cmtype contains either evidence, reference-values, or endorsements. If the ECTs match (Section 9.4), the ev addition ECT is added to the ACS.

Some condition values can match against multiple ACS-ECTs, or sets of ACS-ECTs. If there are multiple matches, then each match is processed independently from the others.

9.3.4.2. Processing Conditional Endorsements

Conditional Endorsement Triples are transformed into an internal representation based on ev. Conditional endorsements have the same processing steps as shown in (Section 9.3.4.1).

9.3.4.3. Processing Conditional Endorsement Series

Conditional Endorsement Series Triples are transformed into an internal representation based on evs. Conditional series endorsements are matched with ACS entries first by iterating through the evs list, where for each evs entry, the condition ECT is compared with an ACS ECT, where the ACS ECT cmtype contains either evidence, reference-values, or endorsements. If the ECTs match (Section 9.4), the evs series array is iterated, where for each series entry, if the selection ECT matches an ACS ECT, the addition ECT is added to the ACS. Series iteration terminates after the first matching series entry is processed or when no series entries match.

9.3.4.4. Processing Key Verifications

To process key verification triples, the internal representation of ECTs containing intrep-keys is used to identify ACS entries containing $crypto-key-type-choice values that require additional key verification steps. If the key-type field is set, the Verifier will apply the verification steps defined below. If the key verification check succeeds, the key is re-asserted by the Verifier as an Endorsement by constructing an ECT that contains the verified key using the authority of the Verifier. Note that, in this case, the

Verifier is acting in the role of an Endorser.

For each ECT from endorsed value (ev) or attestation evidence (ae) entries, the candidate ECT (C-ECT) is compared with an ACS ECT (ACS-ECT), where the ACS-ECT cmtype contains either evidence or endorsements. If the C-ECT and ACS-ECT match (Section 9.4), then for each _key_ in the C-ECT.element-claims.measurement-values-map.intrep-keys, do the following steps:

Step 1. Verify the certificate signatures for each certificate in the certification path.

Step 2. Verify certificate revocation status for each certificate in the certification path.

Step 3. Verify key usage restrictions appropriate for the type of key in key-type.

Step 4. If key verification succeeds for any _key_, allocate an addition ECT (ADDITION).

Step 5. For each verified _key_ in C-ECT:

i *append*(_key_, ADDITION.element-list.element-map.element-claims.measurement-values-map.intrep-keys).

Step 6. *copy*(ACS-ECT.environment, ADDITION.environment‘).

Step 7. *copy*(ACS-ECT.element-list.element-map.element-id, ADDITION.element-list.element-map.element-id).

Step 8. Set ADDITION.cmtype to endorsements.

Step 9. Add the Verifier authority $crypto-key-type-choice to the ADDITION.authority field.

Step 10. Add the ADDITION to the ACS.

Otherwise, do not add the ADDITION to the ACS.

It is possible that a candidate key has been verified during Phase 1 processing (Section 9.2) or is replicated across Evidence or Endorsement ECTs. Implementations might optimize processing of key verifications by checking whether a key has already been verified by the Verifier.

9.3.4.5. Processing Domain Membership

This section assumes that each Domain Membership Triple (see Section 5.1.11.1) has been transformed into an internal representation following the steps described in Section 9.1.7.2, resulting in the representation specified in Section 9.1.7.1.

Domain Membership ECTs (i.e., cmtype equals domain-member) in the dm staging area are matched with ACS entries where cmtype is set to evidence, reference-values i.e. corroborated evidence, or domain-member using the following algorithm:

For each domain in the dm staging area, which has not been processed (outer loop):

For each member m in domain.members (inner loop):

* Check that there is a corresponding ACS entry environment that matches m.environment.

* Check that the ACS entry cmtype is one of evidence, reference-values, or domain-member.

Outer loop resumes:

* If all domain.members matched a corresponding ACS entry, add the domain ECT to the ACS.

* If none of the domain.members matched, proceed to next dm entry.

* If some, but not all of the domain.members matched, proceed to the next dm entry. If the previous execution of the outer loop added any domain entry to the ACS, then run the outer loop again Else STOP processing dm entries.

The processing terminates, when all the Domain Membership ECTs which are appropriate to the Evidence have been added to the ACS.

If any of the expected Domain Membership ECTs have not been added to the ACS, then this may affect outcomes in subsequent phases.

9.3.4.6. Processing Domain Dependency

This section assumes that each Domain Dependency Triple (see Section 5.1.11.2) has been transformed into domain dependency graph (see Section 9.1.8.1) following the steps described in Section 9.1.8.2.

Processing a domain dependency graph (DDG) has the following objectives:

* Verify each edge in a DDG has a corresponding edge in a domain membership graph. DDGs need not be isomorphic to domain membership graphs.

* Verify the DDG is acyclic.

If, in a later processing phase, an appraisal policy for trust dependency exists, the DDG can be further evaluated. For example, a trust dependency policy might specify a strength of function requirement for how Evidence about a TE is integrity protected by its AE.

Domain Dependency ECTs are processed using the following algorithm:

For each dde in the ddg staging array (outer loop):

* Check that the ACS.cmtype contains domain-member.

* Check that the dde.environment matches a domain member ACS entry environment.

* OR that the dde.environment matches one of it’s ACS.members.environment.

For each trustee _t_ in dde.trustees (inner loop):

* Check that the ACS.cmtype contains domain-member.

* Check that _t_ matches an ACS.environment.

Outer loop resumes:

* If the dde.environment record AND all dde.trustees matched an ACS with cmtype domain-member entry. Then add the dde to the ACS.

* Continue to the next dde until all are processed.

Subsequent Verifier stages or Relying Party processing of the ACS can be impacted when Domain Dependency ECTs are not added to the ACS. For example, trust in an ACS entry that depends on trustee ACS entries might not be considered.

9.4. Comparing a condition ECT against the ACS

The Verifier SHALL iterate over all ACS entries and SHALL attempt to match the condition ECT against each ACS entry. See Section 9.4.1. The Verifier SHALL create a "matched entries" set, and SHALL populate it with all ACS entries which matched the condition ECT.

If the matched entries array is not empty, then the condition ECT matches the ACS.

If the matched entries array is empty, then the condition ECT does not match the ACS.

9.4.1. Comparing a condition ECT against a single ACS entry

If the condition ECT contains a profile and the profile defines an algorithm for a codepoint and environment-map then the Verifier MUST use the algorithm defined by the profile, or it MUST use a standard algorithm if the profile defines that. If the condition ECT contains a profile, but the profile does not define an algorithm for a particular codepoint and environment-map then the verifier MUST use the standard algorithm described in this document to compare the data at that codepoint.

The Verifier SHALL perform all of the comparisons defined in Section 9.4.2, Section 9.4.3, and Section 9.4.4.

Each of these comparisons compares one field in the condition ECT against the same field in the ACS entry.

If all of the fields match, then the condition ECT matches the ACS entry.

If any of the fields does not match, then the condition ECT does not match the ACS entry.

9.4.2. Environment Comparison

The Verifier SHALL compare each field which is present in the condition ECT environment-map against the corresponding field in the ACS entry environment-map using binary comparison. Before performing the binary comparison, the Verifier SHOULD convert both environment-map fields into a form which meets CBOR Core Deterministic Encoding Requirements [STD94].

If all fields which are present in the condition ECT environment-map (e.g., instance-id or group-id) are also present in the ACS entry and are binary identical, then the environments match.

If any field which is present in the condition ECT environment-map is not present in the ACS entry, then the environments do not match.

If any field which is present in the condition ECT environment-map is not binary identical to the corresponding ACS entry field, then the environments do not match.

If a field is not present in the condition ECT environment-map then the presence of, and value of, the corresponding ACS entry field SHALL NOT affect whether the environments match.

9.4.3. Authority comparison

The Verifier MUST compare byte-by-byte the condition ECT’s authority value to the candidate entry’s authority value.

If every entry in the condition ECT authority matches byte-by-byte a corresponding entry in the ACS entry authority field, then the authorities match. The order of the fields in each authority field do not affect the result of the comparison.

If any entry in the condition ECT authority does not have a matching entry in the ACS entry authority field then the authorities do not match.

When comparing two $crypto-key-type-choice fields for equality, the Verifier SHALL treat them as equal if their deterministic CBOR encoding is binary equal.

9.4.4. Element list comparison

The Verifier SHALL iterate over all the entries in the condition ECT element-list and compare each one against the corresponding entry in the ACS entry element-list.

If every entry in the condition ECT element-list has a matching entry in the ACS entry element-list field then the element lists match.

The order of the fields in each element-list field do not affect the result of the comparison.

If any entry in the condition ECT element-list does not have a matching entry in the ACS entry element-list field then the element-list do not match.

9.4.5. Element map comparison

The Verifier SHALL compare each element-map within the condition ECT element-list against the ACS entry element-list.

First, the Verifier SHALL locate the entries in the ACS entry element-list which have a matching element-id as the condition ECT element-map. Two element-id fields are the same if they are either both omitted, or both present with binary identical deterministic encodings.

Before performing the binary comparison, the Verifier SHOULD convert both fields into a form which meets CBOR Core Deterministic Encoding Requirements [STD94].

If any condition ECT entry element-id does not have a corresponding element-id in the ACS entry then the element map does not match.

If any condition ECT entry has multiple corresponding element-ids then the element map does not match.

Second, the Verifier SHALL compare the element-claims field within the condition ECT element-list and the corresponding field from the ACS entry. See Section 9.4.6.

9.4.6. Measurement values map map Comparison

The Verifier SHALL iterate over the codepoints which are present in the condition ECT element's measurement-values-map. Each of the codepoints present in the condition ECT measurement-values-map is compared against the same codepoint in the candidate entry measurement-values-map.

If any codepoint present in the condition ECT measurement-values-map does not have a corresponding codepoint within the candidate entry measurement-values-map then Verifier SHALL remove that candidate entry from the candidate entries array.

If any codepoint present in the condition ECT measurement-values-map does not match the same codepoint within the candidate entry measurement-values-map then Verifier SHALL remove that candidate entry from the candidate entries array.

9.4.6.1. Comparison of a single measurement-values-map codepoint

The Verifier SHALL compare each condition ECT measurement-values-map value against the corresponding ACS entry value using the appropriate algorithm.

Non-negative codepoints represent standard data representations. The comparison algorithms for these are defined in this document (in the sections below) or in other specifications. For some non-negative codepoints their behavior is modified by the CBOR tag at the start of the condition ECT measurement-values-map value.

Negative codepoints represent profile defined data representations. The Verifier SHALL use the codepoint number, the profile associated with the condition ECT, and, if present, the tag value to select the comparison algorithm.

If the Verifier is unable to determine the comparison algorithm which applies to a codepoint then it SHALL behave as though the candidate entry does not match the condition ECT.

Profile writers SHOULD use CBOR tags for widely applicable comparison methods to ease Verifier implementation compliance across profiles.

The following subsections define the comparison algorithms for the measurement-values-map codepoints defined by this specification.

9.4.6.1.1. Comparison for version entries

The value stored under measurement-values-map codepoint 0 is an version label, which MUST have type version-map. Two version-map values can only be compared for equality, as they are colloquial versions that cannot specify ordering.

9.4.6.1.2. Comparison for svn entries

The ACS entry value stored under measurement-values-map codepoint 1 is a security version number, which MUST have type svn-type.

If the entry svn-type is a uint or a uint tagged with #6.552, then comparison with the uint named as SVN is as follows.

* If the condition ECT value for measurement-values-map codepoint 1 is an untagged uint or a uint tagged with #6.552 then an equality comparison is performed on the uint components. The comparison MUST return true if the value of SVN is equal to the uint value in the condition ECT.

* If the condition ECT value for measurement-values-map codepoint 1 is a uint tagged with #6.553 then a minimum comparison is performed. The comparison MUST return true if the uint value in the condition ECT is less than or equal to the value of SVN.

If the entry svn-type is a uint tagged with #6.553, then comparison with the uint named as MINSVN is as follows.

* If the condition ECT value for measurement-values-map codepoint 1 is an untagged uint or a uint tagged with #6.552 then the comparison MUST return false.

* If the condition ECT value for measurement-values-map codepoint 1 is a uint tagged with #6.553 then an equality comparison is performed. The comparison MUST return true if the value of MINSVN is equal to the uint value in the condition ECT.

The meaning of a minimum SVN as an entry value is only meaningful as an endorsed value that has been added to the ACS. The condition therefore treats the minimum SVN as an exact state and not one to compare with inequality.

9.4.6.1.3. Comparison for digests entries

A digests entry contains one or more digests, each measuring the same object. When multiple digests are provided, each represents a different algorithm acceptable to the condition ECT author.

In the simple case, a condition ECT digests entry containing one digest matches a candidate entry containing a single entry with the same algorithm and value.

If there are multiple algorithms in common between the condition ECT and candidate entry, then the bytes paired with common algorithms MUST be equal. This is to prevent downgrade attacks. The Verifier SHALL treat two algorithm identifiers as equal if they have the same deterministic binary encoding. If both an integer and a string representation are defined for an algorithm then entities creating ECTs SHOULD use the integer representation. If condition ECT and ACS entry use different names for the same algorithm, and the Verifier does not recognize that they are the same, then a downgrade attack is possible.

The comparison MUST return false if the CBOR encoding of the digests entry in the condition ECT or the ACS value with the same codepoint is incorrect. For example, if fields are missing or if they are the wrong type.

The comparison MUST return false if the condition ECT digests entry does not contain any digests.

The comparison MUST return false if either digests entry contains multiple values for the same hash algorithm.

The Verifier MUST iterate over the condition ECT digests array, locating the common hash algorithm identifiers which are present in both the condition ECT and in the candidate entry. If the value associated with any common hash algorithm identifier in the condition ECT differs from the value for the same algorithm identifier in the candidate entry then the comparison MUST return false.

The comparison MUST return false if there are no hash algorithms from the condition ECT in common with the hash algorithms from the candidate entry ECT.

9.4.6.1.4. Comparison for raw-value entries

A raw-value entry contains binary data.

The value stored under measurement-values-map codepoint 4 in an ACS entry MUST be a raw-value entry, which MUST be tagged and have type bytes.

The value stored under the condition ECT measurement-values-map codepoint 4 may additionally be a tagged-masked-raw-value entry, which specifies an expected value and a mask.

If the condition ECT measurement-value-map codepoint 4 is of tagged-bytes, and there is no value stored under codepoint 5, then the Verifier treats it in the same way as a tagged-masked-raw-value with the value field holding the same contents and a mask of the same length as the value with all bits set. The standard comparison function defined in this document removes the tag before performing the comparison.

For backwards compatibility, if the condition ECT measurement-value-map codepoint 4 is of type tagged-bytes, and there is a mask stored under codepoint 5, then the Verifier treats it in the same way as a tagged-masked-raw-value with the value field holding the same contents and a mask holding the contents of codepoint 5.

The comparison MUST return false if the lengths of the candidate entry value and the condition ECT value are different.

The comparison MUST return false if the lengths of the condition ECT mask and value are different.

The comparison MUST use the mask to determine which bits to compare. If a bit in the mask is 0 then this indicates that the corresponding bit in the ACS Entry value may have either value. If, for every bit position in the mask whose value is 1, the corresponding bits in both values are equal then the comparison MUST return true.

9.4.6.1.5. Comparison for cryptokeys entries

The CBOR tag of the first entry of the condition ECT cryptokeys array is compared with the CBOR tag of the first entry of the candidate entry cryptokeys value. If the CBOR tags match, then the bytes following the CBOR tag from the condition ECT entry are compared byte-by-byte with the bytes following the CBOR tag from the candidate entry. If the byte strings match and there is another array entry, then the next entry from the condition ECTs array is likewise compared with the next entry of the ACS array. If all entries of the condition ECTs array match a corresponding entry in the ACS array, then the cryptokeys condition ECT matches. Otherwise, cryptokeys does not match.

9.4.6.1.6. Comparison for Integrity Registers

For each Integrity Register entry in the condition ECT, the Verifier will use the associated identifier (i.e., integrity-register-id-type-choice) to look up the matching Integrity Register entry in the candidate entry. If no entry is found, the comparison MUST return false. Instead, if an entry is found, the digest comparison proceeds as defined in Section 9.4.6.1.3 after equivalence has been found according to Section 5.1.4.7. Note that it is not required for all the entries in the candidate entry to be used during matching: the condition ECT could consist of a subset of the device’s register space. In TPM parlance, a TPM "quote" may report all PCRs in Evidence, while a condition ECT could describe a subset of PCRs.

9.4.6.1.7. Comparison for int-range entries

The ACS entry value stored under measurement-values-map codepoint 15 is an int range value, which MUST have type int-range-type-choice.

Consider an int ACS entry value named ENTRY in a measurement-values-map codepoint (e.g., 15) that allows comparing int against a either another int or an int-range named CONDITION.

* If CONDITION is an int then an equality comparison is performed with ENTRY.

* If CONDITION is an int-range (CBOR tag 564), then a range inclusion comparison is performed. The comparison MUST return true if and only if all the following conditions are true:

  - CONDITION.min is null or ENTRY is greater than or equal to CONDITION.min.

- CONDITION.max is null or ENTRY is less than or equal to CONDITION.max.

Consider an int-range or int-range (CBOR tag 564) value named ENTRY in a measurement-values-map codepoint (e.g., 15) that allows comparing an int-range against either another int-range or an int named CONDITION.

* If CONDITION is an int, then the comparison MUST return true if and only if ENTRY.min and ENTRY.max are both equal to CONDITION.

* If CONDITION is an int-range (CBOR tag 564), then a range subsumption comparison is performed (i.e., the condition range includes all values of the entry range). The comparison MUST return true if and only if all the following conditions are true:

  - CONDITION.min is null or ENTRY.min is an int that is greater than or equal to CONDITION.min

  - CONDITION.max is null or ENTRY.max is an int that is less than or equal to CONDITION.max.

9.4.7. Profile-directed Comparison

A profile MUST specify comparison algorithms for its additions to $-prefixed CoRIM CDDL codepoints when this specification does not prescribe binary comparison. The profile MUST specify how to compare the CBOR tagged Reference Value against the ACS.

Note that the Verifier may compare Reference Values in any order, so the comparison SHOULD NOT be stateful.

10. 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".

10.1. Veraison

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

* Implementation’s web page: https://github.com/veraison/corim/README.md (https://github.com/veraison/corim/blob/main/README.md)

* Brief general description: There are three CoRIM libraries under project Veraison.

  1. CoRIM golang library (https://github.com/veraison/corim) 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.

  2. CoRIM rust library (https://github.com/veraison/corim-rs) provide a rust implementation of CoRIM specification.

  3. CoRIM Processor (https://github.com/veraison/cover) provides a library for appraisal of Evidence by processing CoRIMs as outlined in Section 9} of this specification.

  In addition to the base CoRIM Libraries, the cocli package (https://github.com/veraison/cocli) uses the golang 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/veraison/cocli/README.md (https://github.com/veraison/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 -06 of the draft

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

* Implementation experience: n/a

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

* Last updated: https://github.com/veraison/corim/commits/main (https://github.com/veraison/corim/commits/main) https://github.com/veraison/corim-rs/commits/master/ (https://github.com/veraison/corim-rs/commits/master/) https://github.com/veraison/cover/commits/main/ (https://github.com/veraison/cover/commits/main/)

11. Security and Privacy Considerations

Evidence appraisal is at the core of any RATS protocol flow, mediating all interactions between Attesters and their Relying Parties. The Verifier is effectively part of the Attesters' and Relying Parties' trusted computing base (TCB). Any mistake in the appraisal procedure conducted by the Verifier could have security implications. For instance, it could lead to the subversion of an access control function, which creates a chance for privilege escalation.

Therefore, the Verifier' s code and configuration, especially those of the CoRIM processor, are primary security assets that must be built and maintained as securely as possible.

The protection of the Verifier system should be considered throughout its entire lifecycle, from design to operation. This includes the following aspects:

* Minimizing implementation complexity (see also Section 6.1 of [I-D.ietf-rats-endorsements]);

* Using memory-safe programming languages;

* Using secure defaults;

* Minimizing the attack surface by avoiding unnecessary features that could be exploited by attackers;

* Applying the principle of least privilege to the system's users;

* Minimizing the potential impact of security breaches by implementing separation of duties in both the software and operational architecture;

* Conducting regular, automated audits and reviews of the system, such as ensuring that users' privileges are correctly configured and that any new code has been audited and approved by independent parties;

* Failing securely in the event of errors to avoid compromising the security of the system.

It is critical that appraisal procedures are auditable and reproducible. The integrity of code and data during execution is an explicit objective, for example, ensuring that the appraisal functions are executed in an attestable trusted execution environment (TEE).

Please review the Security and Privacy Considerations in Sections 8 and 9 of [I-D.ietf-rats-endorsements]; these considerations apply to this document as well.

12. IANA Considerations

12.1. New COSE Header Parameters

12.2. 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	Reserved for backward compatibility	RFCthis
501	map	A tagged-unsigned-corim-map, see Section 4.1	RFCthis
502-504	any	Earmarked for CoRIM	RFCthis
505	bytes	A tagged-concise-swid-tag, see Section 4.1.2	RFCthis
506	bytes	A tagged-concise-mid-tag, see Section 4.1.2	RFCthis

507	any	Earmarked for CoRIM	RFCthis
508	bytes	A tagged-concise-tl-tag, see Section 4.1.2	RFCthis
509-549	any	Earmarked for CoRIM	RFCthis
550	bytes .size (7..33)	tagged-ueid-type, see Section 7.5	RFCthis
552	uint	tagged-svn, see Section 5.1.4.5.4	RFCthis
553	uint	tagged-min-svn, see Section 5.1.4.5.4	RFCthis
554	text	tagged-pkix-base64-key-type, see Section 5.1.4.6	RFCthis
555	text	tagged-pkix-base64-cert-type, see Section 5.1.4.6	RFCthis
556	text	tagged-pkix-base64-cert-path-type, see Section 5.1.4.6	RFCthis
557	[int/text, bytes]	tagged-key-thumbprint-type, see Section 7.7	RFCthis
558	COSE_Key	tagged-cose-key-type, see Section 5.1.4.6	RFCthis
559	digest	tagged-cert-thumbprint-type, see Section 5.1.4.6	RFCthis
560	bytes	tagged-bytes, see Section 7.8	RFCthis
561	digest	tagged-cert-path-thumbprint-type, see Section 5.1.4.6	RFCthis
562	bytes	tagged-pkix-asn1der-cert-type, see Section 5.1.4.6	RFCthis
563	tagged-masked-raw-	tagged-masked-raw-value, see Section 5.1.4.5.6	RFCthis

	value		
564	array	tagged-int-range, see Section 5.1.4.8	RFCthis
565-599	any	Earmarked for CoRIM	RFCthis

Table 9

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

12.3. CoRIM Map Registry

This document defines a new registry titled "CoRIM Map". The registry uses integer values as index values for items in corim-map CBOR maps.

Future registrations for this registry are to be made based on [RFC8126] as follows:

Range	Registration Procedures
0-127	Standards Action
128-255	Specification Required

Table 10: CoRIM Map Items Registration Procedures

All negative values are reserved for Private Use.

Initial registrations for the "CoRIM Map" registry are provided below. Assignments consist of an integer index value, the item name, and a reference to the defining specification.

Index	Item Name	Specification
0	id	RFCthis
1	tags	RFCthis
2	dependent-rims	RFCthis
3	profile	RFCthis
4	rim-validity	RFCthis
5	entities	RFCthis
3-255	Unassigned	

Table 11: CoRIM Map Items Initial Registrations

12.4. CoRIM Entity Map Registry

This document defines a new registry titled "CoRIM Entity Map". The registry uses integer values as index values for items in corim-entity-map CBOR maps.

Future registrations for this registry are to be made based on [RFC8126] as follows:

Range	Registration Procedures
0-127	Standards Action
128-255	Specification Required

Table 12: CoRIM Entity Map Items Registration Procedures

All negative values are reserved for Private Use.

Initial registrations for the "CoRIM Entity Map" registry are provided below. Assignments consist of an integer index value, the item name, and a reference to the defining specification.

Index	Item Name	Value Type	Specification
0	entity-name	text	Name of the entity responsible for the actions of the role.
1	reg-id	uri	A URI associated with the organization that owns the entity name.
2	role	[+ role-type-choice]	A type choice defining the roles that the entity is claiming.
3-255	Unassigned		

Table 13: CoRIM Entity Map Items Initial Registrations

12.5. CoRIM Signer Map Registry

This document defines a new registry titled "CoRIM Signer Map". The registry uses integer values as index values for items in corim-signer-map CBOR maps.

Future registrations for this registry are to be made based on [RFC8126] as follows:

Range	Registration Procedures
0-127	Standards Action
128-255	Specification Required

Table 14: CoRIM Signer Map Items Registration Procedures

All negative values are reserved for Private Use.

Initial registrations for the "CoRIM Signer Map" registry are provided below. Assignments consist of an integer index value, the item name, and a reference to the defining specification.

Index	Item Name	Specification
0	signer-name	RFCthis
1	signer-uri	RFCthis
2-255	Unassigned	

Table 15: CoRIM Signer Map Items
Initial Registrations

12.6. CoMID Map Registry

This document defines a new registry titled "CoMID Map". The registry uses integer values as index values for items in concise-mid-tag CBOR maps.

Future registrations for this registry are to be made based on [RFC8126] as follows:

Range	Registration Procedures
0-127	Standards Action
128-255	Specification Required

Table 16: CoMID Map Items
Registration Procedures

All negative values are reserved for Private Use.

Initial registrations for the "CoMID Map" registry are provided below. Assignments consist of an integer index value, the item name, and a reference to the defining specification.

Index	Item Name	Specification
0	language	RFCthis
1	tag-identity	RFCthis
2	entity	RFCthis
3	linked-tags	RFCthis
4	triples	RFCthis
5-255	Unassigned	

Table 17: CoMID Map Items Initial Registrations

12.7. CoMID Entity Map Registry

This document defines a new registry titled "CoRIM Entity Map". The registry uses integer values as index values for items in corim-entity-map CBOR maps.

Future registrations for this registry are to be made based on [RFC8126] as follows:

Range	Registration Procedures
0-127	Standards Action
128-255	Specification Required

Table 18: CoMID Entity Map Items Registration Procedures

All negative values are reserved for Private Use.

Initial registrations for the "CoMID Entity Map" registry are provided below. Assignments consist of an integer index value, the item name, and a reference to the defining specification.

Index	Item Name	Value Type	Specification
0	entity-name	text	Name of the entity responsible for the actions of the role.
1	reg-id	uri	A URI associated with the organization that owns the entity name.
2	role	[+ role-type-choice]	A type choice defining the roles that the entity is claiming.
3-255	Unassigned		

Table 19: CoMID Entity Map Items Initial Registrations

12.8. CoMID Triples Map Registry

This document defines a new registry titled "CoMID Triples Map". The registry uses integer values as index values for items in the triples-map CBOR maps in concise-mid-tag codepoint 4.

Future registrations for this registry are to be made based on [RFC8126] as follows:

Range	Registration Procedures
0-1023	Standards Action
1024-65535	Specification Required
65536-18446744073709551616	First come first served

Table 20: CoMID Triples Map Items Registration Procedures

All negative values are reserved for Private Use.

Initial registrations for the "CoMID Triples Map" registry are provided below. Assignments consist of an integer index value, the item name, and a reference to the defining specification.

Index	Item Name	Specification
0	reference-triples	RFCthis
1	endorsed-triples	RFCthis
2	identity-triples	RFCthis
3	attest-key-triples	RFCthis
4	dependency-triples	RFCthis
5	membership-triples	RFCthis
6	coswid-triples	RFCthis
7	(reserved)	RFCthis
8	conditional-endorsement-series-triples	RFCthis
9	(reserved)	RFCthis
10	conditional-endorsement-triples	RFCthis
11-18446744073709551616	Unassigned	

Table 21: CoMID Triples Map Items Initial Registrations

12.9. CoMID Measurement Values Map Registry

This document defines a new registry titled "CoMID Measurement Values Map". The registry uses integer values as index values for items in multiple triples' representations.

Future registrations for this registry are to be made based on [RFC8126] as follows:

Range	Registration Procedures
0-1023	Standards Action
1024-65535	Specification Required
65536-18446744073709551616	First come first served

Table 22: CoMID Measurement Values Map Items Registration Procedures

All negative values are reserved for Private Use.

Initial registrations for the "CoMID Measurement Values Map" registry are provided below. Assignments consist of an integer index value, the item name, and a reference to the defining specification.

Index	Item Name	Specification
0	version	RFCthis
1	svn	RFCthis
2	digests	RFCthis
3	flags	RFCthis
4	raw-value	RFCthis
5	raw-value-mask-DEPRECATED	RFCthis
6	mac-addr	RFCthis
7	ip-addr	RFCthis
8	serial-number	RFCthis
9	ueid	RFCthis
10	uuid	RFCthis
11	name	RFCthis
12	(reserved)	RFCthis
13	cryptokeys	RFCthis
14	integrity-registers	RFCthis
15	int-range	RFCthis
16-18446744073709551616	Unassigned	

Table 23: Measurement Values Map Items Initial Registrations

12.10. CoMID Flags Map Registry

This document defines a new registry titled "CoMID Flags Map". The registry uses integer values as index values for items in measurement-values-map codepoint 3.

Future registrations for this registry are to be made based on [RFC8126] as follows:

Range	Registration Procedures
0-1023	Standards Action
1024-65535	Specification Required
65536-18446744073709551616	First come first served

Table 24: CoMID Flags Map Items Registration Procedures

All negative values are reserved for Private Use.

Initial registrations for the "CoMID Measurement Values Map" registry are provided below. Assignments consist of an integer index value, the item name, and a reference to the defining specification.

Index	Item Name	Specification
0	is-configured	RFCthis
1	is-secure	RFCthis
2	is-recovery	RFCthis
3	is-debug	RFCthis
4	is-replay-protected	RFCthis
5	is-integrity-protected	RFCthis
6	is-runtime-meas	RFCthis
7	is-immutable	RFCthis
8	is-tcb	RFCthis
9	is-confidentiality-protected	RFCthis
10-18446744073709551616	Unassigned	

Table 25: Flags Map Items Initial Registrations

12.11. CoTL Map Registry

This document defines a new registry titled "CoTL Map". The registry uses integer values as index values for items in 'concise-tl-tag' CBOR maps.

Future registrations for this registry are to be made based on [RFC8126] as follows:

Range	Registration Procedures
0-127	Standards Action
128-255	Specification Required

Table 26: CoTL Map Items Registration Procedures

All negative values are reserved for Private Use.

Initial registrations for the "CoTL Map" registry are provided below. Assignments consist of an integer index value, the item name, and a reference to the defining specification.

Index	Item Name	Specification
0	tag-identity	RFCthis
1	tags-list	RFCthis
2	tl-validity	RFCthis
3-255	Unassigned	

Table 27: CoTL Map Items Initial Registrations

12.12. New Media Types

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

Name	Template	Reference
rim+cbor	application/rim+cbor	RFCthis, (Section 12.12.1)
rim+cose	application/rim+cose	RFCthis, (Section 12.12.2)

Table 28: New Media Types

12.12.1. rim+cbor

Type name: application
Subtype name: rim+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 11 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
File extension(s): .corim
Macintosh file type code(s): n/a
Person and 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

12.12.2. rim+cose

Type name: application
Subtype name: rim+cose
Required parameters: n/a (cose-type is explicitly not supported, as it is understood to be "cose-sign1")
Optional parameters: "profile" (CoRIM profile in string format. OIDs MUST use the dotted-decimal notation.)
Encoding considerations: binary
Security considerations: Section 11 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 CoRIM payloads protected using COSE Sign1 over HTTP(S), CoAP(S), and other transports.
Fragment identifier considerations: n/a
Magic number(s): D2 84
File extension(s): .corim
Macintosh file type code(s): n/a
Person and 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

12.13. 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/rim+cbor	-	TBD1	RFCthis
application/rim+cose	-	TBD2	RFCthis

Table 29: New Content-Formats

13. References

13.1. Normative References

[I-D.ietf-cose-hash-envelope]
Steele, O., Lasker, S., and H. Birkholz, "COSE Hash Envelope", Work in Progress, Internet-Draft, draft-ietf-cose-hash-envelope-10, 15 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-hash-envelope-10>.

[I-D.ietf-rats-eat-measured-component]
Frost, S., Fossati, T., Tschofenig, H., and H. Birkholz, "Entity Attestation Token (EAT) Measured Component", Work in Progress, Internet-Draft, draft-ietf-rats-eat-measured-component-12, 20 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-measured-component-12>.

[I-D.ietf-rats-msg-wrap]
Birkholz, H., Smith, N., Fossati, T., Tschofenig, H., and D. Glaze, "RATS Conceptual Messages Wrapper (CMW)", Work in Progress, Internet-Draft, draft-ietf-rats-msg-wrap-23, 11 December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-msg-wrap-23>.

[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>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

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

[RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <https://www.rfc-editor.org/rfc/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://www.rfc-editor.org/rfc/rfc8174>.

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

[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://www.rfc-editor.org/rfc/rfc8610>.

[RFC9090] Bormann, C., "Concise Binary Object Representation (CBOR) Tags for Object Identifiers", RFC 9090, DOI 10.17487/RFC9090, July 2021, <https://www.rfc-editor.org/rfc/rfc9090>.

[RFC9164] Richardson, M. and C. Bormann, "Concise Binary Object Representation (CBOR) Tags for IPv4 and IPv6 Addresses and Prefixes", RFC 9164, DOI 10.17487/RFC9164, December 2021, <https://www.rfc-editor.org/rfc/rfc9164>.

[RFC9393] Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", RFC 9393, DOI 10.17487/RFC9393, June 2023, <https://www.rfc-editor.org/rfc/rfc9393>.

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/rfc/rfc9562>.

[RFC9597] Looker, T. and M.B. Jones, "CBOR Web Token (CWT) Claims in COSE Headers", RFC 9597, DOI 10.17487/RFC9597, June 2024, <https://www.rfc-editor.org/rfc/rfc9597>.

[RFC9711] Lundblade, L., Mandyam, G., O’Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", RFC 9711, DOI 10.17487/RFC9711, April 2025, <https://www.rfc-editor.org/rfc/rfc9711>.

[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://www.rfc-editor.org/rfc/rfc3986>.

[STD94] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[STD96] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[X.690] International Telephone and Telegraph Consultative Committee, "ASN.1 encoding rules: Specification of basic encoding Rules (BER), Canonical encoding rules (CER) and Distinguished encoding rules (DER)", CCITT Recommendation X.690, July 2002.

13.2. Informative References

[DICE.cert]
Trusted Computing Group, "DICE Certificate Profiles", Version 1.0, Revision 0.01 , July 2020, <https://trustedcomputinggroup.org/wp-content/uploads/DICE-Certificate-Profiles-r01_pub.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, "A CoRIM Profile for Arm’s Platform Security Architecture (PSA) Endorsements", Work in Progress, Internet-Draft, draft-fdb-rats-psa-endorsements-09, 5 January 2026, <https://datatracker.ietf.org/doc/html/draft-fdb-rats-psa-endorsements-09>.

[I-D.ietf-rats-ar4si]
Voit, E., Birkholz, H., Hardjono, T., Fossati, T., and V. Scarlata, "Attestation Results for Secure Interactions", Work in Progress, Internet-Draft, draft-ietf-rats-ar4si-09, 15 August 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-ar4si-09>.

[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-02, 5 December 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-concise-ta-stores-02>.

[I-D.ietf-rats-endorsements]
Thaler, D., Birkholz, H., and T. Fossati, "RATS Endorsements", Work in Progress, Internet-Draft, draft-ietf-rats-endorsements-09, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-endorsements-09>.

[IANA.coswid]
IANA, "Concise Software Identifier (CoSWID)", <https://www.iana.org/assignments/coswid>.

[IEEE-802.OandA]
"IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture", IEEE, DOI 10.1109/ieeestd.2014.6847097, ISBN ["9780738192192"], July 2014, <https://doi.org/10.1109/ieeestd.2014.6847097>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/rfc/rfc7519>.

[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://www.rfc-editor.org/rfc/rfc7942>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <https://www.rfc-editor.org/rfc/rfc9334>.

[RFC9783] Tschofenig, H., Frost, S., Brossard, M., Shaw, A., and T. Fossati, "Arm’s Platform Security Architecture (PSA) Attestation Token", RFC 9783, DOI 10.17487/RFC9783, June 2025, <https://www.rfc-editor.org/rfc/rfc9783>.

[TNC.Arch] Trusted Computing Group, "TCG Trusted Network Connect TNC Architecture for Interoperability", Specification Version 1.1 Revision 2 , May 2006, <https://trustedcomputinggroup.org/wp-content/uploads/TNC_Architecture_v1_1_r2.pdf>.

[TPM2.Part1] Trusted Computing Group, "Trusted Platform Module Library, Part 1: Architecture", Family "2.0", Level 00, Revision 01.83 , January 2024, <https://trustedcomputinggroup.org/resource/tpm-library-specification/>.

[W3C.rdf11-primer] "RDF 1.1 Primer", W3C NOTE rdf11-primer, W3C rdf11-primer, <https://www.w3.org/TR/rdf11-primer/>.

Appendix A. Base CoRIM CDDL

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

corim = concise-rim-type-choice
concise-rim-type-choice /= tagged-unsigned-corim-map / signed-corim
concise-tl-tag = {
  &(tag-identity: 0) => tag-identity-map,
  &(tags-list: 1) => [+ tag-identity-map],
  &(tl-validity: 2) => validity-map,
}
$concise-tag-type-choice /= tagged-concise-swid-tag / tagged-concise\
                                  -mid-tag / tagged-concise-tl-tag
corim-entity-map = entity-map<$corim-role-type-choice, $$corim-\
                                                 entity-map-extension>
$corim-id-type-choice /= tstr / uuid-type
corim-locator-map = {
  &(href: 0) => uri / [+ uri],
  ? &(thumbprint: 1) => eatmc.digest / [eatmc.digest],
}
corim-map = {
  &(id: 0) => $corim-id-type-choice,
  &(tags: 1) => [+ $concise-tag-type-choice],
  ? &(dependent-rims: 2) => [+ corim-locator-map],
  ? &(profile: 3) => $profile-type-choice,
  ? &(rim-validity: 4) => validity-map,
  ? &(entities: 5) => [+ corim-entity-map],
  * $$corim-map-extension,
}
unsigned-corim-map = corim-map
corim-meta-map = {
```

```
    &(signer: 0) => corim-signer-map,
    ? &(signature-validity: 1) => validity-map,
  }
  $corim-role-type-choice /= &(manifest-creator: 1) / &(manifest-\
                                                          signer: 2)
  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-unsigned-corim-map / hash-envelope-\
                                                      digest / nil,
    signature: bstr,
  ]
  hash-envelope-digest = bstr
  $profile-type-choice /= uri / tagged-oid-type
  cwt-claims = {
    &(iss: 1) => tstr,
    ? &(sub: 2) => tstr,
    ? &(exp: 4) => int / float,
    ? &(nbf: 5) => int / float,
    * int => any,
  }
  protected-corim-header-map = protected-corim-header-map-inline / \
                               protected-corim-header-map-hash-envelope
  protected-corim-header-map-inline = {
    &(alg: 1) => int,
    &(content-type: 3) => "application/rim+cbor",
    meta-group,
    * cose-label => cose-value,
  }
  protected-corim-header-map-hash-envelope = {
    &(alg: 1) => int,
    &(payload_hash_alg: 258) => int,
    &(payload_preimage_content_type: 259) => "application/rim+cbor",
    ? &(payload_location: 260) => tstr,
    meta-group,
    * cose-label => cose-value,
  }
  meta-group = ((
          corim-meta-identity,
          ? cwt-claims-identity,
        ) // cwt-claims-identity)
  corim-meta-identity = (&(corim-meta: 8) => bstr .cbor corim-meta-map)
  cwt-claims-identity = (&(CWT-Claims: 15) => cwt-claims)
```

```
signed-corim = #6.18(COSE-Sign1-corim)
tagged-concise-swid-tag = #6.505(bytes .cbor coswid.concise-swid-tag)
tagged-concise-mid-tag = #6.506(bytes .cbor concise-mid-tag)
tagged-concise-tl-tag = #6.508(bytes .cbor concise-tl-tag)
tagged-unsigned-corim-map = #6.501(unsigned-corim-map)
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,
  * $$concise-mid-tag-extension,
}
attest-key-triple-record = [
  environment: environment-map,
  key-list: [+ $crypto-key-type-choice],
  ? conditions: non-empty<{
            ? &(mkey: 0) => $measured-element-type-choice,
            ? &(authorized-by: 1) => [+ $crypto-key-type-choice],
}>,
]
$class-id-type-choice /= tagged-oid-type / tagged-uuid-type / tagged\
                                                              -bytes
class-map = non-empty<{
    ? &(class-id: 0) => $class-id-type-choice,
    ? &(vendor: 1) => tstr,
    ? &(model: 2) => tstr,
    ? &(layer: 3) => uint,
    ? &(index: 4) => uint,
}>
comid-entity-map = entity-map<$comid-role-type-choice, $$comid-\
                                                 entity-map-extension>
$comid-role-type-choice /= &(tag-creator: 0) / &(creator: 1) / &(\
                                                       maintainer: 2)
conditional-endorsement-series-triple-record = [
  condition: [
        environment: environment-map,
        claims-list: [* measurement-map],
        ? authorized-by: [+ $crypto-key-type-choice],
],
  series: [+ conditional-series-record],
]
conditional-series-record = [
```

```
    selection: [+ measurement-map],
    addition: [+ measurement-map],
  ]
  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,
    [+ coswid.tag-id],
  ]
  $crypto-key-type-choice /= tagged-pkix-base64-key-type / tagged-pkix\
  -base64-cert-type / tagged-pkix-base64-cert-path-type / tagged-cose-\
  key-type / tagged-pkix-asn1der-cert-type / tagged-key-thumbprint-\
  type / tagged-cert-thumbprint-type / tagged-cert-path-thumbprint-\
                                                  type / tagged-bytes
  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-key-thumbprint-type = #6.557(eatmc.digest)
  tagged-cose-key-type = #6.558(COSE_Key)
  tagged-cert-thumbprint-type = #6.559(eatmc.digest)
  tagged-cert-path-thumbprint-type = #6.561(eatmc.digest)
  tagged-pkix-asn1der-cert-type = #6.562(bstr)
  domain-dependency-triple-record = [
    domain-id: domain-type,
    trustees: [+ domain-type],
  ]
  domain-membership-triple-record = [
    domain-id: domain-type,
    members: [+ domain-type],
  ]
  conditional-endorsement-triple-record = [
    conditions: [+ stateful-environment-record],
    endorsements: [+ endorsed-triple-record],
  ]
  domain-type = environment-map
  endorsed-triple-record = [
    condition: environment-map,
    endorsement: [+ 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 = non-empty<{
      ? &(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 / tagged-bytes
  identity-triple-record = [
    environment: environment-map,
    key-list: [+ $crypto-key-type-choice],
    ? conditions: non-empty<{
              ? &(mkey: 0) => $measured-element-type-choice,
              ? &(authorized-by: 1) => [+ $crypto-key-type-choice],
  }>,
  ]
  $instance-id-type-choice /= tagged-ueid-type / tagged-uuid-type / \
  tagged-bytes / tagged-pkix-base64-key-type / tagged-pkix-base64-cert\
  -type / tagged-cose-key-type / tagged-key-thumbprint-type / tagged-\
                  cert-thumbprint-type / tagged-pkix-asn1der-cert-type
  ip-addr-type-choice /= cbor-ip.ipv4-address / cbor-ip.ipv6-address
  int-range-type-choice = int / tagged-int-range
  int-range = [
    min: int / negative-inf,
    max: int / positive-inf,
  ]
  tagged-int-range = #6.564(int-range)
  positive-inf = null
  negative-inf = null
  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 / tagged-uuid-type \
                                                   / uint / tstr
  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) => digests-type,
      ? &(flags: 3) => flags-map,
      ? (
                    &(raw-value: 4) => $raw-value-type-choice,
                    ? &(raw-value-mask-DEPRECATED: 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: 13) => [+ $crypto-key-type-choice],
      ? &(integrity-registers: 14) => integrity-registers,
      ? &(int-range: 15) => int-range-type-choice,
      * $$measurement-values-map-extension,
  }>
  non-empty<M> = M .and ({+ any => any})
  oid-type = bytes
  tagged-oid-type = #6.111(oid-type)
  $raw-value-type-choice /= tagged-bytes / tagged-masked-raw-value
  raw-value-mask-type = bytes
  tagged-masked-raw-value = #6.563([
      value: bytes,
      mask: bytes,
  ])
  reference-triple-record = [
    ref-env: environment-map,
    ref-claims: [+ measurement-map],
  ]
  stateful-environment-record = [
    environment: environment-map,
```

```
    claims-list: [+ 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 = svn / tagged-svn / tagged-min-svn
  $tag-id-type-choice /= tstr / uuid-type
  tag-identity-map = {
    &(tag-id: 0) => $tag-id-type-choice,
    ? &(tag-version: 1) => tag-version-type,
  }
  $tag-rel-type-choice /= &(supplements: 0) / &(replaces: 1)
  tag-version-type = uint .default 0
  tagged-bytes = #6.560(bytes)
  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: 10) => [+ conditional-\
                                        endorsement-triple-record],
      * $$triples-map-extension,
  }>
  ueid-type = bytes .size (7 .. 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) => coswid.$version-scheme,
  }
  digests-type = [+ eatmc.digest]
  integrity-register-id-type-choice = uint / text
  integrity-registers = {+ integrity-register-id-type-choice => \
                                                        digests-type}
  eatmc.measured-component = {
    eatmc.component-id-label => eatmc.component-id,
    eatmc.measurement,
    ? eatmc.authorities-label => [+ eatmc.authority-id-type],
```

```
    ? eatmc.flags-label => eatmc.flags-type,
  }
  eatmc.measurement //= (eatmc.digested-measurement-label => eatmc.\
                        digest // eatmc.raw-measurement-label => bytes)
  eatmc.authority-id-type = eatmc.eat.JC<eatmc.bytes-b64u, bytes>
  eatmc.flags-type = eatmc.eat.JC<eatmc.bytes8-b64u, eatmc.bytes8>
  eatmc.component-id = [
    name: text,
    ? version: eatmc.version,
  ]
  eatmc.version = [
    val: text,
    ? scheme: eatmc.coswid.$version-scheme,
  ]
  eatmc.digest = [
    alg: int / text,
    val: eatmc.digest-value-type,
  ]
  eatmc.digest-value-type = eatmc.eat.JC<eatmc.bytes-b64u, bytes>
  eatmc.bytes-b64u = text .b64u bytes
  eatmc.bytes8 = bytes .size 8
  eatmc.bytes8-b64u = text .b64u eatmc.bytes8
  eatmc.component-id-label = eatmc.eat.JC<"id", 1>
  eatmc.digested-measurement-label = eatmc.eat.JC<"digested-\
                                                  measurement", 2>
  eatmc.raw-measurement-label = eatmc.eat.JC<"raw-measurement", 5>
  eatmc.authorities-label = eatmc.eat.JC<"authorities", 3>
  eatmc.flags-label = eatmc.eat.JC<"flags", 4>
  eatmc.mc-cbor = bytes .cbor eatmc.measured-component
  eatmc.mc-json = text .json eatmc.measured-component
  eatmc.$measurements-body-cbor /= eatmc.mc-cbor / eatmc.mc-json
  eatmc.$measurements-body-json /= eatmc.mc-json / text .b64u eatmc.mc\
                                                             -cbor
  eatmc.eat.JSON-ONLY<J> = J .feature "json"
  eatmc.eat.CBOR-ONLY<C> = C .feature "cbor"
  eatmc.eat.JC<J, C> = eatmc.eat.JSON-ONLY<J> / eatmc.eat.CBOR-ONLY<C>
  eatmc.coswid.$version-scheme /= eatmc.coswid.multipartnumeric / \
  eatmc.coswid.multipartnumeric-suffix / eatmc.coswid.alphanumeric / \
                eatmc.coswid.decimal / eatmc.coswid.semver / int / text
  eatmc.coswid.multipartnumeric = 1
  eatmc.coswid.multipartnumeric-suffix = 2
  eatmc.coswid.alphanumeric = 3
  eatmc.coswid.decimal = 4
  eatmc.coswid.semver = 16384
  coswid.concise-swid-tag = {
    coswid.tag-id => text / bstr .size 16,
    coswid.tag-version => integer,
    ? coswid.corpus => bool,
```

```
    ? coswid.patch => bool,
    ? coswid.supplemental => bool,
    coswid.software-name => text,
    ? coswid.software-version => text,
    ? coswid.version-scheme => coswid.$version-scheme,
    ? coswid.media => text,
    ? coswid.software-meta => coswid.one-or-more<coswid.software-meta-\
                                                          entry>,
    coswid.entity => coswid.one-or-more<coswid.entity-entry>,
    ? coswid.link => coswid.one-or-more<coswid.link-entry>,
    ? coswid.payload-or-evidence,
    * $$coswid-extension,
    coswid.global-attributes,
  }
  coswid.tag-id = 0
  coswid.$version-scheme /= coswid.multipartnumeric / coswid.\
  multipartnumeric-suffix / coswid.alphanumeric / coswid.decimal / \
                                          coswid.semver / int / text
  coswid.one-or-more<T> = T / [2*T]
  coswid.tag-version = 12
  coswid.corpus = 8
  coswid.patch = 9
  coswid.supplemental = 11
  coswid.software-name = 1
  coswid.software-version = 13
  coswid.version-scheme = 14
  coswid.media = 10
  coswid.software-meta = 5
  coswid.software-meta-entry = {
    ? coswid.activation-status => text,
    ? coswid.channel-type => text,
    ? coswid.colloquial-version => text,
    ? coswid.description => text,
    ? coswid.edition => text,
    ? coswid.entitlement-data-required => bool,
    ? coswid.entitlement-key => text,
    ? coswid.generator => text / bstr .size 16,
    ? coswid.persistent-id => text,
    ? coswid.product => text,
    ? coswid.product-family => text,
    ? coswid.revision => text,
    ? coswid.summary => text,
    ? coswid.unspsc-code => text,
    ? coswid.unspsc-version => text,
    * $$software-meta-extension,
    coswid.global-attributes,
  }
  coswid.entity = 2
```

```
   coswid.entity-entry = {
     coswid.entity-name => text,
     ? coswid.reg-id => coswid.any-uri,
     coswid.role => coswid.one-or-more<coswid.$role>,
     ? coswid.thumbprint => coswid.hash-entry,
     * $$entity-extension,
     coswid.global-attributes,
   }
   coswid.link = 4
   coswid.link-entry = {
     ? coswid.artifact => text,
     coswid.href => coswid.any-uri,
     ? coswid.media => text,
     ? coswid.ownership => coswid.$ownership,
     coswid.rel => coswid.$rel,
     ? coswid.media-type => text,
     ? coswid.use => coswid.$use,
     * $$link-extension,
     coswid.global-attributes,
   }
   coswid.payload-or-evidence //= (coswid.payload => coswid.payload-\
                       entry // coswid.evidence => coswid.evidence-entry)
   coswid.global-attributes = (
     ? coswid.lang => text,
     * coswid.any-attribute,
     )
   coswid.multipartnumeric = 1
   coswid.multipartnumeric-suffix = 2
   coswid.alphanumeric = 3
   coswid.decimal = 4
   coswid.semver = 16384
   coswid.activation-status = 43
   coswid.channel-type = 44
   coswid.colloquial-version = 45
   coswid.description = 46
   coswid.edition = 47
   coswid.entitlement-data-required = 48
   coswid.entitlement-key = 49
   coswid.generator = 50
   coswid.persistent-id = 51
   coswid.product = 52
   coswid.product-family = 53
   coswid.revision = 54
   coswid.summary = 55
   coswid.unspsc-code = 56
   coswid.unspsc-version = 57
   coswid.entity-name = 31
   coswid.reg-id = 32
```

```
   coswid.any-uri = uri
   coswid.role = 33
   coswid.$role /= coswid.tag-creator / coswid.software-creator / \
   coswid.aggregator / coswid.distributor / coswid.licensor / coswid.\
                                                 maintainer / int / text
   coswid.thumbprint = 34
   coswid.hash-entry = [
     hash-alg-id: int,
     hash-value: bytes,
   ]
   coswid.artifact = 37
   coswid.href = 38
   coswid.ownership = 39
   coswid.$ownership /= coswid.shared / coswid.private / coswid.\
                                                       abandon / int / text
   coswid.rel = 40
   coswid.$rel /= coswid.ancestor / coswid.component / coswid.feature \
   / coswid.installationmedia / coswid.packageinstaller / coswid.\
   parent / coswid.patches / coswid.requires / coswid.see-also / coswid\
                .supersedes / coswid.supplemental / -256 .. 65536 / text
   coswid.media-type = 41
   coswid.use = 42
   coswid.$use /= coswid.optional / coswid.required / coswid.\
                                                 recommended / int / text
   coswid.payload = 6
   coswid.payload-entry = {
     coswid.resource-collection,
     * $$payload-extension,
     coswid.global-attributes,
   }
   coswid.evidence = 3
   coswid.evidence-entry = {
     coswid.resource-collection,
     ? coswid.date => coswid.integer-time,
     ? coswid.device-id => text,
     ? coswid.location => text,
     * $$evidence-extension,
     coswid.global-attributes,
   }
   coswid.lang = 15
   coswid.any-attribute = (coswid.label => coswid.one-or-more<text> / \
                                                 coswid.one-or-more<int>)
   coswid.tag-creator = 1
   coswid.software-creator = 2
   coswid.aggregator = 3
   coswid.distributor = 4
   coswid.licensor = 5
   coswid.maintainer = 6
```

```
   coswid.shared = 3
   coswid.private = 2
   coswid.abandon = 1
   coswid.ancestor = 1
   coswid.component = 2
   coswid.feature = 3
   coswid.installationmedia = 4
   coswid.packageinstaller = 5
   coswid.parent = 6
   coswid.patches = 7
   coswid.requires = 8
   coswid.see-also = 9
   coswid.supersedes = 10
   coswid.optional = 1
   coswid.required = 2
   coswid.recommended = 3
   coswid.resource-collection = (
     coswid.path-elements-group,
     ? coswid.process => coswid.one-or-more<coswid.process-entry>,
     ? coswid.resource => coswid.one-or-more<coswid.resource-entry>,
     * $$resource-collection-extension,
     )
   coswid.date = 35
   coswid.integer-time = #6.1(int)
   coswid.device-id = 36
   coswid.location = 23
   coswid.label = text / int
   coswid.path-elements-group = (
     ? coswid.directory => coswid.one-or-more<coswid.directory-entry>,
     ? coswid.file => coswid.one-or-more<coswid.file-entry>,
     )
   coswid.process = 18
   coswid.process-entry = {
     coswid.process-name => text,
     ? coswid.pid => integer,
     * $$process-extension,
     coswid.global-attributes,
   }
   coswid.resource = 19
   coswid.resource-entry = {
     coswid.type => text,
     * $$resource-extension,
     coswid.global-attributes,
   }
   coswid.directory = 16
   coswid.directory-entry = {
     coswid.filesystem-item,
     ? coswid.path-elements => {coswid.path-elements-group},
```

```
    * $$directory-extension,
    coswid.global-attributes,
  }
  coswid.file = 17
  coswid.file-entry = {
    coswid.filesystem-item,
    ? coswid.size => uint,
    ? coswid.file-version => text,
    ? coswid.hash => coswid.hash-entry,
    * $$file-extension,
    coswid.global-attributes,
  }
  coswid.process-name = 27
  coswid.pid = 28
  coswid.type = 29
  coswid.filesystem-item = (
    ? coswid.key => bool,
    ? coswid.location => text,
    coswid.fs-name => text,
    ? coswid.root => text,
    )
  coswid.path-elements = 26
  coswid.size = 20
  coswid.file-version = 21
  coswid.hash = 7
  coswid.key = 22
  coswid.fs-name = 24
  coswid.root = 25
  cbor-ip.ipv4-address = bytes .size 4
  cbor-ip.ipv6-address = bytes .size 16
```

Acknowledgments

The authors would like to thank the following people for their review and comments on this document: Carl Wallace Hannes Tschofenig Steven Bellock Jag Raman Giri Mandyam Jeremy O’Donoghue and Michael Richardson

Contributors

Carsten Bormann
Universitt 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.

Andrew Draper
Altera
Email: andrew.draper@altera.com

Andrew contributed the concept, description, and semantics of conditional endorsements as well as consistent contribution to weekly reviews of others' edits.

Dionna Glaze
Google LLC
Email: dionnaglaze@google.com

Dionna contributed many clarifying questions and disambiguations to the semantics of attestation appraisal as well as consistent contribution to weekly reviews of others' edits.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Email: henk.birkholz@ietf.contact

Thomas Fossati
Linaro
Email: Thomas.Fossati@linaro.org

Yogesh Deshpande
arm
Email: yogesh.deshpande@arm.com

Ned Smith
Independent
Email: ned.smith.ietf@outlook.com

Wei Pan
Huawei Technologies
Email: william.panwei@huawei.com